

STAIRS — Understanding and Developing Specifications Expressed as UML Interaction Diagrams

Doctoral Dissertation by

Ragnhild Kobro Runde

Submitted to the Faculty of Mathematics and Natural
Sciences at the University of Oslo in partial
fulfillment of the requirements for
the degree Dr. Scient. in Computer Science

January 2007

Abstract

STAIRS is a method for the step-wise, compositional development of interactions in the setting of UML 2.x. UML 2.x interactions, such as sequence diagrams and interaction overview diagrams, are seen as intuitive ways of describing communication between different parts of a system, and between a system and its users.

STAIRS addresses the challenges of harmonizing intuition and formal reasoning by providing a precise understanding of the partial nature of interactions, and of how this kind of incomplete specifications may be consistently refined into more complete specifications.

For understanding individual interaction diagrams, STAIRS defines a denotational trace semantics for the main constructs of UML 2.x interactions. The semantic model takes into account the partiality of interactions, and the formal semantics of STAIRS is faithful to the informal semantics given in the UML 2.x standard. For developing UML 2.x interactions, STAIRS defines a number of refinement relations corresponding to basic system development steps. STAIRS also defines matching compliance relations, for relating interactions to real computer systems.

An important feature of STAIRS is the distinction between underspecification and inherent nondeterminism. Underspecification means that there are several possible behaviours serving the same overall purpose, and that it is sufficient for a computer system to perform only one of these. On the other hand, inherent nondeterminism is used to capture alternative behaviours that must all be possible for an implementation. A typical example is the tossing of a coin, where both heads and tails should be possible outcomes. In some cases, using inherent nondeterminism may also be essential for ensuring the necessary security properties of a system.

Acknowledgements

First of all, I thank Ketil Stølen and Øystein Haugen for being my supervisors on this thesis work. You have been excellent supervisors, and I am grateful for your invaluable guidance, encouragement and our countless scientific discussions. Thanks also for inviting me to participate in the SARDAS project, which has provided a friendly and challenging environment for discussing my work. I thank everyone involved in the SARDAS project, including the Ph.D. students, senior researchers, guest scientists and advisory board. In particular, I thank Knut Eilif Husa for our collaboration on the early work on STAIRS, and Atle Refsdal for a number of interesting discussions and for close collaboration during the last year. I thank Manfred Broy for inviting me to Munich, and María Victoria Cengarle and Alexander Knapp for interesting discussions during my visit there. I thank the people at the Department for Cooperative and Trusted Systems at SINTEF for welcoming me at several occasions, and for providing valuable feedback on parts of my work. Thanks are also due to many other researchers who have crossed my path over the years, in person or in writing. I am grateful to the Department of Informatics at the University of Oslo for giving me the opportunity to do this thesis work, and I thank all of my colleagues for making this a wonderful place to be. In particular, I thank Anders Moen Hagalisse, Einar Broch Johnsen and Olaf Owe for various kinds of support over the years. I thank the staff at the Informatics Library for providing me with the necessary background literature, and Dag Langmyhr for helping me with my L^AT_EX problems. Finally, my greatest thanks go to Hans Arild for continuous support through all of these years.

Contents

Abstract	iii
Acknowledgements	v
I Overview	1
1 Introduction	3
1.1 UML Interactions and STAIRS	3
1.2 Contribution and Overview of the Thesis	4
2 Research Method	7
2.1 Problem Analysis	7
2.2 Innovation	8
2.3 Evaluation	8
3 Problem Analysis	11
3.1 Problem Specification	11
3.2 Limiting the Scope of this Thesis	12
3.2.1 UML Diagrams	12
3.2.2 Model Relations	16
3.3 Goals and Success Criteria	16
4 State of the Art	19
4.1 Semantics	19
4.2 Time	20
4.3 Nondeterminism	21
4.4 Refinement	22
4.5 UML-Related Methodologies	23
4.6 Message Sequence Charts	25
5 STAIRS	27
5.1 The First STAIRS	27
5.2 STAIRS Today	28
5.2.1 Syntax	28
5.2.2 Semantics	30

5.2.3	Refinement	32
6	Overview of the Papers	35
7	Discussion	39
7.1	The Original Semantic Model of STAIRS	39
7.1.1	Semantic Model	39
7.1.2	Refinement	40
7.1.3	Definition of <code>xalt</code>	41
7.2	Evaluating STAIRS with Respect to the Success Criteria	42
7.3	Generalization of the Results	44
7.4	Related Work	45
7.4.1	Harald Störrle: Trace Semantics of Interactions in UML 2.0	45
7.4.2	María Victoria Cengarle and Alexander Knapp: UML 2.0 Interactions — Semantics and Refinement	46
7.4.3	Other Work on UML 2.x Interactions	47
8	Future Work	49
Bibliography		51
II	Research Papers	57
9	STAIRS towards Formal Design with Sequence Diagrams	59
9.1	Introduction	60
9.2	Requirements to STAIRS	61
9.3	How STAIRS Meets the Requirements	62
9.3.1	Spelling out the Trace Semantics of UML 2.0	62
9.3.2	Capturing Positive Behavior	65
9.3.3	Capturing Negative Behavior	67
9.3.4	Distinguishing Mandatory from Potential Behavior	69
9.4	STAIRS Spelled out: Supplementing	70
9.5	STAIRS Spelled out: Narrowing	70
9.6	STAIRS Spelled out: Detailing	71
9.7	Formal Foundation	71
9.7.1	Representing Runs by Traces	72
9.7.2	Semantics of Sequence Diagrams	73
9.7.3	Refinement	75
9.8	Conclusions	77
9.8.1	Related Work	78
	References	80
10	Why Timed Sequence Diagrams Require Three-Event Semantics	83
10.1	Introduction to STAIRS	84
10.2	Motivating Timed STAIRS	85
10.3	Formal Foundation	87

10.3.1	Mathematical Background on Sequences	87
10.3.2	Syntax of Sequence Diagrams	88
10.3.3	Representing Executions by Traces	90
10.3.4	Interaction Obligations	91
10.3.5	Semantics of Sequence Diagrams	93
10.4	Two Abstractions	95
10.4.1	Standard Interpretation	96
10.4.2	Black-Box Interpretation	97
10.5	The General Case	97
10.6	Refinement	100
10.6.1	Definition of Glass-Box Refinement	100
10.6.2	Supplementing and Narrowing	101
10.6.3	Example of Glass-Box Refinement	101
10.6.4	Definition of Black-Box Refinement	101
10.6.5	Example of Black-Box Refinement	103
10.6.6	Detailed	103
10.6.7	Refinement Through Time Constraints	104
10.7	Conclusions and Related Work	105
References	107
10.A	Extending STAIRS to Handle Gates	108
10.A.1	Syntax	108
10.A.2	Semantics	109
10.A.3	Example	110
10.B	Trace Completeness	112
10.C	Associativity, Commutativity and Distributivity	113
10.C.1	Lemmas on Well-Formedness	114
10.C.2	Lemmas on Trace Sets	115
10.C.3	Lemmas on Interaction Obligations	127
10.C.4	Lemmas on Sets of Interaction Obligations	130
10.C.5	Theorems on Sequence Diagram Operators	133
10.D	Reflexivity and Transitivity	134
10.D.1	Reflexivity	134
10.D.2	Transitivity	135
10.E	Monotonicity	136
10.E.1	Monotonicity of \rightsquigarrow_r	137
10.E.2	Monotonicity of \rightsquigarrow_g with Respect to Operators on Sets of Interaction Obligations	142
10.E.3	Monotonicity of \rightsquigarrow_g with Respect to the Sequence Diagram Operators .	149
11	How to Transform UML neg into a Useful Construct	155
11.1	Introduction	156
11.2	Background	156
11.2.1	Interactions and Trace Semantics	156
11.2.2	Refinement	158
11.3	Alternative Definitions of neg	158
11.3.1	Alternative a: The Positive Traces of neg d Are the Negative Traces of d .	159

11.3.2 Alternative b: The Positive Traces of $\text{neg } d$ Are All Traces Except the Negative Traces of $\text{neg } d$	161
11.3.3 Alternative c: $\text{neg } d$ Has No Positive Traces	160
11.3.4 Alternative d: The Only Positive Trace for $\text{neg } d$ Is the Empty Trace	162
11.3.5 Suggested Solution	163
11.4 Related Work	163
11.5 Conclusions	165
References	165
11.A Proofs	166
12 Refining UML Interactions with Underspecification and Nondeterminism	169
12.1 Introduction	170
12.2 Background: UML Interactions with Denotational Trace Semantics	171
12.2.1 Representing Executions by Traces	172
12.2.2 Syntax of Interactions	172
12.2.3 Semantics of Interactions	174
12.3 STAIRS and Nondeterminism	179
12.4 Extending STAIRS with Data and Guards	181
12.4.1 Data	182
12.4.2 Assignment	182
12.4.3 Constraints (State Invariants)	183
12.4.4 Guards (Interaction Constraints)	185
12.5 Refinement	187
12.5.1 Background: Formal Definitions	187
12.5.2 Adding Positive Behaviour	189
12.5.3 Adding Negative Behaviour	192
12.5.4 Redefining Positive Behaviour as Negative	193
12.5.5 Adding More Details	195
12.6 Implementation	197
12.7 Conclusions	199
12.7.1 Related Work	199
References	200
12.A Refinement by Adding Assignments and Constraints	201
12.B Identity of skip	202
12.C Comparing the Guarded and Unguarded Versions of alt and xalt	211
12.D Reflexivity and Transitivity of Limited Refinement	212
12.E Monotonicity Results	213
12.E.1 Interactions without Data	213
12.E.2 Interactions with Data	219
13 The Pragmatics of STAIRS	225
13.1 Introduction	226
13.2 The Semantic Model of STAIRS	226
13.3 The Pragmatics of Creating Interactions	229
13.3.1 The Use of alt Versus xalt	229
13.3.2 The Use of Guards	233

13.3.3 The Use of refuse, veto and assert	235
13.3.4 The Use of seq	237
13.4 The Pragmatics of Refining Interactions	240
13.4.1 The Use of Supplementing	241
13.4.2 The Use of Narrowing	242
13.4.3 The Use of Detailing	243
13.4.4 The Use of General Refinement	246
13.4.5 The Use of Limited Refinement	247
13.5 Related Work	248
13.6 Conclusions and Future Work	250
References	250
14 Underspecification, Inherent Nondeterminism and Probability in Sequence Diagrams	253
14.1 Introduction	254
14.2 Underspecification	255
14.2.1 Motivation	255
14.2.2 Semantic Representation	255
14.2.3 Refinement	256
14.2.4 Simple Example	256
14.2.5 Properties of alt and Refinement	257
14.3 Inherent Nondeterminism	257
14.3.1 Motivation	257
14.3.2 Semantic Representation	258
14.3.3 Refinement Revisited	258
14.3.4 Simple Example	258
14.3.5 Relating xalt to alt	260
14.3.6 Properties of xalt and Refinement	261
14.4 Probability	262
14.4.1 Motivation	262
14.4.2 Semantic Representation	262
14.4.3 Refinement Revisited	264
14.4.4 Simple Example	265
14.4.5 Relating palt to xalt and alt	265
14.4.6 Properties of alt, palt and Refinement	266
14.5 Related Work	267
14.6 Conclusion	268
References	269
14.A Proofs	270
15 Relating Computer Systems to Sequence Diagrams with Underspecification, Inherent Nondeterminism	
15.1 Introduction	276
15.2 Requirements	277
15.3 Sequence Diagrams and Trace Semantics	277
15.4 Relating Computer Systems to Sequence Diagrams with Underspecification	279
15.4.1 Refinement	280

15.4.2 Compliance	280
15.4.3 Example	281
15.5 Relating Computer Systems to Sequence Diagrams with Inherent Nondeterminism	283
15.5.1 Refinement	283
15.5.2 Compliance	284
15.5.3 Example	284
15.6 Results	286
15.7 Related Work	288
15.8 Conclusions	289
References	289
15.A Summary of Results	290
15.B Proofs	291
15.B.1 Specifications with Underspecification	291
15.B.2 Specifications with Inherent Nondeterminism	299
15.B.3 Correspondence	310
16 STAIRS Case Study: The BuddySync System	315
16.1 Introduction	316
16.2 Evaluation Criteria	316
16.3 Initial Description of the BuddySync System	317
16.4 Development Methodology	319
16.5 Iteration 1: RequestService and O erService	320
16.5.1 User Requirements: RequestService	320
16.5.2 User Requirements: O erService	323
16.5.3 System Specification: RequestService	324
16.5.4 System Specification: O erService	327
16.5.5 Finishing the Iteration	327
16.6 Iteration 2: RemoveRequest and RemoveO er	333
16.6.1 User Requirements: RemoveRequest	333
16.6.2 User Requirements: RemoveO er	333
16.6.3 System Specification: RemoveRequest	334
16.6.4 System Specification: RemoveO er	335
16.6.5 Finishing the Iteration	335
16.7 Iteration 3: SubscribeService and UnsubscribeService	336
16.7.1 User Requirements: SubscribeService	336
16.7.2 User Requirements: UnsubscribeService	336
16.7.3 System Specification: SubscribeService	336
16.7.4 System Specification: UnsubscribeService	337
16.7.5 System Specification: RequestService Updated	337
16.7.6 System Specification: O erService Updated	340
16.7.7 Finishing the Iteration	340
16.8 Discussion	342
16.8.1 Validating the Specification	342
16.8.2 Evaluating STAIRS	342
16.9 Conclusions	345

References	346
16.A Guidelines from “The Pragmatics of STAIRS”	347
16.A.1 The Pragmatics of Creating Interations	347
16.A.2 The Pragmatics of Refining Interactions	348

Part I

Overview

Chapter 1

Introduction

This thesis presents work on the STAIRS method, a method for the step-wise, compositional development of interactions in the setting of UML. This chapter gives a short introduction to the thesis work, together with an overview of the thesis.

1.1 UML Interactions and STAIRS

During the past decade, UML has become the de facto modelling standard used in industry. From being an approach unifying the leading modelling languages at the time, UML has gradually developed and is now in version 2.1 [OMG06].

UML 2.1 interactions, such as sequence diagrams and interaction overview diagrams, are seen as intuitive ways of describing communication between different parts (e.g. components or objects) of a system, and between a system and its users. According to the UML 2.1 standard [OMG06], an interaction describes a set of valid and a set of invalid traces, i.e. system behaviours. Interactions are usually incomplete specifications, meaning that there will typically be many traces that are not described by the interaction at all, and it is impossible to know whether these are valid or not.

A problem with UML 2.1 interactions is that their semantics is only explained in natural language, and for a given interaction it is often difficult, or even impossible, to know its precise meaning. Another aspect not addressed by the UML 2.1 standard is the relationships between different interactions for the same system, i.e. what it means for one interaction to be a refinement of another interaction, or for two interactions to describe the system from different viewpoints. Also, it is not defined what it means for a computer system to be in compliance with an interaction, i.e. when is a computer system a valid implementation of a specification in the form of UML 2.1 interactions. In particular, it is not clear whether the valid traces of an interaction describe behaviours that *must* or *may* be present in the final system.

As long as these aspects are not addressed properly, different people tend to interpret the same interaction differently. This leads to confusion and misunderstanding, where the end result might be that the systems being built are not the ones requested by the customers.

Another problem is the lack of tools supporting system development using UML 2.1 interactions. With a proper formal semantics, and with precise definitions of viewpoint correspondence, refinement and compliance, it is possible to make advanced tools for e.g. automatic analysis and

consistency checking. Errors are an inevitable part of any system development process, but with adequate tool support, fewer errors may be introduced during the development process, and the errors that are made may be discovered earlier, possibly resulting in substantially reduced development costs.

The STAIRS method presented in this thesis defines a denotational trace semantics for the main constructs of UML 2.1 interactions. The semantic model takes into account the partial nature of interactions, and the formal semantics of STAIRS corresponds closely to the informal semantics given in the UML 2.1 standard. STAIRS also defines a number of refinement relations for relating interactions made at different stages of the development process, and corresponding compliance relations.

Our vision is that the intuitive feeling of UML 2.1 interactions should be maintained while also providing the means for formal analysis such as security analysis, testing and automatic transformation into executable code.

1.2 Contribution and Overview of the Thesis

This thesis is organised as a collection of eight papers presenting work on STAIRS, together with an introductory part providing the context of this work. The organisation of the rest of this introductory part is as follows: In chapter 2 we present our research method, while chapter 3 gives a thorough problem analysis, presenting the setting of this thesis together with the goals and success criteria for the STAIRS method. Based on this problem analysis, in chapter 4 we discuss the state of the art that are relevant for this thesis work. Chapter 5 gives a summary of STAIRS, while chapter 6 provides a brief overview of the papers included in this thesis. Chapter 7 discusses the results obtained and recent related work on UML 2.1 interactions. Finally, ideas for future work is presented in chapter 8.

STAIRS, and the papers included in this thesis, is the result of a collaborative effort in a group of researchers led by Ketil Stølen and Øystein Haugen. In the following, we list the main contributions of STAIRS together with references to the papers where these are described. In chapter 6, the particular contributions of Ragnhild Kobro Runde are described for each of the eight papers in question.

- STAIRS improves the language of UML 2.1 interactions by providing additional mechanisms for
 - distinguishing between mandatory alternatives (e.g. inherent nondeterminism) and potential alternatives (e.g. underspecification). Described in: Papers 1, 2, 4, 5 and 6.
 - using two different negation operators depending on the desired positive behaviours. Described in: Paper 3.
 - distinguishing between the reception and the consumption of a message, an essential feature when working with time constraints. Described in: Paper 2.
- STAIRS provides a precise understanding of the partial nature of UML interactions by
 - defining a semantic model for UML 2.1 interactions with the extensions listed above. Described in: Paper 1.

- defining a denotational trace semantics for the most commonly used parts of UML 2.1 interactions, including time, guarded alternatives and the extensions listed above. Described in: Papers 1, 2, 3, 4 and 5.
- STAIRS supports stepwise and compositional development of interactions by defining basic refinement relations that
 - take the partiality of interactions into account, together with all the features mentioned above. Described in: Papers 1, 2, 4, 5 and 7.
 - are sound, meaning that the desirable mathematical properties of reflexivity, transitivity and monotonicity hold. Described in: Papers 2, 3, 4 and 7.
- STAIRS defines what it means for a computer system to be compliant with an interaction by
 - explaining how computer systems may be understood in terms of our semantic model. Described in: Paper 7.
 - defining sound compliance relations corresponding to the different refinement relations. Described in: Paper 7.
- STAIRS provides methodological guidelines for how to create and refine interactions using the main principles of STAIRS. Described in: Papers 5 and 8.

The research on STAIRS has been partly carried out within the context of the SARDAS project [SAR], which is funded by the Research Council of Norway under the IKT-2010 programme. The overall goal of SARDAS is to improve on state of the art for the specification, design and development of systems with high availability. The main goal of STAIRS has been to provide a firm foundation for the other activities to build on.

STAIRS has been successfully used both in other parts of the SARDAS project, and in related projects. Mass Soldal Lund has developed an operational semantics for STAIRS [LS06b] and used it for building a tool for test case generation from UML 2.1 interactions [LS06a]. Atle Refsdal and Knut Eilif Husa have developed probabilistic STAIRS [RHS05, RRS06], which extends STAIRS with probabilistic choice and soft real-time constraints. Fredrik Seehusen has used STAIRS for defining secure information flow property preserving refinement and transformation of interaction diagrams [SS06].

Chapter 2

Research Method

This chapter presents the research method on which this thesis work has been based.

Compared to most other sciences, computer science is a relatively new discipline and without an established research method [DC02]. Even the question of whether or not computer science qualifies as a science in the traditional sense is still being debated [Den05]. Computer science has ancestors in disciplines as diverse as mathematics, physics, engineering and social science. Consequently, researchers in computer science have to a varying degree adapted research methods used within each of these disciplines. However, computer science research is also often performed in an ad hoc manner and without a clear thought about research method [Gla95].

Much of computer science research, including this thesis work, fall into the category called technological research in [SS07]. While the aim of classical research in e.g. the natural and social sciences is to achieve more knowledge about some existing part of the world, the aim of technological research is to create new or improved artefacts. In computer science research, such artefacts may be e.g. programs, programming languages, security protocols, hardware processors, or methods as in our case.

Similar to classical research methods, the technological research method advocated in [SS07] is an iterative process consisting of three main steps: problem analysis, innovation and evaluation. These steps are very similar to the phases proposed in [Gla95]. Problem analysis corresponds to the informational phase, i.e. gathering or aggregating information. Innovation covers both the propositional and the analytical phase, i.e. proposing a hypothesis, method, etc, and exploring this proposition. Finally, evaluation corresponds to the evaluative phase, where the proposition is evaluated by e.g. experimentation or observation.

In the following sections, we describe how each of the steps in [SS07] have been instantiated in this thesis work.

2.1 Problem Analysis

First, as documented in chapter 3, we investigated the current situation in model-based system development using UML and identified the need for a new artefact — a method giving a precise semantics for UML 2.x, definitions of model relations and methodological guidelines for using UML 2.x for specifications. As this would be too much to cover within the scope of one thesis, we identified the STAIRS method for developing UML 2.x interactions as the artefact to be created

by this thesis work. Also, we formulated a number of requirements that should be fulfilled by STAIRS.

Next, as documented in chapter 4, we investigated existing theories and methods in order to evaluate to what extent these fulfilled our requirements for the formal framework. The main conclusion from this investigation was that these theories and methods were not sufficient, and that new research was needed in order to create the required framework.

2.2 Innovation

The innovation part of this thesis work has been to create the STAIRS method as a response to the identified need for a formal framework for UML 2.x interactions. A summary of STAIRS is given in chapter 5, and a more thorough description is given in the attached papers 1–7.

The development of STAIRS has been performed by treating the most basic parts of interactions first, and then iteratively adding more and more features. Similar to what often happens in iterative system development, the addition of new features to STAIRS has sometimes led to minor modifications of previous work. The choice of what new features to include in each step has been guided by feedback received when presenting our work to other researchers in the field, and by ourselves identifying weaknesses and additional needs when trying to use STAIRS on small toy examples.

Even though we have not reached all of the initial goals described in section 3.3, our claim is that STAIRS satisfies the main requirements and provides a useful basis for further research in this area.

2.3 Evaluation

In this thesis work, the evaluation has been performed alongside the development of the STAIRS method, and is documented in the attached papers. Also, a summarizing discussion may be found in chapter 7.

For evaluation, there exists a number of methods and techniques that may be categorized in different ways. In [McG84], a distinction is made between eight different evaluation methods, each with its own advantages and disadvantages. A perfect method leads to results that are both general, realistic and precise. However, [McG84] points out that no such perfect method exists, and that trying to increase one of these properties invariably results in reducing one or both of the other properties. The key, then, is to use different methods that complement each other. Several factors influence the exact choice of evaluation methods, including the time and resources required to carry out each of the methods. Also, the stated requirements are important as the chosen method must be able to both verify and falsify the claim that the artefact meets these requirements.

The STAIRS method consists mainly of a formal foundation for UML 2.x interactions, and it has therefore been natural to use formal theory and mathematical proofs for establishing desirable properties of this formalization. The main properties required of STAIRS are described in section 3.3, and proved in the attachments of papers 2, 4, 6 and 7.

For evaluating the usefulness of STAIRS, the results from formal theory have been supplemented with the use of STAIRS in several examples. Papers 1–7 all contain running examples

illustrating the main points. As STAIRS is meant to be a general method for UML 2.x interactions, these examples are on purpose taken from quite different domains as can be seen from the following overview:

Paper 1 describes an automatic teller machine, focusing on withdrawal of money.

Paper 2 elaborates on the original STAIRS example in [HS03], the making of dinner at an ethnic restaurant.

Paper 3 illustrates its main point using a small example with a vending machine selling tea and coffee.

Paper 4 uses an example of network communication.

Paper 5 is a tutorial paper using an appointment system as its running example.

Paper 6 describes the games of playing tic-tac-toe, flipping a coin and throwing a dice.

Paper 7 describes aspects of a gambling machine.

In addition to these examples, a larger case study has been performed. The case study describes a system for automatically matching service providers with users of those services. Based on the case study, we performed an evaluation of STAIRS. Both the case study and the following evaluation are documented in paper 8. The case study was performed by ourselves, which gave complete control over the setting of the case study. This ensured that the subsequent evaluation really evaluated STAIRS, and not some other uncontrollable factor. A natural next step in the evaluation of STAIRS, would be to perform a field study where STAIRS is used in the development of a real system.

Chapter 3

Problem Analysis

This chapter presents the result of the problem analysis. Sections 3.1 and 3.2 outline the problem to be solved in this thesis, i.e. describing the need the STAIRS method is supposed to meet. Section 3.3 refines this overall outline into a set of success criteria that the STAIRS method should fulfil.

3.1 Problem Specification

During the past decade, UML has become the de facto modelling standard used in industry. For this thesis work, we started out using the U2 Partners' submission [OMG03a] for the UML 2.0 superstructure. In 2005, a revised version of this proposal became an adopted OMG specification [OMG05], and UML is now being updated to version 2.1 [OMG06]. For our purposes, there are no significant differences between these versions, and we will refer to them collectively as "the UML 2.x standard" or simply "the standard".¹

Using the classification of Martin Fowler [Fow03], there are three primary ways of thinking about UML. In UmlAsSketch, UML diagrams are used informally for discussing selected aspects of a system to be built, or for explaining parts of an existing system. Using UmlAsBlueprint, the UML diagrams serve as a detailed specification and/or documentation of the system, containing all major design decisions. From the specification, programming the system should be pretty straightforward. Finally, UmlAsProgrammingLanguage is the idea that UML may be used as a high-level programming language, and that tools may automatically transform the UML models into executable code.

We believe that one of the main reasons for the attractiveness of UML, is its intuitive use in informal sketches. Our vision is that this intuitive feeling should be kept while at the same time improving UML for better use in blueprints and as a programming language.

All UML users, including those who use UML only for informal sketches, desire tool support for drawing diagrams. More advanced users will also need tools for analysing UML diagrams and for transforming them into executable code. Such tools, and in particular interoperability between such tools, can only be achieved if the interpretation of a given UML diagram is unam-

¹Note that there are important differences between UML 1.x and UML 2.x, in particular with respect to interactions which is the topic of this thesis. The various composition operators introduced with UML 2.x interactions have received particular attention in this thesis work.

biguous, i.e. tool vendors need UML to have a well-defined, precise, semantics. As Bran Selic points out in [Sel04], UML is not completely without a semantics, but the semantics given in the UML 2.x standard is not detailed or precise enough to answer questions about the exact meaning of more complicated diagrams. If the UML diagrams are to be used as blueprints, the lack of precise semantics is problematic not only because of little tool support, but also because it leads to situations where different users interpret the same diagram slightly differently. The result may be that the system being built is not the same system as the one intended by the person(s) making the blueprint.

Partly due to the fact that UML is a language and not a methodology, very little is found in the UML 2.x standard with respect to the semantic relationship between different UML diagrams of the same system, and between UML diagrams and computer systems. Understanding such relationships is important for ensuring that the initial requirements are contained also in later versions of the specification, for performing consistency checks across several diagrams, and for enabling reuse of analysis results obtained on diagrams created early in the development process.

3.2 Limiting the Scope of this Thesis

Currently, the UML 2.x standard describes 13 different diagram types. Clearly, it is out of scope for this thesis to cover all of UML 2.x and all possible relations between different UML models. The purpose of this section is to define the limits for this thesis work. We discuss UML diagrams in section 3.2.1, before turning to model relations and relationships between UML diagrams and computer systems in section 3.2.2.

3.2.1 UML Diagrams

In the context of UML, much work has been done on diagram types for describing system structure, but less has been done on diagram types for describing behaviour (see e.g. [Whi02], which gives an overview of formal approaches to UML). For behaviour, the UML 2.x standard describes altogether seven different diagram types, including state machine diagrams and four kinds of interaction diagrams (interaction overview diagrams, sequence diagrams, communication diagrams, and timing diagrams).

While state machines describe the complete behaviour of one part of the system (typically one or more objects), interactions are partial descriptions of communication between different parts of the system. We expect that if we first understand how to deal with the partiality of interactions, it will be relatively easy to generalize the obtained results to complete descriptions in the form of e.g. state machines. Consequently, interactions have been chosen as the focus of this thesis. Chapter 7.3 provides a more detailed discussion of to what extent the obtained results generalize to other specification techniques.

Interactions are appealing, as they are seen to be intuitive and easy to understand and thus useful for communicating both with customers and within the development team. In addition to being used for capturing and analysing requirements, interactions are very useful as system documentation as they describe how the different components or objects in the system interact to achieve the required behaviour. Also, test scenarios may be specified and documented using interactions.

As an example interaction, the sequence diagram in figure 3.1 (taken from [PP05]) describes how a workstation sends two ping packets to the server. The two responses may take different routes in the network, leading to the second response message arriving at the workstation before the first response. The workstation and the server are called lifelines, the arrows represent the messages sent between them and the arrowheads indicate the direction of each message. The open arrowheads indicate that the messages are asynchronous, meaning that the sender of a message is not required to wait for a reply before continuing with its behaviour.

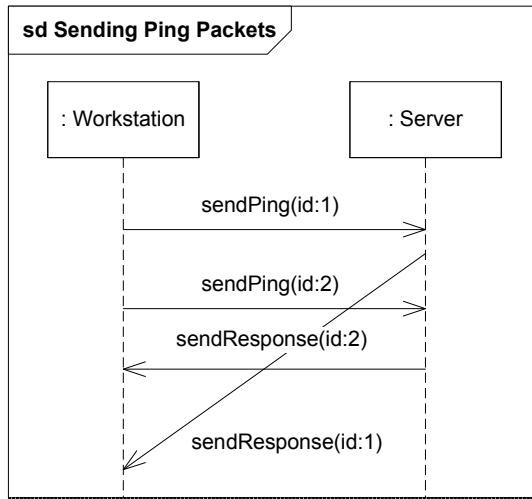


Figure 3.1: Example interaction

A message consists of two events, the send event and the receive event. The implicit composition operator in UML 2.x interactions is weak sequencing, where events on the same lifeline are ordered from the top and downwards. Events on different lifelines may happen in any order (with the obvious restriction that a message must be sent before it may be received). For the interaction in figure 3.1, this means for instance that the workstation may send both ping messages before any one of them is received by the server, or that the server may receive the first message (and possibly also send the response) before the workstation sends the second ping message.

The interaction in figure 3.1 is obviously not a complete specification of how ping packets may be sent. Is the workstation allowed to send more ping messages if it takes too much time before it receives any response? May the workstation send only one ping message? What if the server is down, and the workstation never receives any response message at all? May the server respond to some, but not all, of the ping messages? The given interaction does not provide an answer to any of these questions. This does not mean that the interaction is wrong, but rather suggests that the interaction is a partial specification as it only describes a few example scenarios.

Another example interaction is given in figure 3.2. In this sequence diagram, a time constraint is used to describe that after the server has received the request from the client, it should take between zero and five time units before the server sends the response back. Again, this is not a complete specification. The sequence diagram in figure 3.2 does not describe what should happen if, for instance, the server receives a second request or if the server fails to respond within the given time limit. Also, from the UML 2.x standard it is not clear whether the time constraint

refers to the point in time when the request arrives in the input queue of the server, or to the point in time when the sever consumes the message from the bu er and starts the handling of the request.

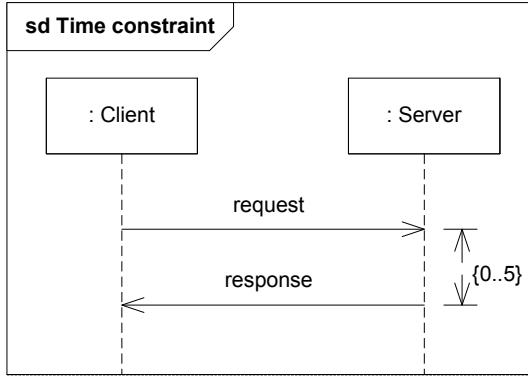


Figure 3.2: Interaction with time constraint

For UML 2.x interactions, the standard also includes a number of operators for specifying e.g. parallel execution and alternative behaviours. Figure 3.3 (taken from paper 5) is an example interaction using the alt-operator when describing how a client may interact with an appointment system in order to book an appointment. This interaction describes altogether four di erent system behaviours. What is not clear from the UML 2.x standard, is whether a system

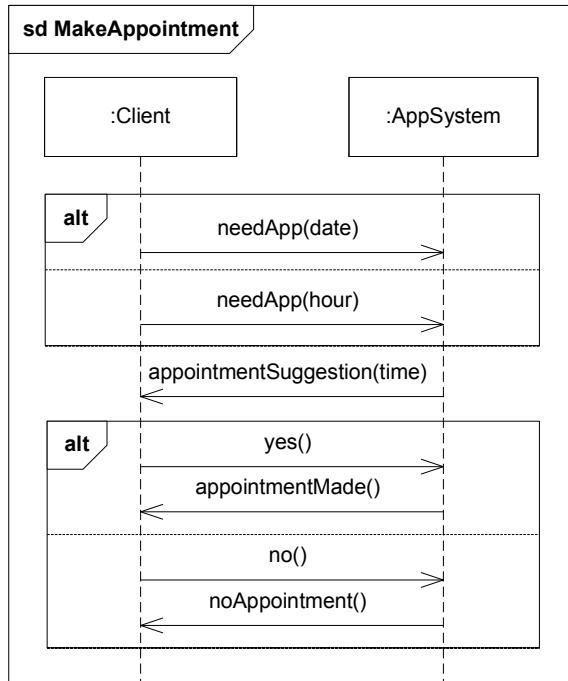


Figure 3.3: Alternative behaviours

compliant with the interaction must be able to perform all of these alternatives or if performing only one (or maybe even none) is sufficient. In other words, does the alt operator specify alternative behaviours that are mandatory (i.e. required) or potential (i.e. optional)? Currently, both interpretations are being used by UML practitioners.

UML 2.x also provides operators for specifying negative behaviour in the interactions. As an example, consider the interaction in figure 3.4 (which is a simplified version of an example given in paper 3). Here, the neg operator of UML 2.x is used to specify that if a user orders coffee from a vending machine, he should not receive tea. The exact interpretation of this interaction is not given by the informal semantics in the UML 2.x standard. For instance, is the behaviour where the user orders coffee and then nothing more happens positive? What about the behaviour where the user orders coffee and then receives cappuccino?² Does the interaction describe any positive behaviours at all? And what is the exact set of negative behaviours described by the interaction? For instance, if the user orders coffee and then receives both tea and coffee, is that behaviour also negative?

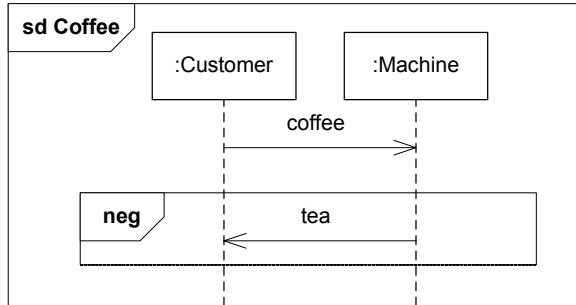


Figure 3.4: Negative behaviours

The discussion of these examples demonstrates that a precise semantics for UML 2.x interactions is needed in order to understand the exact sets of positive and negative behaviours described by an interaction. In particular, the following is a list of features of interactions that must be addressed by the work in this thesis:

- An interaction may describe both positive (i.e. valid) and negative (i.e. invalid) system behaviour.
- Interactions are partial, meaning that there are behaviours not described as either positive or negative.
- Interactions may specify both mandatory (i.e. required) and potential (i.e. optional) behaviour.
- Interactions may specify time constraints, and these may be related to both the sending, the arrival (i.e. reception) and the consumption of messages.

²Some readers may find that this is a strange question to ask for the given interaction. However, one possible interpretation of the standard is that *everything* except the behaviour inside neg should be positive, i.e. that everything is positive except from the user receiving tea.

3.2.2 Model Relations

We find it useful to distinguish between three basic ways that two interactions describing the same system may be related:

1. One of the interactions is a corrected version of the other, for instance correcting errors or taking into account changed requirements.
2. One of the interactions is a refinement, i.e. a more detailed description, of the other interaction.
3. The two interactions describe the system addressing different concerns, i.e. using different viewpoints.

Although correction is a central part of system development, relating the original and the corrected model is interesting mainly for ensuring that important refinement relations and viewpoint correspondences still hold after the correction. In general, using viewpoints are particularly useful when specifying large and complex systems, while refinement is important in any step-wise and incremental development process.

Consequently, the focus of this thesis is on refinement relations. The concept of refinement has been studied within the area of formal methods since the early 1970s (see section 4.4), and one of the challenges of this thesis is to find out how the essence of this theory may be explained and used in the practical setting of UML 2.x interactions.

From formal methods, we know that in order to support step-wise, compositional development, the semantics of UML 2.x interactions should be compositional. Following [dRdBH⁺01], this means that the semantics of an interaction should be a function of the semantics of each of its sub-interactions and the operator(s) used for composing them. No more knowledge about the operands is required. Also, the composition operators should be monotonic with respect to refinement, meaning that separate refinement of each operand results in a refinement of the complete interaction. As demonstrated in e.g. [dR85], [Col93] and [MS00], achieving compositionality is not straightforward, and great care should be taken when formalizing the semantics of UML 2.x interactions. For the refinement relations, we also know that they should be transitive, as this ensures that the result of several successive refinement steps is a refinement also of the original interaction.³

In addition to refinement, we also address the notion of compliance for relating UML 2.x interactions and computer systems. The final system should be in compliance with all of the interactions made during the step-wise development process, meaning that compliance should be a special case of refinement. As UML 2.x interactions does not prescribe any particular technology to be used for the final system, the notion of compliance should not depend on any particular kind of technology either.

3.3 Goals and Success Criteria

The overall goal of this thesis is to improve system development processes that use UML. Based on the above analysis, we conclude that there is a need for a method that may be used by both

³For simplicity, we often refer to monotonicity as a property of the refinement relations, and list monotonicity together with other refinement properties such as transitivity and reflexivity.

tool vendors and UML users. In particular, tool vendors need:

- A precise semantics for UML 2.x, making it possible to create tools that may assist when creating and analysing UML models, and possibly also perform automatic analysis of UML models.
- Precise definitions of possible model relationships, enabling tools to perform consistency checking and assist in creating and validating refinements.

In addition to improved tool support, UML users need

- Methodological guidelines for creating, developing, implementing and maintaining specifications expressed as UML models.

Looking at the combined needs of tool vendors and UML users, and narrowing it down to the scope of this thesis, there is a need for a formal foundation for UML 2.x interactions, consisting of:

- A formal definition of the semantics of UML 2.x interactions.
- An explanation of useful refinement relations for interactions, with corresponding formal definitions.
- An explanation of the relation between interactions and computer systems, i.e. an answer to the question of when a given system is in compliance with a given interaction. Again, the informal explanation should be accompanied by corresponding formal definitions.
- A methodology explaining how to use this formal foundation in practical system development with UML 2.x interactions.

In order to verify that the STAIRS method presented in this thesis provides the needed foundation, based on the above analysis we also formulate a set of success criteria for such a foundation.

- The formal semantics should
 - take into account the partiality of UML 2.x interactions.
 - handle both positive and negative behaviour.
 - handle both mandatory and potential behaviour.
 - include a notion of time.
 - be compositional, i.e.
 - * the meaning of an interaction should be completely determined by the semantics of its sub-interactions and the composition operators used.
 - * the composition operators should be monotonic with respect to refinement.
 - be in accordance with the UML 2.x standard.
- The refinement relations should

- take into account the partiality of interactions.
 - handle both positive and negative behaviour.
 - handle both mandatory and potential behaviour.
 - capture the main refinement notions known from classical formal methods.
 - be transitive, thus enabling stepwise development.
- The compliance relation between interactions and computer systems should
 - be a special case of refinement.
 - be independent of technology used for the computer system.
 - The methodology should
 - be conservative, i.e. based on existing UML methodology.
 - be useful without thorough knowledge of the formal definitions.

Chapter 4

State of the Art

This chapter presents state of the art representing background material for this thesis work. In particular, we investigate to what extent existing theories and methods fulfil the requirements from section 3.3.

In section 4.1 we consider methods for defining semantics, and in section 4.2 we investigate alternative ways of introducing time in the semantics. Section 4.3 considers different ways to specify alternative behaviours, i.e. different kinds of nondeterminism. Section 4.4 provides an overview of refinement in various formal methods, focusing on to what extent they include mechanisms for supporting the special features of interactions described in section 3.2.1. In section 4.5, we consider UML-related methodologies and other relevant work on UML. When advancing UML from version 1.5 [OMG03b] to version 2.0 [OMG04], sequence diagrams underwent a major revision, strongly influenced by Message Sequence Charts (MSC) [ITU99]. MSC is treated in section 4.6.

Our overall conclusion is that these theories and methods provide useful background material, but they are not sufficient for fulfilling the requirements formulated for STAIRS in section 3.3.

4.1 Semantics

The semantics of a language defines the meaning of statements formed using the syntactical constructs of that language. In computer science, it is common to categorize semantics definition methods as axiomatic, denotational or operational (see e.g. [Sch96]). In this section we give a brief overview of each of these methods and evaluate their suitability for defining the semantics of UML 2.x interactions.

In axiomatic semantics, the meaning of individual statements is defined indirectly by describing logical axioms and rules that apply to these statements. Axiomatic semantics are mainly used for deriving or proving desirable properties of the statements, such as in Hoare logic [Hoa69]. The axioms given are usually concise and understandable, but the descriptions tend to be very large and complex for real languages with many basic constructs [Pri00]. As a result, we find axiomatic semantics less suitable for defining the semantics of interactions.

In denotational semantics, the meaning of statements in the language is given as a mathematical function from syntactical expressions to a well-known domain. The main challenge using

denotational semantics is deciding on the target domain that should be used. On the one hand, the domain should be well-known and thus understandable, while on the other hand the semantic function should be as simple as possible for improving readability. A common approach is therefore to first define a specialized target language (using e.g. axiomatic semantics), and then basing the denotational semantics on this specialized language [Pri00]. In the UML 2.x standard, the semantics of interactions is explained using traces of event occurrences, meaning that such traces would form a natural target domain for a formal denotational semantics of interactions.

Compared to axiomatic and denotational semantics, operational semantics is closer to a real implementation. In operational semantics, the meaning of individual statements are defined in terms of how these statements may be executed using an abstract interpreter (which in turn must have a well-defined semantics). Operational semantics provides a good formalization of implementation, and is often easily understandable for tool developers. However, it may be difficult to derive formal proofs from an operational semantics [Pri00], making it less suitable for our purposes.

To conclude, a formal semantics for UML 2.x interactions should most probably use a denotational semantics based on traces of events.

4.2 Time

There exists several techniques for introducing time in the syntax and semantics of specification languages. In [Lam05], a distinction is made between explicit- and implicit-time descriptions. In explicit-time descriptions, time is introduced by a special variable representing the current time. The passing of time is modelled by an action incrementing this special variable. The advantage of using explicit-time is that the time variable is treated in the same way as other variables by both the language and its tools. Explicit-time works well for e.g. state machines, but is less suitable for trace-based formalisms and other formalisms containing no explicit notion of a global state. Instead, such formalisms usually use implicit-time descriptions, in which the language is extended with special constructs for expressing timing properties.

[AH92] contains a survey of possible formal semantics for real-time systems. A general semantics based on interval sequences is described, together with four semantical choices leading to a total of sixteen possible kinds of formal semantics. The first choice is whether to use state sequences or observation sequences in the semantics. In section 4.1, we have already concluded that traces of events, i.e. observation sequences, should be used. Secondly, we will make the assumption that events are instantaneous, meaning that using time points is sufficient and that the more general time intervals are not needed.

The third choice in [AH92] is whether to use strictly monotonic or weakly monotonic time. Weakly monotonic time means that adjacent time instants may be identical, and is necessary for modelling interleaving of simultaneous events. This is the case for interactions, where two events on different lifelines may occur at the same time. Finally, there is a choice between real-numbered and integer time. [AH92] states that integer time is sufficient for synchronous systems. However, interactions typically describe asynchronous messages, meaning that real-numbered time seems to be the best choice.

4.3 Nondeterminism

In some sense, potential and mandatory behaviours may be understood as different kinds of nondeterminism. Potential behaviour is similar to nondeterminism in the form of underspecification, where an implementer is free to choose only one of the given behaviours. As for mandatory behaviour, inherent nondeterminism requires that all of the behaviours given in the specification are also reflected in the final system.

As pointed out in e.g. [Ros95] and [Jür01], distinguishing between underspecification and inherent nondeterminism (also called unpredictability) may be essential for ensuring certain security properties of a system. For instance, when creating keys and nounces, the set of possible outcomes should be sufficiently large to make them unguessable by an adversary. If the different alternatives are specified using underspecification, this set may be significantly reduced in the final system leading to an insecure system. However, most formalisms do not distinguish between underspecification and inherent nondeterminism. In [Jür01], unpredictability is ensured using specific primitives instead of relying on nondeterministic choice being interpreted as inherent nondeterminism.

In the setting of algebraic specifications, [WM01] argues for using explicit nondeterminism in cases where underspecification might in fact lead to overspecification. However, the resulting system may still be deterministic.

In VDM-SL, a similar distinction is made between underdeterminedness (i.e. underspecification) and nondeterminism when interpreting looseness in specifications (i.e. specifications allowing alternative behaviours) [LAMB89]. Looseness in function definitions is interpreted as underdeterminedness, meaning that the exact definition is chosen at implementation time. Looseness in operations is interpreted as nondeterminism where the choice may be delayed until run-time, meaning that the final system may be either deterministic or nondeterministic.

CSP [Hoa85, Ros98] includes two different operators for nondeterminism. With internal nondeterminism, the system is free to choose whether it should offer all alternatives or only one (or some) of them. The choice may be performed at run-time, making the system nondeterministic, but the choice may also be made by the implementer, resulting in a deterministic system. For external nondeterminism (also called environmental choice), the behaviour is determined by the environment and the system must be able to perform all alternatives.

A similar distinction is that between angelic and demonic choice made in e.g. the refinement calculus [BvW98]. With angelic choice, the choice between the alternatives is made by the system with the goal of establishing a given postcondition. This means that if the behaviours are similar up to some point, the choice between them may be deferred as long as possible in order to increase the chances of obtaining the desired end result. Demonic choice, on the other hand, is assumed to be resolved by an environment with another goal. Hence, the system may only guarantee the given postcondition if that condition may be established for *all* of the demonic alternatives.

[SBDB97] extends the process algebraic language LOTOS [ISO89] with a disjunction operator for specifying implementation freedom (i.e. underspecification), leaving the LOTOS choice operator to be used for inherent nondeterminism. With this new disjunction operator, exactly one of the alternatives may be implemented, in contrast to the usual interpretation of underspecification which also allows implementations with several of the alternative behaviours.

To conclude, the described formalisms include a variety of operators for specifying non-

determinism and choice, but the distinction between potential and mandatory behaviour is not fully covered by any of these approaches.

4.4 Refinement

[Dij68], [Wir71] and [DDH72] introduced the notion of stepwise program construction/refinement in the setting of sequential programs. In each refinement step, one or more high-level instructions may be decomposed into more detailed instructions, refining also the data structure whenever necessary. The notion of stepwise refinement was then formalized in e.g. [Hoa72] and [Jon72].

Later, methods such as CSP [Hoa85, Ros98], TLA [AL91, Lam02] and F [BS01] have investigated refinement in the setting of concurrent and reactive systems. Common for all of these methods is that in each refinement step, properties are added to the specification in order to make it more precise or deterministic. This means that the properties of the refinement should imply the properties of the original specification. For trace-based formalisms this corresponds to trace inclusion, i.e. all traces of the refinement must also be traces of the original specification.

In addition to reducing the set of possible behaviours, a refinement step may also make changes to the representation of data. This is captured by the notion of interface refinement in e.g. [Lam02] and [BS01], where the correspondence between the two representations is given as part of the refinement step. A unifying treatment of data refinement may be found in [dRE98]. [BS01] also defines a more general notion called conditional refinement, in which additional assumptions may be made about the environment.

Traditional refinement methods assume that the specifications are complete, meaning that if a behaviour does not have the properties required by the specification, that behaviour is negative and should not be exhibited by the specified system. As we have seen in section 3.2.1, this “closed world assumption” does not hold for interactions in general. As a consequence, the methods described so far cannot be directly applied in the setting of interactions.

In traditional pre/post-specifications [Hoa69], arbitrary behaviour is allowed if the pre-condition is false. Still, all behaviours are categorized as either positive or negative. Positive behaviours are those where the pre-condition is false, together with the behaviours satisfying both the pre- and the post-condition. Behaviours that satisfy the pre-condition but not the post-condition are negative. Refinement means to reduce the set of positive behaviours, either by strengthening the post-condition or weakening the pre-condition.

In [MBD00], pre/post-specifications in Z are given a three-valued interpretation, the third truth value being “don’t care” or undefinedness. In this interpretation, behaviours with a false pre-condition are undefined and may later be refined as either positive or negative. To restrict the undefinedness, more general guards are introduced in addition to the pre-conditions. If a guard is false, the behaviour is negative, while if the guard is true and the pre-condition is false, the behaviour is undefined. Refinement now means removing undefinedness by either weakening the pre-condition or strengthening the guard, or removing underspecification by strengthening the post-condition.

[MBD00] also presents an alternative three-valued interpretation in order to capture required nondeterminism, i.e. mandatory behaviour. In this interpretation, a behaviour is mandatory if the pre-condition is true, while behaviours where the pre-condition is false and the guard is true

are potential behaviours. A refinement step may no longer strengthen the post-condition, but the gap between the guard and the pre-condition may be made smaller as long as it is not eliminated entirely.

These results concerning pre/post-specifications cannot be directly applied to specifications in the form of interactions. Typically, pre-conditions apply to the input values of a (sub-)system, whereas post-conditions constrain the set of valid output values. In other words, if the pre-condition holds, the post-condition categorizes all possible output behaviours as either positive or negative. For an interaction, it is the complete sequence of events that are either positive, negative, or not described by the interaction. As a result, an interaction may be incomplete also with respect to the allowed output behaviours for some valid input.

[dJvdPH00] presents a formal method for refining incomplete requirements specifications, incomplete meaning “yet unfinished”. However, the incompleteness addressed is a controlled form of incompleteness, limited to additional nondeterminism (intended or unintended) and parameterized specifications where the parameters refer to parts that are not specified yet. General incompleteness, such as may be the case with interactions, is not addressed.

4.5 UML-Related Methodologies

There exists a number of UML-related methodologies, including the Unified Process [JBR99], the Rational Unified Process (RUP) [Kru04], Catalysis [DW99], KobrA [ABB⁺02] and Agile Modeling [Amb02]. As representatives, we have here chosen to focus on RUP and Catalysis. RUP is a specialization of the Unified Process, and these two methodologies are the ones most closely related to UML. Catalysis is chosen as it includes the most thorough treatment of refinement.

RUP [Kru04] is a generic software engineering process developed as a complement to UML. While UML is only a language that may be used for expressing models, RUP describes which models should be created at each stage in the development process. RUP divides the development process into four main phases consisting of a sequence of iterations, and nine disciplines that cut across these iterations. Although prescribing the creation of a number of UML models (and other documents), RUP is vague on the relationship between these. [Kru04] states a few places that two or more models should be consistent, or that one model should refine another, but this is not formalized and the concept of refinement is not explored in any depth.

Catalysis [DW99] is a method for object and component-based development using UML 1.x. For the development process, Catalysis gives a number of process patterns that may be adapted by a concrete system development project. Catalysis include many of the same ideas as RUP, such as phases, iterations, and the various activities performed within each of the iterations. For our purposes, the most interesting part of Catalysis is its treatment of refinement. Four basic kinds of refinement are defined:

- Operation refinement: Refining the behaviour of a type, i.e. its operation specifications (usually in the form of pre- and post-conditions).
- Model refinement: Refining the attributes of a type, i.e. its static model, allowing the implementation attributes to be different from those of the model.

- Action refinement: Refines a single action into a set of actions or a complex protocol of interactions between objects.
- Object refinement: Refines a single object into a set of objects.

Most refinements can be understood as combinations of these four, either in sequence (“big refinements”) or performed together (for instance combining action and object refinement). The same refinement relations are used both for relating two specifications at different level of abstraction, and for relating specifications and computer systems. Several examples of Java implementations are given.

Catalysis does not precisely define the various kinds of refinement as the aim is a “more practical solution”. Instead, [DW99] gives an overview of questions that must be answered in order to justify that a basic refinement step is correct. For action refinement, which is the most relevant for interactions, the question to be answered is: “What sequences of detailed actions will realize the effect of the abstract action?”. In addition to writing down a justification for the refinement, one should also write down the reasons from choosing this realization instead of one of the alternatives.

Neither of the methodologies mentioned above provides any attempt to do anything about the lack of a precise semantics for UML 2.x. However, there have been many other attempts to formalize UML (see e.g. [Whi02]). Much of the work on UML have focused on the static diagram types (such as class diagrams), and are not relevant for our work on interactions.

[Öve99] gives meaning to UML 1.x collaborations (which are described using interaction diagrams) by defining what it means for a set of objects to conform to a collaboration. Being based on message sequences, and not sequences of events, there is no way to express weak sequencing. The partiality of interactions are taken care of, in that objects may participate in other collaborations as well. As UML 1.x does not include any operators for negation, the concept of negative behaviours is not treated. For relating two collaborations, [Öve99] defines the notion of specialization. The specialization must include all sequences of the original collaboration, but may add both new sequences and new messages interleaved in the original sequences.

A more formal treatment of UML 1.x collaborations is given in [Kna99], using temporal logic and also incorporating semantics of actions in the form of transition systems. [Kna99] also provides a semantics based on pomsets, which are seen as closer to the informal semantics given in the UML 1.x standard. Again, the concept of negative behaviours is not treated. Neither is there any notion of refinement.

In [GHK99], the behavioural diagram types of UML 1.x (including state diagrams and collaboration diagrams) are given a common semantics in the form of constraint processes in cTLA (compositional TLA). Partial specifications are handled, in that each process constrains only those actions that are relevant with respect to the given diagram. However, negative behaviours are not treated, and neither are refinement.

[Jür02] (later published in a revised form as [Jür05]) provides a formal semantics for a restricted and simplified part of several kinds of UML 1.x diagrams, including sequence diagrams. The semantics is based on Abstract State Machines, and does not include explicit negative behaviours. Corresponding to the similar notions in [BS01], [Jür02] defines behavioural and interface refinement for UML specifications. In addition, delayed refinement is defined for being able to introduce time delays. However, the partiality of interactions is not taken into account. Neither is there any distinction between mandatory and potential behaviours.

4.6 Message Sequence Charts

UML 2.x interactions are strongly influenced by Message Sequence Charts (MSC) [ITU99]. Although there are some differences in terminology and the graphical syntax used, the two sequence diagram dialects are very similar [Hau04]. Simple MSCs correspond to the sequence diagrams of UML 2.x, and includes operators for specifying alternatives, parallel composition, iteration, exceptions and optional behaviour. All of these operators are included in UML 2.x interactions, where the operator break corresponds to MSC exceptions [Hau04]. UML 2.x interaction overview diagrams are a slightly generalized version of high-level MSCs, which provides an overview of how the basic MSCs are composed.

MSC does not provide any constructs for defining negative behaviour. Instead, [Hau97] proposes a methodology where different MSC documents (a number of MSCs) may be categorized as describing possible, not possible or all possible sequences of the system.

The MSC specification [ITU99] includes informal descriptions of the intended semantics. This has not been formalized, but [ITU98] contains an official operational semantics for [ITU96], a previous version of MSC. The semantics is defined compositionally, and includes definitions for the time concepts given in [ITU96].

The only notion of refinement mentioned in [ITU99] is that of instance decomposition. This refinement is fairly weak, as it does not require behavioural refinement but only that there is some structural similarity between the decomposed instance and its decomposition. The formalization in [ITU98] does not treat refinement at all.

The work in [Krü00] is very relevant in our setting as it contains both a formal semantics, refinement notions and a methodology for MSC. While the operational semantics in [ITU98] is based on process algebra, [Krü00] defines a denotational semantics based on streams where the semantics of a given MSC specification is a set of channel and state valuations. The main difference between the two is that [Krü00] uses strict and not weak sequencing. As noted in section 3.2.1, weak sequencing is important for UML 2.x interactions. To achieve weak sequencing, [Krü00] requires that this should be explicitly stated in the MSC. The semantics in [Krü00] is defined compositionally, and important properties such as associativity and commutativity of the operators are established. The inclusion of time is discussed informally, but is not part of the formal semantics.

[Krü00] defines four different refinements notions for MSCs. [Krü00] allows references to non-existing MSCs, which are interpreted as arbitrary behaviour. By reference binding, this arbitrary behaviour may then be refined by defining the missing MSCs. Property refinement means removing underspecification by reducing the possible behaviours of the overall system, while message refinement means substituting an interaction sequence or protocol for a single message. Finally, structural refinement means replacing a simple component with a set of other components, similar to instance decomposition in MSC (and UML 2.x). Both message refinement and structural refinement are global substitutions, i.e. all occurrences of the message or component must be replaced by the same sequence or decomposition. Property refinement is reflexive, transitive and monotonic with respect to all MSC operators.

Related to implementations, [Krü00] defines four possible interpretations of a single MSC, such that within one specification the different MSCs may have different interpretations. While an existential MSC describes partial system behaviour that *may* occur during execution of the system, a universal MSC describes behaviour that *must* occur at some point in time in all executions

of the system. An exact MSC prohibits all other behaviours than the ones specified by the MSC, while a negated MSC describes negative behaviours. Except from existential interpretation, the notions are monotonic with respect to property refinement.

Live Sequence Charts (LSC, [DH01, HM03]) is an extension of MSC focusing on the ability to specify liveness, i.e. things that must occur. LSC has influenced the work in [Kri00], and to some extent also UML 2.x interactions.

LSC distinguishes between existential and universal diagrams. An existential diagram specifies possible behaviour, i.e. example scenarios that must be satisfied by at least one execution of the system. A universal diagram specifies necessary behaviour, i.e. behaviour that must be fulfilled by all executions. A universal diagram consists of two parts, a prechart and the main chart. If the system at some point in time fulfils the prechart, then the main chart must also be fulfilled. However, if the prechart is never satisfied, the diagram imposes no restriction on the system behaviour. [DH01] also discusses the closing of a specification with respect to a system, which means that for all possible executions of the system at least one LSC is satisfied, i.e. the system cannot exhibit behaviour not described by any of the diagrams.

Within a single diagram, LSC elements may be specified as cold (meaning that they *may* occur) or hot (meaning that they *must* occur). LSC does not include the standard MSC operators for control structures like alternatives and iteration. Instead, cold conditions may be used for specifying this. Hot conditions may be used to specify anti-scenarios (i.e. negative behaviour) by including the unwanted behaviour in the prechart of a universal diagram where the main chart contains a single false hot condition.

[HM03] provides an operational semantics for LSC, tailored towards their Play-Engine tool which provides means for capturing requirements by using a graphical user interface and afterwards executing the resulting LSCs. For simplifying the tool, the LSC semantics differ from MSC on important aspects. For instance, synchronization between all lifelines is performed at the beginning of each sub-diagram. As a result, the semantics of a loop is not the same as the semantics of writing its content in full. Also, the temperature (hot/cold) of messages is only syntactic sugar without any semantic implication, as the temperature of the corresponding locations (for the send and the receive event) is given higher priority.

[HM03] also includes more advanced constructs such as variables and time. Even though there is a notion of sub-diagrams, there are no construct for referencing one LSC within another or for composing LSCs. LSC contains no notion of refinement, but [HM03] mentions object refinement (i.e. decomposition) as an important area for further research.

Chapter 5

STAIRS

In this chapter we give an introduction to the STAIRS method. First, in section 5.1 we describe the initial work on STAIRS by Øystein Haugen and Ketil Stølen. In section 5.2, we give a brief summary of STAIRS as it appears today.

5.1 The First STAIRS

At the UML 2003 conference, Øystein Haugen and Ketil Stølen presented the first paper on STAIRS, called “STAIRS — Steps to Analyze Interactions with Refinement Semantics” [HS03]. This paper contained the basic ideas of STAIRS, which has been further developed by the work presented in this thesis.

The main ideas in [HS03] can be summarized as follows:

Mandatory vs potential behaviour. A specification may employ nondeterminism in order to capture two very different kinds of requirements. First, nondeterminism in the form of under-specification is used where there are alternative behaviours serving the same purpose and a correct implementation is only required to fulfil one of these. The alternative behaviours then represent what is referred to as potential behaviour.

On the other hand, explicit nondeterminism is used where the nondeterminism is required also by a correct implementation. As an example, every lottery ticket in a lottery should have the possibility to win the prizes, even though only one ticket is drawn for each prize. Explicit nondeterminism is referred to as mandatory behaviour.

For describing potential behaviour, the common UML alt operator is used, while a new operator called xalt is introduced in order to capture mandatory behaviour and distinguish this from potential behaviour.

Negative, positive and inconclusive behaviour. An interaction is understood as describing a set of positive (i.e. valid, legal, or desirable) traces, and a set of negative (i.e. invalid, illegal, or undesirable) traces. Traces not considered by the interaction are referred to as inconclusive.

Supplementing, narrowing and detailing. Three basic refinement relations are described, corresponding to basic system development steps:

Supplementing categorizes earlier inconclusive traces as either positive or negative, recognizing that early specifications in the form of interactions are usually incomplete. Positive traces remain positive, and negative traces remain negative.

Narrowing reduces the set of positive traces by redefining some of them as negative, capturing new design decisions or matching the problem better. Inconclusive traces remain inconclusive and negative traces remain negative.

Detailing introduces a more detailed description without significantly altering the externally observable behaviour.

Semantics. Trace semantics for simple interactions using the operators seq (weak sequencing), ref (interaction occurrence), par (parallel combined fragment), neg (negation), alt (potential behaviour) and xalt (mandatory behaviour) are explained informally using small example interactions.

A semantic model for interactions is proposed, capturing all of the ideas described above and in particular the distinction between mandatory and potential behaviour. In our later work on formalizing STAIRS, we chose a slightly different semantic model. The reasons for this, and a comparison between the different models, may be found in section 7.1. We also chose another interpretation of the neg operator, as discussed in paper 3.

5.2 STAIRS Today

As explained in section 2, the STAIRS method has been developed by iteratively adding more and more features (e.g. new operators, refinement relations or methodological advice) to it. As our understanding of system development using UML 2.x interactions has improved, some minor adjustments have been made to the formal definitions in order to reflect this. In this section, we give a brief overview of the main syntax, semantic model and refinement relations as they appear in STAIRS today. For motivation, examples, and more details we refer to the attached papers. Also, the pragmatical guidelines from paper 5 are not repeated, neither are the compliance relations from paper 7.

5.2.1 Syntax

The syntax of basic STAIRS interactions is defined by the BNF-grammar in figure 5.1. Signal represents the actual content of a message, Lifeline is the name of a lifeline (representing a component) in the diagram and Set should be an expression that evaluates to a subset of $\mathbb{N}_0 \cup \{\infty\}$ (the natural numbers including 0, and ∞).

In addition to the operators in figure 5.1, veto and opt are defined as high-level operators by:

$$\text{veto } d \stackrel{\text{def}}{=} \text{alt} [\text{refuse } [d], \text{skip}] \quad (5.1)$$

$$\text{opt } d \stackrel{\text{def}}{=} \text{alt} [d, \text{skip}] \quad (5.2)$$

The ref construct is seen as a syntactical short-hand for the contents of the referenced interaction. Gates are treated in appendix A of paper 2.

$\langle \text{Interaction} \rangle$	$\rightarrow \langle \text{Empty} \rangle \mid \langle \text{Event} \rangle \mid \langle \text{Refuse} \rangle \mid \langle \text{Assert} \rangle \mid \langle \text{Weak sequencing} \rangle \mid \langle \text{Parallel execution} \rangle \mid \langle \text{Loop} \rangle \mid \langle \text{Potential alternatives} \rangle \mid \langle \text{Mandatory alternatives} \rangle$
$\langle \text{Empty} \rangle$	$\rightarrow \text{skip}$
$\langle \text{Event} \rangle$	$\rightarrow (\langle \text{Kind} \rangle, \langle \text{Message} \rangle)$
$\langle \text{Kind} \rangle$	$\rightarrow \langle \text{Transmission} \rangle \mid \langle \text{Reception} \rangle$
$\langle \text{Transmission} \rangle$	$\rightarrow !$
$\langle \text{Reception} \rangle$	$\rightarrow ?$
$\langle \text{Message} \rangle$	$\rightarrow (\text{Signal}, \langle \text{Transmitter} \rangle, \langle \text{Receiver} \rangle)$
$\langle \text{Transmitter} \rangle$	$\rightarrow \text{Lifeline}$
$\langle \text{Receiver} \rangle$	$\rightarrow \text{Lifeline}$
$\langle \text{Refuse} \rangle$	$\rightarrow \text{refuse} [\langle \text{Interaction} \rangle]$
$\langle \text{Assert} \rangle$	$\rightarrow \text{assert} [\langle \text{Interaction} \rangle]$
$\langle \text{Weak sequencing} \rangle$	$\rightarrow \text{seq} [\langle \text{Interaction list} \rangle]$
$\langle \text{Parallel execution} \rangle$	$\rightarrow \text{par} [\langle \text{Interaction list} \rangle]$
$\langle \text{Loop} \rangle$	$\rightarrow \text{loop Set} [\langle \text{Interaction} \rangle]$
$\langle \text{Potential alternatives} \rangle$	$\rightarrow \text{alt} [\langle \text{Interaction list} \rangle]$
$\langle \text{Mandatory alternatives} \rangle$	$\rightarrow \text{xalt} [\langle \text{Interaction list} \rangle]$
$\langle \text{Interaction list} \rangle$	$\rightarrow \langle \text{Interaction} \rangle \mid \langle \text{Interaction list} \rangle, \langle \text{Interaction} \rangle$

Figure 5.1: Syntax of basic STAIRS interactions

In our work, we have defined two orthogonal extensions of STAIRS, TimedSTAIRS (paper 2) and guarded STAIRS (paper 4). For TimedSTAIRS, the syntax is extended as defined by the BNF-grammar in figure 5.2. Nonterminals that are unchanged from the syntax of basic STAIRS in figure 5.1 are not repeated. In TimedSTAIRS, every event is decorated with a unique timestamp tag (`TimestampTag`) as a placeholder for real timestamp values. `TimeConstraint` is a boolean expression on such timestamp tags. In TimedSTAIRS, we also distinguish between the reception of a message (the arrival of the message in the input buffer of the lifeline) and the consumption of the message (when it is taken from the input buffer and processed by the lifeline).

For interactions with data and guards, the syntax is extended as defined by the BNF-grammar in figure 5.3. `Variable` should be either a global variable or a variable local to the lifeline on which the assignment is placed (not shown in our textual syntax), while `Expression` is a mathematical expression and `Constraint` is a boolean expression on variables. In guarded STAIRS, `alt/xalt`-operands without an explicit guard, are interpreted as having the boolean constant `true` as guard.

Except from having a few additional operators, we only address sequence diagrams that are considered syntactically correct in UML 2.x. Also, we do not handle extra global combined fragments, and for all operators except from `seq` and `par` we assume that each operand consists of complete messages only. We also require that for diagrams consisting of more than one event, the message should be complete if both the transmitter and the receiver lifelines are present in the diagram. These requirements are written down formally in paper 2.

$\langle \text{Interaction} \rangle$	$\rightarrow \langle \text{Timed interaction} \rangle$
$\langle \text{Timed interaction} \rangle$	$\rightarrow \langle \text{Empty} \rangle \mid \langle \text{Event} \rangle \mid \langle \text{Refuse} \rangle \mid \langle \text{Assert} \rangle \mid \langle \text{Weak sequencing} \rangle \mid \langle \text{Parallel execution} \rangle \mid \langle \text{Loop} \rangle \mid \langle \text{Potential alternatives} \rangle \mid \langle \text{Mandatory alternatives} \rangle \mid \langle \text{Time-constrained interaction} \rangle$
$\langle \text{Time-constrained interaction} \rangle$	$\rightarrow \langle \text{Timed Interaction} \rangle \text{ tc TimeConstraint}$
$\langle \text{Event} \rangle$	$\rightarrow (\langle \text{Kind} \rangle, \langle \text{Message} \rangle, \text{TimestampTag})$
$\langle \text{Kind} \rangle$	$\rightarrow \langle \text{Transmission} \rangle \mid \langle \text{Reception} \rangle \mid \langle \text{Consumption} \rangle$
$\langle \text{Transmission} \rangle$	$\rightarrow !$
$\langle \text{Reception} \rangle$	$\rightarrow \sim$
$\langle \text{Consumption} \rangle$	$\rightarrow ?$

Figure 5.2: Syntax of TimedSTAIRS interactions

$\langle \text{Interaction} \rangle$	$\rightarrow \langle \text{Empty} \rangle \mid \langle \text{Assignment} \rangle \mid \langle \text{Constraint} \rangle \mid \langle \text{Event} \rangle \mid \langle \text{Refuse} \rangle \mid \langle \text{Assert} \rangle \mid \langle \text{Weak sequencing} \rangle \mid \langle \text{Parallel execution} \rangle \mid \langle \text{Loop} \rangle \mid \langle \text{Guarded alt} \rangle \mid \langle \text{Guarded xalt} \rangle$
$\langle \text{Assignment} \rangle$	$\rightarrow \text{assign} (\text{Variable}, \text{Expression})$
$\langle \text{Constraint} \rangle$	$\rightarrow \text{constr} (\text{Constraint})$
$\langle \text{Guarded alt} \rangle$	$\rightarrow \text{alt} [\langle \text{Guarded list} \rangle]$
$\langle \text{Guarded xalt} \rangle$	$\rightarrow \text{xalt} [\langle \text{Guarded list} \rangle]$
$\langle \text{Guarded list} \rangle$	$\rightarrow \langle \text{Guarded interaction} \rangle \mid \langle \text{Guarded list} \rangle, \langle \text{Guarded interaction} \rangle$
$\langle \text{Guarded interaction} \rangle$	$\rightarrow \langle \text{Guard} \rangle \rightarrow \langle \text{Interaction} \rangle$
$\langle \text{Guard} \rangle$	$\rightarrow \text{Constraint}$

Figure 5.3: Syntax of guarded STAIRS interactions

5.2.2 Semantics

STAIRS defines denotational trace semantics for interactions that are using the syntax of section 5.2.1. Our semantic domain is the set of all well-formed traces, denoted \mathcal{H} . In basic STAIRS, a trace is well-formed if, for each message, the send event is ordered before the corresponding receive event. For TimedSTAIRS, we also have additional requirements ensuring e.g. that all events in a trace are ordered by time. These requirements may be found in paper 2.

The semantics of an interaction is given as a set of m interaction obligations, where an interaction obligation is a pair of trace-sets (p, n) which gives a classification of all traces in \mathcal{H} into three categories: the positive traces p , the negative traces n , and the inconclusive traces $\mathcal{H} \setminus (p \cup n)$. Visually, we illustrate an interaction obligation as an oval divided into three regions as shown in figure 5.4.

Intuitively, each interaction obligation represents a mandatory alternative that must be present in any computer system compliant with the interaction. In earlier work on STAIRS, we classified

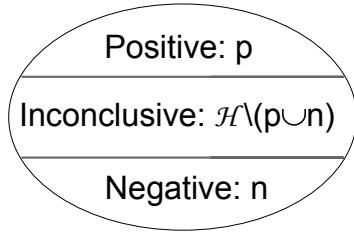


Figure 5.4: Illustrating an interaction obligation

an interaction obligation with $p \cap n \neq \emptyset$ as contradictory. We no longer find this term useful, as the same trace may very well be both positive and negative in the same interaction obligation when e.g. abstracting away guards or other details.

When defining the semantics of the individual operators, we have used the following main principles:

- The definitions should be in accordance with the informal semantics and explanations given in the UML 2.x standard.
- The definitions should be context-free, i.e. the meaning of an operator should not depend on the context in which it is used.
- All traces that are not explicitly described in a (sub-)interaction should be inconclusive for that (sub-)interaction.
- In order to support compositional development of interactions, the composition operators should be monotonic with respect to refinement.

A few times, these principles have been in conflict, and we have had to make minor compromises. In our work, monotonicity has been an important principle, as we find it essential that different parts of a specification may be developed separately. As proved in the attached papers, all operators except assert are monotonic with respect to refinement.¹ For assert, we have monotonicity in the special case of narrowing.

For the formal definitions of each operator, we refer to the attached papers, and in particular papers 1, 2 and 4. Some definitions are slightly different in the various papers. This is mainly structural differences, such as defining operators also on the level of sets of interaction obligations or changing the number of operands (which makes no difference due to the associativity results in paper 2). In the following, we comment on the more significant changes that have been made during the development of STAIRS.

First of all, papers 1 and 2 used the UML operator neg for specifying negative behaviour. In paper 3, we investigated alternative formal definitions for neg, and argued for replacing neg

¹Actually, each of the attached papers includes only a subset of the total set of operators covered by STAIRS, depending on the focus of that particular paper. All operators (except assert) are proved to be monotonic with respect to the most general refinement relation (section 5.2.3, definition (5.7) with definition (5.5)). However, in the papers introducing other refinement relations, monotonicity is only proved for the operators included in each of these papers.

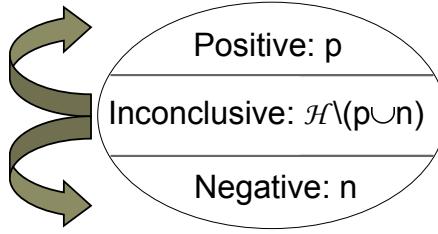


Figure 5.5: Supplementing of interaction obligations

with two new negation operators called *refuse* and *veto*. The operator *refuse* is the most basic operator, used in the definition of *veto* (which corresponds to our initial definition of *neg*). All later papers use *refuse* and *veto* instead of *neg*.

In the definition of guarded alt in paper 4, we followed the UML 2.x standard in that if all guards in an alt-construct is false, the empty trace (i.e. doing nothing) should be positive. However, the resulting definition is not associative, difficult to read and understand, and also contradicts our principle that traces not explicitly described should be inconclusive. As a result, in paper 5 we changed the definition so that the implicit empty trace was no longer included. The resulting definition is more readable, and also re-establishes associativity of alt. If the empty trace should be positive in case all other guards are false, this is easily specified by including *skip* with the guard *else* as the last alt-operand. For alt-constructs where the guards are exhaustive, the definitions in papers 4 and 5 are equivalent.

5.2.3 Refinement

In STAIRS, we have defined several refinement relations for relating UML 2.x interactions made at different stages of the development process. The refinement relations are found at two different levels: refinement of interactions and refinement of individual interaction obligations.

For individual interaction obligations, we have formalized the notions of supplementing, narrowing, and detailing described in [HS03]. In addition, in paper 7 we introduced the notion of restricted refinement.

Supplementing means adding more behaviours to the specification by reducing the set of inconclusive behaviours. This is illustrated in figure 5.5 and formally defined by:

$$(p, n) \rightsquigarrow_s (p', n') \stackrel{\text{def}}{=} p \subseteq p' \wedge n \subseteq n' \quad (5.3)$$

Narrowing means reducing underspecification by redefining positive traces as negative. This is illustrated in figure 5.6 and formally defined by:

$$(p, n) \rightsquigarrow_n (p', n') \stackrel{\text{def}}{=} p' \subseteq p \wedge n' = n \cup p \setminus p' \quad (5.4)$$

Combining supplementing and narrowing results in our most common refinement relation, formally defined by:

$$(p, n) \rightsquigarrow_r (p', n') \stackrel{\text{def}}{=} n \subseteq n' \wedge p \subseteq p' \cup n' \quad (5.5)$$

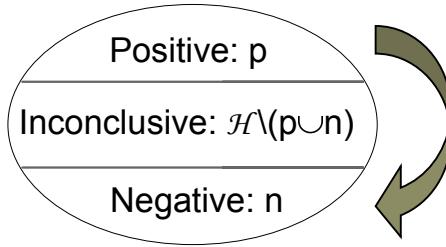


Figure 5.6: Narrowing of interaction obligations

In restricted refinement, supplementing traces as positive is not allowed:

$$(p, n) \rightsquigarrow_{rr} (p', n') \stackrel{\text{def}}{=} (p, n) \rightsquigarrow_r (p', n') \wedge p' \subseteq p \quad (5.6)$$

Detailing means reducing the level of abstraction by decomposing one or more lifelines, corresponding to structural decomposition in UML 2.x. The decomposition may result in a change in the sender/receiver of the messages, and also in internal messages, guards, etc being revealed. For a formal definition of detailing, we refer to paper 5.

For interactions which may have several interaction obligations as their semantics, we have defined two refinement notions: general and limited refinement. In general refinement, each interaction obligation for the original interaction must be refined by an interaction obligation for the refining interaction, but new interaction obligations may be added freely:

$$d \rightsquigarrow_g d' \stackrel{\text{def}}{=} \forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow o' \quad (5.7)$$

where the refinement relation \rightsquigarrow is one of the refinement relations given for interaction obligations above.

Limited refinement restricts the possibility of adding new interaction obligations by also requiring that each interaction obligation for the refining interaction must be a refinement of an interaction obligation for the original interaction:

$$d \rightsquigarrow_l d' \stackrel{\text{def}}{=} d \rightsquigarrow_g d' \wedge \forall o' \in \llbracket d' \rrbracket : \exists o \in \llbracket o \rrbracket : o \rightsquigarrow o' \quad (5.8)$$

Chapter 6

Overview of the Papers

The main results of this thesis work are documented in the eight papers found in part II. In the following, we list the publication details of each paper, together with a short description of its main research contributions. For each paper, my contribution is specified.

Paper 1: Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. STAIRS towards formal design with sequence diagrams. *Journal of Software and Systems Modeling*, 22(4):349–458, 2005.

This paper formalizes the main ideas of STAIRS, based on the presentation in [HS03]. This includes formal definitions of the main UML 2.x operators, the distinction between mandatory and potential alternatives, and the refinement relations supplementing, narrowing and detailing. In particular, all of the definitions take into account the partial nature of interactions. The semantic model used in this paper and all our subsequent work on STAIRS is slightly different from the one proposed in [HS03]. This difference is discussed in section 7.1.

My contribution: One of four main authors, responsible for approximately 25% of the work (with an emphasis on the more formal aspects).

Paper 2: Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309. Extended and revised version of: Why timed sequence diagrams require three-event semantics. In *Scenarios: Models, Transformations and Tools*, volume 3466 of *LNCS*, pages 1–25. Springer, 2005.

This paper extends STAIRS with time and three-event semantics, arguing that distinguishing between the reception and the consumption of an event is essential for evaluating whether a system fulfils the specified time constraints or not. This paper also provides formal definitions for the treatment of gates, and for additional operators such as assert and loop. The other main contribution of this paper is the appendices (not included in the LNCS-publication) containing several proofs for the soundness of the given formalization. We have associativity, commutativity and distributivity where expected, and the refinement relations are reflexive, transitive and monotonic with respect to the defined operators, enabling step-wise and compositional development of interactions.

My contribution: One of four main authors of the original paper, and the main author behind the revised report. My main responsibility was the writing up of the detailed proofs. Altogether, I was responsible for approximately 35% of the work.

Paper 3: Ragnhild Kobra Runde, Øystein Haugen, and Ketil Stølen. How to transform UML neg into a useful construct. In *Norsk Informatikkonferanse NIK'2005*, pages 55–66. Tapir, 2005.

This paper contains a discussion of alternative definitions of the neg operator used for negation in UML 2.x. The main conclusion is that having only one operator for negation is not sufficient to capture the different uses, and we propose replacing neg with the two operators refuse and veto (where veto is defined in terms of refuse).

My contribution: I was the main author, responsible for approximately 90% of the work.

Paper 4: Ragnhild Kobra Runde, Øystein Haugen, and Ketil Stølen. Refining UML interactions with underspecification and nondeterminism. Technical Report 325. Extended and revised version of: Refining UML interactions with underspecification and nondeterminism. *Nordic Journal of Computing*, 12(2):157–188, 2005.

First of all, this paper contains a thorough discussion of the difference between potential alternatives in the form of underspecification, and mandatory alternatives in the form of inherent nondeterminism. Secondly, STAIRS is extended with data and guards. Also, the notion of limited refinement is introduced, restricting the possibilities for increasing the inherent nondeterminism required of the specified system. Again, the appendices (not included in the journal-publication) contain the necessary proofs that transitivity and monotonicity holds also for this extended version of STAIRS, and that the definitions of guarded alternatives are consistent with the original definitions given for unguarded alternatives.

My contribution: I was the main author, responsible for approximately 80% of the work.

Paper 5: Ragnhild Kobra Runde, Øystein Haugen, and Ketil Stølen. The pragmatics of STAIRS. In *Proc. Formal Methods for Components and Objects (FMCO 2005)*, volume 4111 of *LNCS*, pages 88–114. Springer, 2006.

This is a tutorial paper giving pragmatical guidelines for creating and refining interactions. Also, the refinement relations of STAIRS are described in more detail.

My contribution: I was the main author, responsible for approximately 80% of the work.

Paper 6: Atle Refsdal, Ragnhild Kobra Runde, and Ketil Stølen. Underspecification, inherent nondeterminism and probability in sequence diagrams. Technical Report 335. Extended version of: Underspecification, inherent nondeterminism and probability in sequence diagrams. In *Proc. Formal Methods for Open Object-Based Distributed Systems (FMOODS 2006)*, volume 4037 of *LNCS*, pages 138–155. Springer, 2006.

This paper provides a further discussion of the different kinds of nondeterminism, and in particular how to understand interactions where these are combined in various ways. In addition to underspecification and inherent nondeterminism, we also discuss probabilistic choice, which may be understood as a generalization of inherent nondeterminism. The

report version includes an appendix containing proofs of claims made in the main body of the paper. Probabilistic STAIRS first appeared in [RHS05], and is slightly revised in this paper. In the setting of this thesis, it is interesting how probabilistic choice relates to underspecification and inherent nondeterminism, but we are not interested in probabilities as such. Hence, probabilities has not been a theme in the other parts of this introductory part.

My contribution: One of two main authors. The paper was written in close collaboration, and I was responsible for approximately 45% of the work.

Paper 7: Ragnhild Kobro Runde, Atle Refsdal and Ketil Stølen. Relating computer systems to sequence diagrams with underspecification, inherent nondeterminism and probabilistic choice. Part 1: underspecification and inherent nondeterminism. Technical Report 346.

In this paper we define how to understand computer systems in terms of the semantic model of STAIRS, and give criteria for when a system is correct with respect to a given interaction containing different kinds of nondeterminism. We discuss different refinement relations and corresponding compliance relations for relating computer systems to interactions, and we explore the mathematical properties of these relations. This is part 1 of the work, discussing underspecification and inherent nondeterminism. In part 2, we discuss probabilistic choice. As this thesis focuses on underspecification and inherent nondeterminism, and not on probabilities or probabilistic choice, this second part is not included here.

My contribution: The paper was written in close collaboration, with me as the main author responsible for approximately 65% of the work.

Paper 8: Ragnhild Kobro Runde. STAIRS case study: The BuddySync System. Technical Report 345.

This is the last paper in the thesis, presenting a case study demonstrating the practical usefulness of STAIRS. The paper also describes some weaknesses identified during the work on the case study, and provides suggestions for improving these weaknesses.

My contribution: I was the sole author.

Chapter 7

Discussion

This chapter contains a more detailed discussion of STAIRS than what has been possible within the context of the attached papers. In section 7.1 we compare the original semantic model of STAIRS as proposed in [HS03] to the one used in our work. In section 7.2 we evaluate STAIRS with respect to the success criteria in section 3.3, while in section 7.3 we discuss to what extent the results obtained for interaction diagrams may be generalized to other diagram types such as UML state machine diagrams. Finally, in section 7.4 we discuss related work.

7.1 The Original Semantic Model of STAIRS

The semantic model proposed in [HS03] is slightly different from our semantic model as presented in section 5.2.2. In this section we first present the semantic model of [HS03] in section 7.1.1. Our main motivation for using another semantic model comes from being able to refine and implement the different `xalt`-operands separately. Hence, in section 7.1.2 we define the refinement notion supplementing and narrowing for the semantic model in [HS03], while section 7.1.3 contains a discussion of alternative definitions of `xalt`.

7.1.1 Semantic Model

The semantic model proposed in [HS03] differs from our semantic model as presented in section 5.2.2 in that in [HS03] there are several positive trace-sets (corresponding to our interaction obligations), but only one negative trace-set. This alternative semantic model is illustrated in figure 7.1.

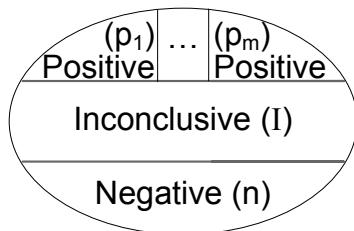


Figure 7.1: Illustrating the semantic model of [HS03]

Formally, the semantics described in [HS03] is a pair (P, n) , where P is a set of $\{p_1, \dots, p_m\}$ sets of positive traces, while n is a set of negative traces. The inconclusive traces are those that are neither in any of the positive trace-sets nor in the negative trace-set, i.e. all traces in the set $I = \mathcal{H} \setminus (n \cup \bigcup_{i \in [1, m]} p_i)$. The intuition behind this semantic model is that a correct implementation should be able to perform at least one trace from each of the positive trace-sets, but none of the negative traces.

Using only one negative trace-set is intuitively appealing, since defining a trace as negative would mean that the system should definitely not be able to perform that trace. In contrast, in our semantic model a trace that is negative in one interaction obligation may still be positive or inconclusive in another, and the trace may or may not be performed by the system.

7.1.2 Refinement

Refinement is only informally explained in [HS03], and not all special cases are treated. The following formalization constitutes our interpretation of the informal exposition in [HS03].

Supplementing means to reduce the set of inconclusive traces by adding former inconclusive traces to the negative trace-set, to one of the positive trace-sets, or as a new positive trace-set. Formally:

$$(P, n) \rightsquigarrow_s (P', n') \stackrel{\text{def}}{=} n \subseteq n' \wedge n' \setminus n \subseteq I \wedge \forall p \in P : \exists p' \in P' : p \subseteq p' \wedge p' \setminus p \subseteq I \quad (7.1)$$

Narrowing means reducing underspecification by reducing one or more of the positive trace-sets without making any of them empty. If, as a result of this, a former positive trace is no longer in any of the positive trace-sets, the trace should be included in the negative trace-set. Formally:

$$(P, n) \rightsquigarrow_n (P', n') \stackrel{\text{def}}{=} n' = n \cup (\bigcup_{p_i \in P} p_i \setminus \bigcup_{p'_i \in P'} p'_i) \wedge \forall p \in P : \exists p' \in P' : p' \subseteq p \wedge p' \neq \emptyset \quad (7.2)$$

One problem with these refinement definitions is that combining supplementing and narrowing is not transitive with respect to the proposed notion of compliance. As an example, let t_1 and t_2 be two different, well-formed, traces and consider the three interactions d_1 , d_2 and d_3 with semantics as follows:

$$\begin{aligned} \llbracket d_1 \rrbracket &= (\{\{t_1\}\}, \emptyset) \\ \llbracket d_2 \rrbracket &= (\{\{t_1, t_2\}\}, \emptyset) \\ \llbracket d_3 \rrbracket &= (\{\{t_2\}\}, \{t_1\}) \end{aligned}$$

It is straightforward to see that d_3 is a narrowing refinement of d_2 , which is a supplementing refinement of d_1 . A system performing only the trace t_2 is in compliance with d_3 (and d_2), but not d_1 as it contains no trace from the positive trace-set $\{t_1\}$.

This problem could be solved for instance by a more relaxed notion of compliance where for each of the positive trace-sets, the system must perform either one of the positive traces or an inconclusive trace. This notion would be closer to our compliance relation for interaction obligations.

7.1.3 Definition of xalt

We now turn to discussing possible definitions of xalt , using the semantic model described above. From the explanations in [HS03], we get that the new set of positive trace-sets is the union of the two sets given by each of its operands. However, it is not obvious what the new negative trace-set should be. The two most obvious choices are using disjunction or union of the two operand sets.

In our chosen semantic model in section 5.2.2, consisting of a set of interaction obligations, a trace is negative for a system only if it is negative in all interaction obligations. In the semantic model of [HS03], this would correspond to using disjunction of the negative trace-sets of the two operands:

$$\llbracket \text{xalt } [d_1, d_2] \rrbracket \stackrel{\text{def}}{=} (P_1 \cup P_2, n_1 \cap n_2) \quad (7.3)$$

where we assume that $\llbracket d_1 \rrbracket = (P_1, n_1)$ and $\llbracket d_2 \rrbracket = (P_2, n_2)$.

However, this definition is not monotonic with respect to the refinement definitions (7.1) and (7.2), meaning that d_1 and d_2 cannot be developed separately. One problem is that in the original specification, a trace may be inconclusive in one operand and positive in the other. As a simple example, consider the interaction

$$d = \text{xalt } [t_1, \text{alt } [t_2, t_3]]$$

with semantics

$$\llbracket d \rrbracket = (\{\{t_1\}, \{t_2, t_3\}\}, \emptyset)$$

A valid narrowing refinement of the operand $\text{alt } [t_2, t_3]$ is the interaction $\text{alt } [t_2, \text{refuse } [t_3]]$. Replacing $\text{alt } [t_2, t_3]$ with its refinement in the original interaction, we get the interaction

$$d' = \text{xalt } [t_1, \text{alt } [t_2, \text{refuse } [t_3]]]$$

with semantics

$$\llbracket d' \rrbracket = (\{\{t_1\}, \{t_2\}\}, \emptyset)$$

However, d' is not a valid refinement of d as the trace t_3 has been moved from positive to inconclusive which is not allowed by either supplementing or narrowing.

Another problem with definition (7.3) is that when refining each of the xalt -operands separately, the positive traces of the first operand may be supplemented as positive in the second operand, and vice versa. Consider again the interaction d in the previous example. A valid supplementing refinement of the operand $\text{alt } [t_2, t_3]$ is the interaction $\text{alt } [t_1, \text{alt } [t_2, t_3]]$. Substituting this into the original interaction, we get the interaction

$$d'' = \text{xalt } [t_1, \text{alt } [t_1, \text{alt } [t_2, t_3]]]$$

with semantics

$$\llbracket d'' \rrbracket = (\{\{t_1\}, \{t_1, t_2, t_3\}\}, \emptyset)$$

Again, this is not a valid refinement of d . We also see that a system performing only the trace t_1 is in compliance with d'' but not with the original interaction d .

Using union instead of disjunction for the negative trace-set, xalt could be defined by:

$$\llbracket \text{xalt } [d_1, d_2] \rrbracket \stackrel{\text{def}}{=} (P_1 \cup P_2, n_1 \cup n_2) \quad (7.4)$$

This definition solves the monotonicity problems with respect to the first, but not the second of the examples above. In fact, with the semantic model used in this section, it is not possible to define $xalt$ in a way that ensures monotonicity when supplementing each operand separately. Instead, a possible solution is using different negative trace-sets for each positive trace-set, as with our interaction obligations.

7.2 Evaluating STAIRS with Respect to the Success Criteria

In this section, we evaluate STAIRS with respect to each of the success criteria given in section 3.3.

- **The formal semantics should take into account the partiality of UML 2.x interactions.**

This is achieved by the semantic model of STAIRS, categorizing behaviours as either positive, inconclusive or negative. As explained in section 5.2.2, one of the main principles behind our formalization is that all traces not explicitly described as positive or negative in a (sub-)interaction are inconclusive for that (sub-)interaction.

- **The formal semantics should handle both positive and negative behaviour.**

Again, this is achieved by the semantic model of STAIRS as described for the previous success criteria.

- **The formal semantics should handle both mandatory and potential behaviour.**

Mandatory behaviour, also referred to as inherent nondeterminism, is specified using $xalt$, and is reflected in the different interaction obligations in the semantic model. Potential behaviour, or underspecification, is specified using alt (or through weak sequencing), and is reflected in each interaction obligation having a set of positive behaviours.

- **The formal semantics should include a notion of time.**

Time is included by giving all events a timestamp in the form of a positive, real number (see paper 2). All events in a trace must be ordered by time, but two events may happen at the same time. Time constraints restrict the valid timestamps of one or more events such that traces with valid timestamps are defined as positive, while traces with invalid timestamps are defined as negative.

- **The formal semantics should be compositional.**

The semantics of an interaction is given in terms of the semantics of its sub-interactions, and the operators used for composing these. All operators except `assert` are monotonic with respect to general refinement, meaning that different sub-interactions may be developed separately. This is proved in papers 2, 4 and 7, which also contains monotonicity proofs for the operators and refinement relations covered by each particular paper. For `assert`, we have monotonicity in the special case of narrowing. As explained in paper 2, the lack of monotonicity with respect to supplementing is not important, as `assert` is usually used to state that all positive behaviours have been defined and that no more supplementing is needed.

- **The formal semantics should be in accordance with the UML 2.x standard.**

We believe that the formalization in STAIRS is faithful to the underlying ideas of UML 2.x, including the partial nature of interactions and using weak sequencing as the main composition operator. However, it has not always been possible to follow the standard in every respect. In particular, we have defined two different operators for negation as explained in paper 3, extended the language with the `xalt` operator, and given a slightly different interpretation of guarded alt in paper 5.

The UML 2.x standard is vague regarding how the different composition operators should be understood with respect to negative behaviours. In this situation, we have tried to follow the other principles given in section 5.2.2, and in particular that of monotonicity.

- **The refinement relations should take into account the partiality of interactions.**

All of the refinement relations in STAIRS acknowledges that an interaction usually does not provide a complete description of the system. In particular, the notion of supplementing may be used to define more traces as positive or negative. Also, the notion of general refinement may be used to add more mandatory behaviours to the interaction.

- **The refinement relations should handle both positive and negative behaviour.**

As the semantic model of STAIRS includes both positive and negative behaviour, so does the refinement relations. Within a single interaction obligation, negative behaviour is interpreted as behaviour that *must not* be present in the system, while positive behaviour is interpreted as behaviour that *may* be present.

- **The refinement relations should handle both mandatory and potential behaviour.**

Refinement of both mandatory and potential behaviour is handled by defining relations for refining both individual interaction obligations and sets of interaction obligations (see section 5.2.3). By narrowing, positive traces in an interaction obligation may be redefined as negative, hence reducing the set of potential behaviours. General and limited refinement requires that all mandatory behaviour specified in the original interaction must be present also in the refinement.

- **The refinement relations should capture the main refinement notions known from classical formal methods.**

Classical refinement in the sense of reducing underspecification is captured by the STAIRS notion of narrowing. In addition, STAIRS includes the notion of supplementing, for adding earlier inconclusive traces to the specification. Interface refinement is partially captured by the STAIRS notion of detailing, but including a more general notion of message refinement in STAIRS would probably be useful.

- **The refinement relations should be transitive.**

All refinement relations are proved to be transitive, see papers 2, 4 and 7.

- **The compliance relation between interactions and computer systems should be a special case of refinement.**

The compliance relations in paper 7 are all special cases of refinement. With the exception of restricted compliance, all compliance relations are special cases of the corresponding refinement relations. In addition, it is proved that all systems that are in compliance with a refinement will also be in compliance with the original interaction.

- **The compliance relation between interactions and computer systems should be independent of technology used for the computer system.**

Technology independence is achieved by assuming that a computer system is given by its set of traces, and then translating this trace-set into the semantics model of STAIRS.

- **The methodology should be conservative, i.e. based on existing UML methodology.**

Within the limits of this thesis, there has not been room for creating a thorough methodology based on STAIRS. What we have, is a number of pragmatic guidelines for creating and refining UML 2.x interactions (see paper 5). In paper 8 we give some initial ideas of how these may be used together with existing development methodologies such as e.g. RUP.

- **The methodology should be useful without thorough knowledge of the formal definitions.**

The guidelines in paper 5 were proven useful in the case study in paper 8. However, they are not complete, and more guidelines must be defined before the principles of STAIRS may be applied to system development by people without knowledge of the formal definitions.

7.3 Generalization of the Results

In our work, we have focused exclusively on UML 2.x interactions in the form of sequence diagrams and interaction overview diagrams. This means that we have only studied semantics and refinement with respect to dynamic behaviour, and not static structure. The static structure diagrams of UML 2.x, such as class diagrams and object diagrams, are also partial descriptions in the same manner as interactions, and the distinction between positive (i.e. possible or valid), inconclusive and negative (i.e. impossible or invalid) may be useful also for such description techniques. However, it is not common to talk about negation in this setting, and it is not obvious how the results of STAIRS may be used for these diagram types.

The interaction diagrams of UML 2.x include also communication diagrams and timing diagrams. These have not been explicitly treated in our work, but it is straightforward to use these diagram types within the context of STAIRS. As with sequence diagrams, communication diagrams specify interaction between objects. The difference between the two is that while sequence diagrams focus on the ordering of events, communication diagrams focus on the relationships between the objects. However, for the behavioural description, any communication diagram may be translated into an equivalent sequence diagram. Similarly, timing diagrams are a special kind of sequence diagrams, explicitly showing the time ticks and also state changes in the lifelines.

The main principles of STAIRS generalize nicely to other approaches for behavioural specification, both within and outside UML. The exact semantics must of course be defined in each

case, but the basic semantic model consisting of several interaction obligations, each distinguishing between positive, inconclusive and negative behaviour, should be applicable for a wide range of languages. In particular, our results with respect to refinement and compliance are applicable for other behavioural specifications as soon as the basic understanding of their semantics is obtained. Usually, only a simpler notion of refinement is needed, as most behavioural description techniques are seen as giving complete specifications where the set of inconclusive behaviour is empty. In those cases, the most interesting contribution of STAIRS is the distinction between mandatory and potential behaviour, and how this difference is treated in the semantic model and by the refinement relations.

7.4 Related Work

In this section we discuss closely related work on UML 2.x interactions. For a general treatment of related work, such as other kinds of sequence diagrams and work on the different kinds of nondeterminism, we refer to chapter 4 and the sections on related work in the attached papers.

7.4.1 Harald Störrle: Trace Semantics of Interactions in UML 2.0

In [Stö04], Harald Störrle defines trace semantics for interactions, based on the UML 2.0 standard [OMG04]. The basic concepts are very similar to those in STAIRS, and the two approaches mainly agree with respect to the definitions of positive behaviour. Neither approach treats extra global combined fragments, but Störrle defines semantics for operators not covered by STAIRS, including consider, ignore and variants of break and critical. Guards are not treated in [Stö04], while time is treated in a manner very similar to that of STAIRS.

Positive behaviours are in [Stö04] interpreted as *must*, similar to our `xalt` operator and analogous to negative behaviours being interpreted as *must not* (in both approaches). Underspecification is not treated in [Stö04].

For negative behaviours, there are also interesting differences. STAIRS and [Stö04] mainly agree on the definition of `assert`, where the only positive traces are the positive traces of the operand. The difference between the two definitions is that if a trace is both positive and negative in the operand, the trace will remain both positive and negative in STAIRS, whereas it becomes only positive in [Stö04].

With respect to `neg`, Störrle investigates three alternative definitions, but neither of these corresponds to the STAIRS definitions of `refuse` and `veto`. For all of the alternatives in [Stö04], the negative traces of `neg` are the positive traces of its operand (and not also the negative behaviours as in STAIRS). The difference between the three alternatives is with respect to what constitutes the positive behaviours. The first alternative in [Stö04], `loose negate`, is similar to our `refuse` in that the set of positive behaviours are empty. With this definition, the negative traces of the operand is “lost”, i.e. they become inconclusive. Therefore, a second alternative, `strict negate`, is proposed for which all traces that are not negative in the operand are taken as positive (i.e. `neg` is taken as the opposite of `assert`). The third alternative, `flip negate`, takes the negative traces of the operand as positive, corresponding to negation in classical logic. This alternative seems to be the one favoured by Störrle. Our reasons for not choosing this definition are given in paper 3.

[Stö04] also discusses some problems of combining `neg` with other operators. For instance,

an interaction such as $\text{seq } [d_1, \text{neg } [d_2]]$ intuitively means that the behaviours of d_2 should *not* follow the positive behaviours of d_1 . However, using the definitions in [Stö04], the only negative behaviours of this interaction is the *negative* behaviours of d_1 sequenced with the negative behaviours of d_2 . These problems lead Störrle to conclude that “maybe, the concept of negative traces is not such a good idea after all”. We do not agree with this statement, and the formal definitions in STAIRS do not have similar problems as the ones discussed in [Stö04].

For refinement, [Stö04] defines refinement of traces, refinement of interactions, and refinement of time constraints. A single trace may be refined by adding new events to it, or replacing coarse-grained events by sequences of fine-grained events. This may be seen as a generalization of detailing refinement in STAIRS. For refining interactions, Störrle only considers a notion similar to STAIRS supplementing, as there are no underspecification in the interactions. Finally, a time constraint may be refined by narrowing the valid timestamps for the events constrained by it. In STAIRS, this is captured by the general definition of narrowing.

7.4.2 María Victoria Cengarle and Alexander Knapp: UML 2.0 Interactions — Semantics and Refinement

Another trace-based formalization of UML 2.0 interactions is the work by Cengarle and Knapp in [CK04]. The operators consider and ignore are treated, but not break or critical as in [Stö04]. Extra global combined fragments are not treated, neither are time nor guards.

For the positive traces of an interaction, the semantics in [CK04] mainly coincides with that in [Stö04] (and STAIRS), except that having several positive traces is interpreted as underspecification, similar to alt in STAIRS. Also, the empty trace is taken as the only positive trace for the interaction neg $[d]$, similar to our veto operator.

For the negative traces of an interaction, there are several differences between STAIRS and the approach in [CK04]. The definitions of assert are similar, but for negation, Cengarle and Knapp takes only the positive traces of the operand as negative, similar to Störrle. For alt, a trace is taken as negative only if it is negative in both operands.

Cengarle and Knapp pose an interesting question with respect to the use of negation in specifications: Should a trace be negative if a prefix of it is specified as negative? Their answer is essentially yes, proposing an even stronger approach where a trace is taken as negative as soon as it has completed a negative sub-interaction. An advantage of this is that it allows for earlier identification (or even prevention) of negative traces. With this approach, there will typically be many traces that are negative for an interaction even though the traces are not explicitly described in the diagram. In STAIRS, we follow our main principles as stated in section 5.2.2 and regard these traces as inconclusive. For a further discussion of these alternative approaches, see the discussion of related work in paper 3.

For Cengarle and Knapp, a valid implementation of an interaction must show at least one positive trace and no negative traces. This is similar to the STAIRS notion of restricted compliance (see paper 7), which is indeed inspired by the notion in [CK04].

The refinement notion in [CK04] is based on a model-theoretic view, and states simply that one interaction is a refinement of another interaction if all valid implementations of the refinement are also valid implementations of the original interaction. This implies that at least one of the positive traces of the initial specification must remain positive during all of the develop-

ment process, and that supplementing inconclusive traces as positive is not allowed (similar to our notion of restricted refinement). In contrast to all of the refinement relations in STAIRS, refinement in [CK04] allows positive traces to become inconclusive.

A major disadvantage with the refinement relation in [CK04] is that it does not give monotonicity for all of the composition operators. In an attempt to remedy this, Cengarle and Knapp define a restricted refinement notion, positive refinement, where the set of positive traces are kept unchanged during refinement (i.e. the only possible refinement step is supplementing traces as negative). This refinement notion gives monotonicity for all operators under the assumption that all refinements are implementable and does not contain the same trace as both positive and negative. With this refinement notion, an implementation may still remove underspecification by implementing only some of the positive traces, but refinement can no longer be used to resolve this at the specification level.

7.4.3 Other Work on UML 2.x Interactions

Other recent work on UML 2.x interaction include [GS05], [EFM⁺05] and [HM06]. As these deviate more from the UML 2.x standard, they are treated here with less detail than the two approaches above.

[GS05] interprets positive and negative interactions as specifying liveness and safety properties, respectively. This is a much stronger interpretation than the traditional use of sequence diagrams for illustrating example runs. Based on a large amount of transformation, [GS05] then defines the semantics of interactions in the form of two Büchi automata, one for the positive and one for the negative behaviour. Refinement is defined as language inclusion, and is monotonic with respect to the most common composition operators.

[EFM⁺05] uses Petri Nets to give semantics to UML 2.x interactions. However, the approach deviates from UML on important points. First of all, it is assumed that all possible behaviours are explicitly described in the diagram (i.e. there are no inconclusive behaviours), meaning that operators such as assert and neg are not defined as there is no need for specifying negative behaviours. Also, in contrast to the UML 2.x standard, synchronization between all lifelines are assumed at the beginning of each sub-interaction. Still, [EFM⁺05] is interesting as it includes aspects of sequence diagrams not treated in most other formal approaches, including lost and found messages, and the creation and destruction of lifelines. Data in sequence diagrams is also covered, but not time. No notion of refinement is included in [EFM⁺05].

[HM06] argues that the assert and neg operators are insufficient for specifying required and forbidden behaviours, and proposes Modal UML Sequence Diagrams (MUSD) as an extension to UML 2.x sequence diagrams. Based on LSC ([DH01, HM03], see section 4.6), MUSD allows sequence diagram elements to be specified as either hot (universal) or cold (existential). With this approach, assert is interpreted as specifying that all of the events in the operand should be hot, while neg is interpreted as if the condition *false* was added immediately after the last event(s) in the operand. This interpretation of neg leads to the same approach as that proposed in [CK04], where a trace is negative as soon as it has completely traversed a negative (sub-)interaction.

Chapter 8

Future Work

As mentioned in chapter 1 this work has already been used as a basis for other work on UML 2.x interactions including test case generation [LS06a], probabilistic sequence diagrams [RHS05, RRS06] and for defining secure information flow [SS06]. Also, we have been involved in work on extending sequence diagrams with time exceptions for handling violation of time constraints [HRH07].

This work may also be extended in several other directions, some of which are hinted at in sections 7.2 and 7.3. One possible direction is continuing the work on providing a formal semantics for UML 2.x interactions. Although we have covered what we believe is the most commonly used parts of interactions, interesting aspects such as critical region and extra global combined fragments are not treated, and their formalization is not entirely obvious. In section 3 we identified a need for new or improved UML tools based on a precise semantics for UML 2.x. Providing a semantics for interactions is only the first step. Covering all of UML 2.x is a large and complex work, which is currently being addressed by the UML 2 semantics project [BCD⁺07].

A more interesting direction is extending our work on refinement, both with respect to the formal definitions and with respect to the pragmatic guidelines explaining them. Formally, we believe that we have covered the interesting variations of behavioural refinement (i.e. supplementing, narrowing and restricted refinement), but more work is needed with respect to message refinement. As a step towards defining more pragmatical guidelines for refinement, developing a formal refinement calculus for STAIRS would be very useful. Also, guidelines could be developed with respect to when the different refinement and compliance notions are most successfully used in practical system development.

For relating different specifications of the same system there is also the notion of viewpoint correspondences. For a simple notion of viewpoint consistency, we may use STAIRS and say that two specifications are consistent if there exists a specification that is a valid refinement of both of the original specifications. However, viewpoint correspondences involves more than just simple consistency. We believe that STAIRS provides a useful basis for investigating these issues, but much more work is needed here.

Bibliography

- [ABB⁺02] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jürgen Wüst, and Jörg Zettel. *Component-based Product Line Engineering with UML*. Addison-Wesley, 2002.
- [AH92] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *Real-Time: Theory in Practice*, volume 600 of *LNCS*, pages 74–106. Springer, 1992.
- [AL91] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.
- [Amb02] Scott W. Ambler. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. Wiley, 2002.
- [BCD⁺07] Manfred Broy, Michelle L. Crane, Juergen Dingel, Alan Hartman, Bernhard Rumpe, and Bran Selic. 2nd UML 2 semantics symposium: Formal semantics for UML. In *Models in Software Engineering*, volume 4364 of *LNCS*, pages 318–323. Springer, 2007.
- [BS01] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems : Foundations on Streams, Interfaces, and Refinement*. Springer, 2001.
- [BvW98] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus: A Systematic Introduction*. Springer, 1998.
- [CK04] María Victoria Cengarle and Alexander Knapp. UML 2.0 interactions: Semantics and refinement. In *Proc. Critical Systems Development with UML (CSDUML'04)*, Technical report TUM-I0415, pages 85–99. Institut für Informatik, Technische Universität München, 2004.
- [Col93] Pierre Collette. Application of the composition principle to unity-like specifications. In *Proc. TAPSOFT'93: Theory and Practice of Software Development*, volume 668 of *LNCS*, pages 230–242. Springer, 1993.
- [DC02] Gordana Dodig-Crnkovic. Scientific methods in computer science. In *Proc. Conf. for the promotion of research in IT at new universities and at university colleges in Sweden*, 2002.

- [DDH72] Ole-Johan Dahl, Edsger W. Dijkstra, and C. A. R. Hoare. *Structured Programming*. Academic Press, 1972.
- [Den05] Peter J. Denning. Is computer science science? *Communications of the ACM*, 48(4):27–31, 2005.
- [DH99] Werner Damm and David Harel. LSCs: Breathing life into Message Sequence Charts. In *Proc. Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, pages 293–312. Kluwer, 1999.
- [DH01] Werner Damm and David Harel. LSCs: Breathing life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001. A preliminary version of this paper appeared as [DH99].
- [Dij68] Edsger W. Dijkstra. Stepwise program construction. Published as [Dij82], February 1968.
- [Dij82] Edsger W. Dijkstra. Stepwise program construction. In *Selected Writings on Computing: A Personal Perspective*, pages 1–14. Springer, 1982.
- [dJvdPH00] Edwin de Jong, Jaco van de Pol, and Jozef Hooman. Refinement in requirements specification and analysis: A case study. In *Proc. Engineering of Computer Based Systems (ECBS 2000)*, pages 290–298. IEEE Computer Society, 2000.
- [dR85] Willem-Paul de Roever. The quest for compositionality — A survey of assertion-based proof systems for concurrent programs. Part 1: Concurrency based on shared variables. Technical Report RUU-CS-85-2, Department of Computer Science, University of Utrecht, 1985.
- [dRdBH⁺01] Willem-Paul de Roever, Frank de Boer, Ulrich Hannemann, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers. *Concurrency verification: Introduction to Compositional and Noncompositional Methods*. Cambridge University Press, 2001.
- [dRE98] Willem-Paul de Roever and Kai Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge University Press, 1998.
- [DW99] Desmond Francis D’Souza and Alan Cameron Wills. *Objects, Components, and Frameworks with UML: the Catalysis Approach*. Addison-Wesley, 1999.
- [EFM⁺05] Christoph Eichner, Hans Fleischhack, Roland Meyer, Ulrik Schrimpf, and Christian Stehno. Compositional semantics for UML 2.0 sequence diagrams using Petri Nets. In *Proc. SDL 2005: Model Driven Systems Design*, volume 3530 of *LNCS*, pages 133–148. Springer, 2005.
- [Fow03] Martin Fowler. Uml blik: UmlMode, May 2003. <http://www.martinfowler.com/bliki/UmlMode.html>.

- [GHK99] Günter Graw, Peter Herrmann, and Heiko Krumm. Constraint-oriented formal modelling of OO-systems. In *Proc. Distributed Applications and Interoperable Systems (DAIS'99)*, pages 345–358. Kluwer, 1999.
- [Gla95] Robert L. Glass. A structure-based critique of contemporary computing research. *Journal of Systems and Software*, 28(1):3–7, 1995.
- [GS05] Radu Grosu and Scott A. Smolka. Safety-liveness semantics for UML 2.0 sequence diagrams. In *Proc. Applications of Concurrency to System Design (ACSD'05)*, pages 6–14. IEEE Computer Society, 2005.
- [Hau97] Øystein Haugen. The MSC-96 distillery. In *SDL'97: Time for Testing: SDL, MSC and Trends*. Elsevier, 1997.
- [Hau04] Øystein Haugen. Comparing UML 2.0 interactions and MSC-2000. In *Proc. System Analysis and Modeling (SAM 2004)*, volume 3319 of *LNCS*, pages 65–79. Springer, 2004.
- [HM03] David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.
- [HM06] David Harel and Shahar Maoz. Assert and negate revisited: Modal semantics for UML sequence diagrams. In *Proc. Scenarios and State Machines: Models, Algorithms, and Tools (SCESM'06)*, pages 13–20. ACM, 2006.
- [Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.
- [Hoa72] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.
- [Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [HRH07] Oddleif Halvorsen, Ragnhild Kobro Runde, and Øystein Haugen. Time exceptions in sequence diagrams. In *Models in Software Engineering*, volume 4364 of *LNCS*, pages 131–142. Springer, 2007.
- [HS03] Øystein Haugen and Ketil Stølen. STAIRS — Steps to analyze interactions with refinement semantics. In *Proc. "UML 2003" — The Unified Modeling Language: Modeling Languages and Applications*, volume 2863 of *LNCS*, pages 388–402. Springer, 2003.
- [ISO89] International Standards Organization. *Information Processing Systems – Open Systems Interconnection – LOTOS – a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour – ISO 8807*, 1989.
- [ITU96] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1996.

- [ITU98] International Telecommunication Union. *Recommendation Z.120 — Annex B: Formal semantics of Message Sequence Charts*, 1998.
- [ITU99] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1999.
- [JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.
- [Jon72] Cli B. Jones. Formal development of correct algorithms: An example based on Earley's recogniser. *ACM SIGPLAN Notices*, 7(1):150–169, 1972.
- [Jür01] Jan Jürjens. Secrecy-preserving refinement. In *Proc. FME 2001: Formal Methods for Increasing Software Productivity*, volume 2021 of *LNCS*, pages 135–152. Springer, 2001.
- [Jür02] Jan Jürjens. *Principles for Secure Systems Design*. PhD thesis, Oxford University Computing Laboratory, 2002.
- [Jür05] Jan Jürjens. *Secure Systems Development with UML*. Springer, 2005.
- [Kna99] Alexander Knapp. A formal semantics for UML interactions. In *Proc. "UML"’99: The Unified Modeling Language: Beyond the Standard*, volume 1723 of *LNCS*, pages 116–130. Springer, 1999.
- [Krü00] Ingolf Heiko Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Institut für Informatik, Technische Universität München, 2000.
- [Kru04] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, third edition, 2004.
- [Lam02] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.
- [Lam05] Leslie Lamport. Real time is really simple. Technical Report MSR-TR-2005-30, Microsoft Research, 2005.
- [LAMB89] Peter Gorm Larsen, Michael Meincke Arentoft, Brian Q. Monahan, and Stephen Bear. Towards a formal semantics of the BSI/VDM specification language. In *Information processing 89: Proc. IFIP 11th World Computer Congress*, pages 95–100. Elsevier, 1989.
- [LS06a] Mass Soldal Lund and Ketil Stølen. Deriving tests from UML 2.0 sequence diagrams with neg and assert. In *Proc. 1st International Workshop on Automation of Software Test (AST’06)*, pages 22–28. ACM Press, 2006.
- [LS06b] Mass Soldal Lund and Ketil Stølen. A fully general operational semantics for UML 2.0 sequence diagrams with potential and mandatory choice. In *Proc. FM 2006: Formal Methods*, volume 4085 of *LNCS*, pages 380–395. Springer, 2006.

- [MBD00] Ralph Miarka, Eerke Boiten, and John Derrick. Guards, preconditions, and refinement in Z. In *Proc. ZB 2000: Formal Specification and Development in Z and B*, volume 1878 of *LNCS*, pages 286–303. Springer, 2000.
- [McG84] Joseph Edward McGrath. *Groups: Interaction and Performance*. Prentice-Hall, 1984.
- [MS00] David Meier and Beverly Sanders. Composing leads-to properties. *Theoretical Computer Science*, 243(1-2):339–361, 2000.
- [OMG03a] Object Management Group. *U2 Partners' UML 2.0 Superstructure Specification*, document: ad/03-04-01 edition, 2003.
- [OMG03b] Object Management Group. *UML 1.5 Specification*, document: formal/03-03-01 edition, 2003.
- [OMG04] Object Management Group. *UML 2.0 Superstructure Specification*, document: ptc/04-10-02 edition, 2004.
- [OMG05] Object Management Group. *UML 2.0 Superstructure Specification*, document: ptc/05-07-04 edition, 2005.
- [OMG06] Object Management Group. *UML 2.1 Superstructure Specification*, document: ptc/06-04-02 edition, 2006.
- [Öve99] Gunnar Övergaard. A formal approach to collaborations in the Unified Modeling Language. In *Proc. "UML"99: The Unified Modeling Language: Beyond the Standard*, volume 1723 of *LNCS*, pages 99–115. Springer, 1999.
- [PP05] Dan Pilone and Neil Pitman. *UML 2.0 in a Nutshell*. O'Reilly, 2005.
- [Pri00] Andreas Prinz. Formal semantics of specification languages. *Teletronikk*, (4), 2000.
- [RHS05] Atle Refsdal, Knut Eilif Husa, and Ketil Stølen. Specification and refinement of soft real-time requirements using sequence diagrams. In *Proc. Formal Modeling and Analysis of Timed Systems (FORMATS 2005)*, volume 3829 of *LNCS*, pages 32–48. Springer, 2005.
- [Ros95] A. W. Roscoe. CSP and determinism in security modelling. In *Proc. 1995 IEEE Symposium on Security and Privacy*, pages 114–127. IEEE Computer Society, 1995.
- [Ros98] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1998.
- [RRS06] Atle Refsdal, Ragnhild Kobro Runde, and Ketil Stølen. Underspecification, inherent nondeterminism and probability in sequence diagrams. In *Proc. Formal Methods for Open Object-Based Distributed Systems (FMOODS 2006)*, volume 4037 of *LNCS*, pages 138–155. Springer, 2006.

- [SAR] The SARDAS project: Securing availability by robust design, assessment and specification. <http://heim.ifi.uio.no/~ketils/sardas/sardas.htm>.
- [SBDB97] Maarten Steen, Howard Bowman, John Derrick, and Eerke Boiten. Disjunction of LOTOS specifications. In *Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE X / PSTV XVII '97)*, pages 177–192. Chapman & Hall, 1997.
- [Sch96] David A. Schmidt. Programming language semantics. *ACM Computing Surveys*, 28(1):265–267, 1996.
- [Sel04] Bran Selic. On the semantic foundations of standard UML 2.0. In *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *LNCS*, pages 181–199. Springer, 2004.
- [SS06] Fredrik Seehusen and Ketil Stølen. Information flow property preserving transformation of UML interaction diagrams. In *Proc. Symposium on Access Control Models and Technologies (SACMAT 2006)*, pages 150–159. ACM, 2006.
- [SS07] Ida Solheim and Ketil Stølen. Technology research explained. Technical Report A313, SINTEF ICT, January 2007.
- [Stö04] Harald Störrle. Trace semantics of interactions in UML 2.0. Technical Report TR 0403, University of Munich, 2004.
- [Whi02] Jon Whittle. Formal approaches to systems analysis using UML: An overview. In *Advanced Topics in Database Research, Vol. 1*, pages 324–341. Idea Group, 2002.
- [Wir71] Niklaus Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14(4):221–227, 1971.
- [WM01] M. Walicki and S. Meldal. Nondeterminism vs. underspecification. In *Proc. Systems, Cybernetics and Informatics (ISAS-SCI 2001)*, pages 551–555. IIIS, 2001.

Part II

Research Papers

Chapter 9

STAIRS towards Formal Design with Sequence Diagrams

Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen

Publication.

Published in *Journal of Software and Systems Modeling*, 22(4):349–458, 2005.

Abstract.

The paper presents STAIRS [HS03], an approach to the compositional development of UML interactions supporting the specification of mandatory as well as potential behavior. STAIRS has been designed to facilitate the use of interactions for requirement capture as well as test specification. STAIRS assigns a precise interpretation to the various steps in incremental system development based on an approach to refinement known from the field of formal methods and provides thereby a foundation for compositional analysis. An interaction may characterize three main kinds of traces. A trace may be (1) positive in the sense that it is valid, legal or desirable, (2) negative meaning that it is invalid, illegal or undesirable, or (3) inconclusive meaning that it is considered irrelevant for the interaction in question. The basic increments in system development proposed by STAIRS, are structured into three main kinds referred to as supplementing, narrowing and detailing. Supplementing categorizes inconclusive traces as either positive or negative. Narrowing reduces the set of positive traces to capture new design decisions or to match the problem more adequately. Detailing involves introducing a more detailed description without significantly altering the externally observable behavior.

Keywords.

UML interactions, formal semantics, explicit non-determinism, refinement, sequence diagrams

9.1 Introduction

A UML interaction is a specification of how messages are sent between objects or other instances to perform a task. Interactions are used in a number of different situations. They are used to get a better grip of a communication scenario for an individual designer or for a group that needs to achieve a common understanding of the situation. Interactions are also used during the more detailed design phase where the precise inter-process communication must be set up according to formal protocols. When testing is performed, the behavior of the system can be described as interactions and compared with those of the earlier phases.

Interactions seem to have the ability to be understood and produced by professionals of computer systems design as well as potential end-users and stakeholders of the (future) systems.

Interactions will typically not tell the complete story. There are normally other legal and possible behaviors that are not contained within the described interactions. Some people find this disturbing and some project leaders have even tried to request that all possible behaviors of a system should be documented through interactions in the form of e.g. sequence diagrams or similar notations.

Our position is that UML interactions are expressed through notations that lend themselves well to conveying important information about the interplay between objects, but interactions are not so well suited to define the complete behavior.

Partial information is not worthless because it is incomplete. Most statements about a system are partial in their nature. The informal statement “When pushing the ON-button on the television, the television should show a program” is definitely not a complete definition of a television set, but it is a piece of requirement that the TV vendors should take into account. The same can be said for interactions; they are partial statements about a system (or a part of a system) defining properties of the system, but not necessarily all relevant properties.

This paper advocates an approach, in the following referred to as STAIRS, aiming to provide a formal foundation for the use of UML interactions in step-wise, incremental system development. STAIRS views the process of developing the interactions as a process of learning through describing. From a fuzzy, rough sketch, the aim is to reach a precise and detailed description applicable for formal handling. To come from the rough and fuzzy to the precise and detailed, STAIRS distinguishes between three main sub-activities: (1) supplementing, (2) narrowing and (3) detailing.

Supplementing categorizes (to this point) inconclusive behavior as either positive or negative recognizing that early descriptions normally lack completeness. The initial requirements concentrate on the most obvious normal situations and the most obvious exceptional ones. Supplementing supports this by allowing less obvious situations to be treated later. Narrowing means reducing the allowed behavior to match the problem better. Detailing involves introducing a more detailed description without significantly altering the externally observable behavior.

Although the starting point for STAIRS is UML interactions in shape of UML 2.0 sequence diagrams, the approach should be considered generic to all types of behaviors.

The remainder of the paper is structured into seven sections. Section 9.2 provides further motivation and background in the form of requirements we would like STAIRS to fulfill. Section 9.3 explains how STAIRS meets these requirements. Sections 9.4, 9.5 and 9.6 spell out STAIRS in an example-driven manner addressing respectively the notions of supplementing, narrowing and detailing. Section 9.7 describes the formal semantics of STAIRS. Section 9.8 provides a brief

summary and relates STAIRS to approaches known from the literature.

9.2 Requirements to STAIRS

In order to explain its overall structure and architecture, we formulate and motivate a number of requirements that STAIRS has been designed to fulfill.

1. Should allow specification of potential behavior.

Under-specification is a well-known feature of abstraction. In the context of interactions, “under-specification” means specifying several behaviors, each representing a potential alternative serving the same purpose, and that fulfilling only some of them (more than zero but not all) is acceptable for an implementation to be correct.

2. Should allow specification of mandatory behavior.

Under-specification as described in the previous paragraph gives rise to non-determinism in the specification. Under-specification allows the system developer to choose between several potential behaviors. Sometimes, however, it is essential to retain non-determinism in the implementation reflecting choice. For example, in a lottery, it is critical that every lottery ticket has the possibility to win the prizes. Otherwise, the lottery is not fair. This means that every behavior given by the different tickets should appear as possibilities in an implementation even though in any given execution only a few prizes are awarded. It seems unproblematic to reduce non-determinism if the different alternatives represent implementation dependent variations of the same behavior. It is quite different, however, to reduce non-determinism if each alternative represents a distinct and intended behavior. As a consequence, we need to distinguish explicit non-determinism capturing mandatory behavior from non-determinism expressing potential behavior.

3. Should allow specification of negative behavior in addition to positive behavior.

Interactions are not only suited to capture system requirements. They may just as well describe illegal or undesirable behavior. For example, security is a major issue in most modern systems. To identify security requirements, risk analysis is a well-known measure. To be able to assign risk values to risks we need a clear understanding of the circumstances under which the risks may appear, and the impact they may have on system assets. Interactions are well suited to describe this kind of threat scenario. Hence, we need an integrated approach to specifying negative as well as positive behavior.

4. Should capture the notion of refinement.

The notion of refinement was formalized in the early 1970s [Mil71, Hoa72, Jon72], and has since then been thoroughly investigated within numerous so-called formal methods. STAIRS should build on this theory, but the theory must be adapted to take into account that interactions may be partial, describe positive as well as negative situations, and may be used to formalize both potential and mandatory behavior.

5. Should formalize aspects of incremental development.

Incremental development of interactions involves various sub-activities as described informally in the introduction. STAIRS should provide precise and intuitive definitions of these activities.

6. Should support compositional analysis, verification and testing.

Models are of little help if they cannot be used as a basis for analysis, verification and testing. STAIRS should provide a foundation for these activities facilitating compositionality [Jon81, dR85] in the sense that components can be developed independently from their specifications.

9.3 How STAIRS Meets the Requirements

The most visible aspects of a UML interaction are the messages between the lifelines. The sequence of the messages is considered important for the understanding of the situation. The data that the messages convey may also be very important, but the interactions do not focus on the manipulation of data even though data can be used to decorate the diagrams.

The sequencing is the heart of what is explained through an interaction. The possible flows of control throughout the process are described in two dimensions, the horizontal dimension showing the different active objects, and the vertical dimension showing the ordering in time.

Interactions focus on the interplay between objects. In the tradition of telecommunications these objects are independent and themselves active as stand-alone processes. Therefore, when a message is sent from one lifeline to another, what happens on the sending lifeline is independent from what happens on the receiving side. The only invariant is that the sending of a message must occur before the reception of that very message. Most people find this obvious.

The sending of a message and the reception of a message are examples of what we call events. An event is something that happens on a lifeline at one point in time. An event has no duration.

A trace is a sequence of events ordered by time. A trace describes the history of message-exchange corresponding to a system run. A trace may be partial or total. Interactions may be timed in the sense that they contain explicit time constraints. Although STAIRS with some minor adjustments carry over to timed interactions (see [HHR04]), such interactions are not treated in this paper.

9.3.1 Spelling out the Trace Semantics of UML 2.0

In this section we will give a very brief introduction to the trace semantics of UML 2.0 interactions expressed in sequence diagrams and interaction overview diagrams [OMG04]. For more on UML 2.0, see also [HMPW03].

The interaction in Fig. 9.1 is almost the simplest interaction there is – only one message from one lifeline to another. Following our introduction above, this message has two events – the sending event on L_1 (which we here choose to denote $!x$) and the reception event on L_2 (which we choose to denote $?x$). The sending event must come before the receiving event and the semantics of this interaction is described by one single trace which we denote $\langle !x, ?x \rangle$.

The interaction of Fig. 9.2 shows two messages both originating from L_1 and targeting L_2 . The order of the events on each lifeline is given by their vertical positions, but the two lifelines

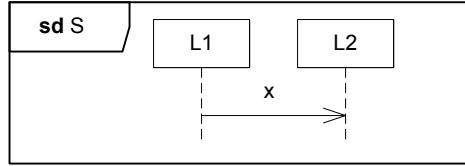


Figure 9.1: Simple interaction with only one message

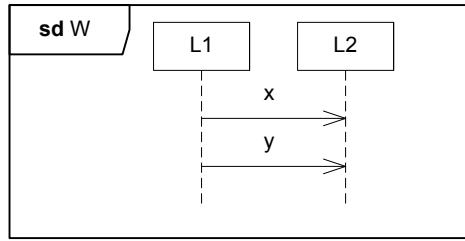


Figure 9.2: Weak sequencing

are independent. Each of the messages has the semantics given for the message in Fig. 9.1, and they are combined with what is called weak sequencing. Weak sequencing takes into account that L_1 and L_2 are independent. The weak sequencing operator on two interactions as operands is defined by the following invariants:

1. The ordering of events within each of the operands is maintained in the result.
2. Events on different lifelines from different operands may come in any order.
3. Events on the same lifeline from different operands are ordered such that an event of the first operand comes before that of the second operand.

Thus, if we denote the weak sequencing operator by seq according to UML 2.0, we get:

$$\begin{aligned} W &= \langle !x, ?x \rangle \text{seq} \langle !y, ?y \rangle \\ &= \{ \langle !x, ?x, !y, ?y \rangle, \langle !x, !y, ?x, ?y \rangle \} \end{aligned}$$

The sending of x must be the first event to happen, but after that either L_1 may send y or L_2 may receive x .

In Fig. 9.3 we show a construct called an interaction occurrence. The interaction S specified in Fig. 9.1 is referenced from within IO . Intuitively, an interaction occurrence is merely shorthand for the contents of the referenced interaction. Semantically we get that:

$$\begin{aligned} IO &= S \text{seq} \langle !y, ?y \rangle \\ &= \langle !x, ?x \rangle \text{seq} \langle !y, ?y \rangle \\ &= \{ \langle !x, ?x, !y, ?y \rangle, \langle !x, !y, ?x, ?y \rangle \} \end{aligned}$$

In Fig. 9.4 we introduce another construct called combined fragment. Combined fragments are expressions of interactions combined differently according to which operator is used. In fact,

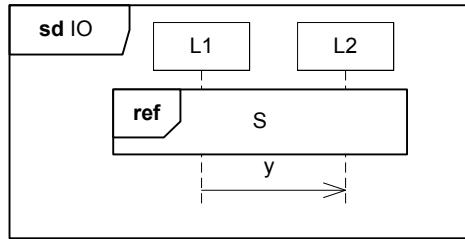


Figure 9.3: Interaction occurrence

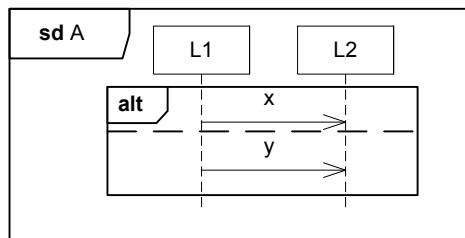


Figure 9.4: Combined fragment (alternative)

also weak sequencing is such an operator. In Fig. 9.4 we have an alternative combined fragment, and its definition is simply the union of the traces of its operands. The dashed vertical line separates the operands. We get:

$$\begin{aligned} A &= \langle !x, ?x \rangle \text{alt} \langle !y, ?y \rangle \\ &= \{ \langle !x, ?x \rangle, \langle !y, ?y \rangle \} \end{aligned}$$

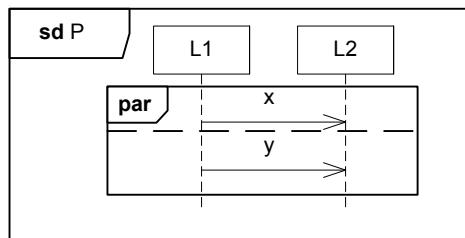


Figure 9.5: Parallel combined fragment

UML 2.0 defines also a number of other operators, but in this paper we only apply one other, namely the parallel merge operator as depicted in Fig. 9.5. The definition of parallel merge says that a parallel merge defines a set of traces that describes all the ways that events of the operands may be interleaved without obstructing the order of the events within the operand. This gives

the following traces for P :

$$\begin{aligned} P &= \langle !x, ?x \rangle \text{par} \langle !y, ?y \rangle \\ &= \{ \langle !x, ?x, !y, ?y \rangle, \langle !x, !y, ?x, ?y \rangle, \\ &\quad \langle !x, !y, ?y, ?x \rangle, \langle !y, ?y, !x, ?x \rangle, \\ &\quad \langle !y, !x, ?y, ?x \rangle, \langle !y, !x, ?x, ?y \rangle \} \end{aligned}$$

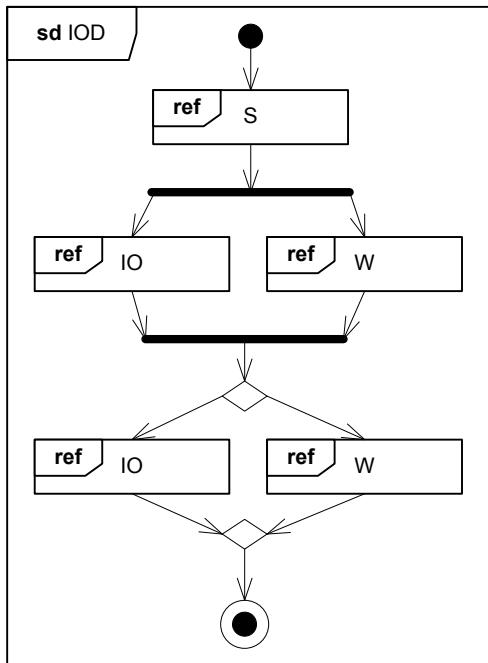


Figure 9.6: Interaction overview diagram

Finally we show the alternative syntax of interaction overview diagrams for these combined fragments. Fig. 9.6 presents a more complicated interaction with the definition:

$$IOD = S \text{ seq } (IO \text{ par } W) \text{ seq } (IO \text{ alt } W)$$

where the diamond joins represent alternatives and the vertical bars represent parallel merge. We have not taken the time and space to calculate the explicit traces, but it is a mechanical task that only requires patience or tool support.

9.3.2 Capturing Positive Behavior

To illustrate our approach we use an everyday example that we hope seems intuitive. We describe the behavior of an ATM (Automatic Teller Machine). The ATM offers withdrawal of native money or the purchase of a number of foreign currencies. We have specified euros (EUR) or US dollars (USD). The ATM must also have cash refill such that the customers can get what they order.

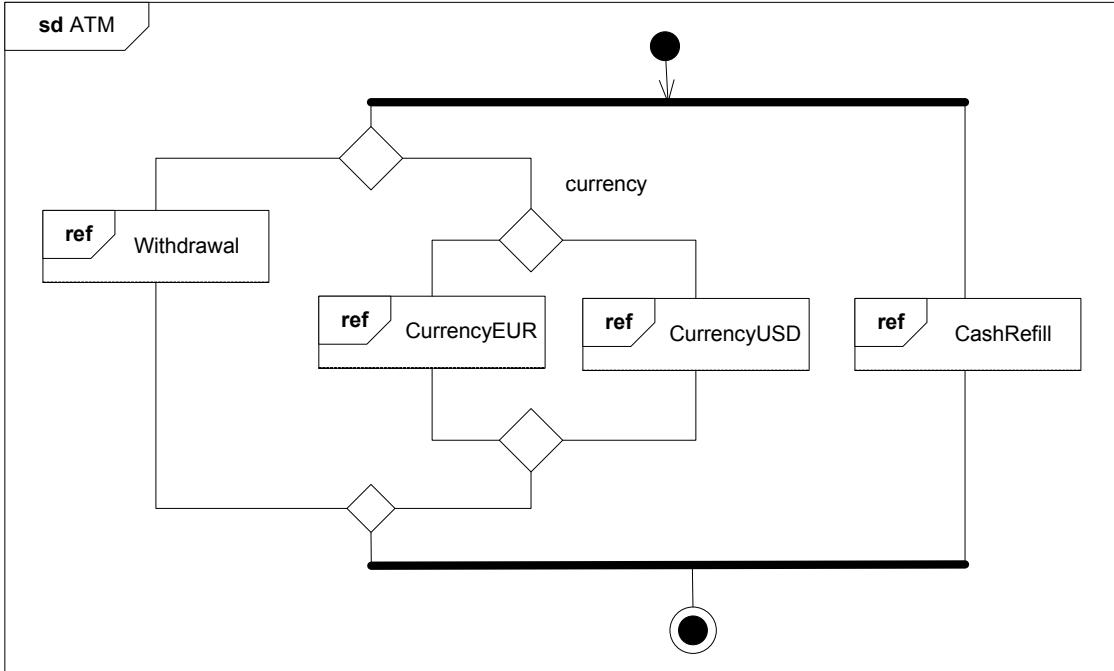


Figure 9.7: Automatic teller machine

In an interaction overview diagram the behavior of the ATM may look like Fig. 9.7. To look into the details down to the events, we can follow e.g. Withdrawal as shown in Fig. 9.8.

Our withdrawal sequence diagram shows a trace that makes up some of the traces of the full ATM. Our withdrawal sequence is a positive one, one that is acceptable to the customers and as such desirable. It does not define all possible scenarios of a withdrawal of native money.

Each of the interaction occurrences in Fig. 9.7 represents a set of positive traces. The vertical bars between the withdrawal side and refill side represents a parallel merge combination meaning that all traces of the withdrawal are braided (interleaved) with every trace of the refill order. This must in practice be restricted, but that is not significant for the subject of this paper. The branching within the withdrawal side represents alternative choices and their combination is essentially a union of traces. Abbreviating the subsequences such that W stands for Withdrawal, E for CurrencyEUR, U for CurrencyUSD and C for CashRefill and then using the operators of UML 2.0 combined fragment, the following formula defines the positive traces:

$$ATM = ((W \text{ alt } (E \text{ alt } U)) \text{ par } C)$$

To expand this to a set of traces, all the interactions referenced must be defined and the operations applied according to the UML 2.0 definition. We believe that the contained traces of the behavior of the ATM are intuitively understood.

Summary 1 *Semantically, each positive behavior is represented by a trace. Considering positive (potential) behavior only, the semantics of an interaction may be represented by a set of traces, each capturing a (potential) positive behavior.*

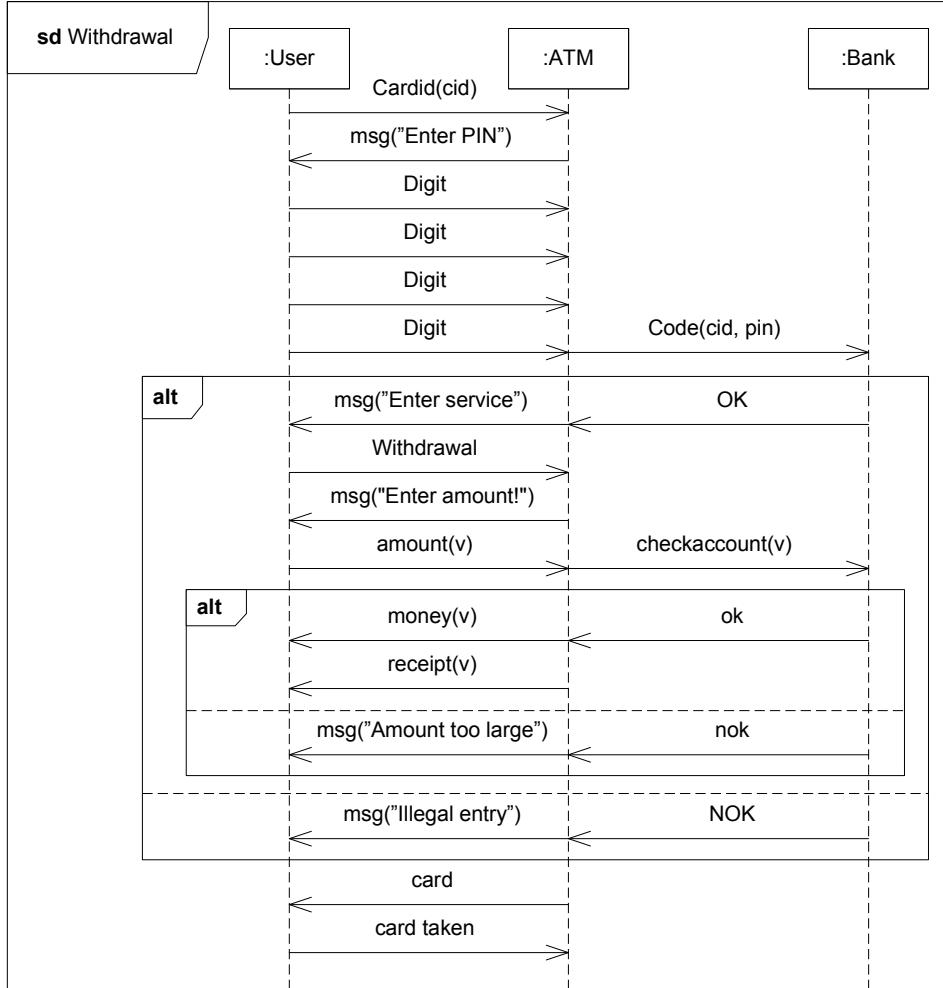


Figure 9.8: Withdrawal (positive traces)

9.3.3 Capturing Negative Behavior

In general the withdrawal procedure may include more than a single trace which is shown in Fig. 9.8 by the fact that there are alternative courses of the traces given by the alternative combined fragments. We have described both situations where the user is eligible for money and when his identification or funds are inadequate. These are all normal situations of an ATM and as such are considered “positive” as they will occur in an implementation.

On the other hand what should not be expected of an ATM is that the machine pretends to function correctly, but the user receives no money. In Fig. 9.9 we show a more elaborated scenario where this negative scenario has been included through a combined fragment with the operator neg.

This indicates that all traces in this fragment are “negative” or undesirable. In combination with other traces this negative fragment gives negative traces for every trace leading up to it. The diagram in Fig. 9.9 also defines a set of positive traces that just omit the negative fragment. This also includes the trace where the card is returned directly after the code has been entered. This

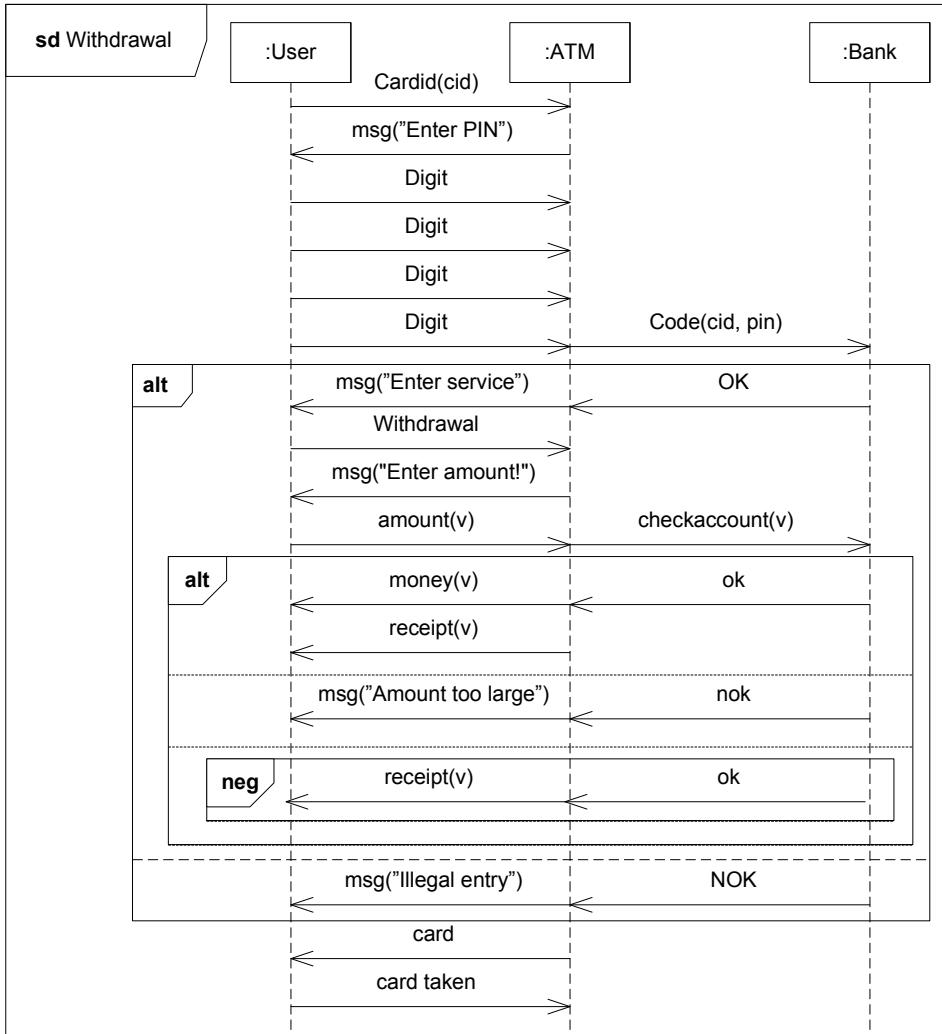


Figure 9.9: Withdrawal with negative traces

trace is included because the neg fragment also introduces the empty but positive trace $\langle \rangle$.

In our example the intuition is simple; any trace that gives no money back when the receipt says it should have, is a negative scenario. The substraces that will follow the negative fragment, the return of the card, certainly does not make the scenario less negative. Still we have not defined all possible scenarios of withdrawing money in an ATM. At this stage it is up to our imagination and the scope of our specification what cases we care to describe. It may or may not be relevant to specify what happens when the customer leaves the ATM without taking the card. Our diagram in Fig. 9.9 leaves that scenario inconclusive.

Summary 2 Semantically, each negative behavior is represented by a trace. Ignoring mandatory behavior that is the issue for the next section, but considering both positive and negative behavior, the semantics of an interaction may be represented by a pair of sets (p, n) where n contains the negative traces and p contains the positive traces. The same trace should not be both positive and negative. Traces that are neither negative nor positive are inconclusive, i.e. considered irrelevant for the specification.

9.3.4 Distinguishing Mandatory from Potential Behavior

Assume that we intend to use our ATM scenario as a requirement specification for purchasing ATMs. The question then becomes whether every ATM needs to be able to perform every positive trace. This would mean that every ATM must be able to offer both euros and US dollars. This would in some places be cumbersome and costly. Thus, this is not an adequate interpretation. On the other hand, we would like to convey that every ATM should offer withdrawal of native money. We specify that this is a mandatory requirement. We need a way to say that it is provisional whether both euros and US dollars are offered, but there is no choice not to offer withdrawal of native money. The latter distinction cannot be expressed directly by the operators of UML 2.0, but we have introduced a small extension and called this choice between alternatives that are mandatory as `xalt`. We have shown our modified specification in Fig. 9.10.

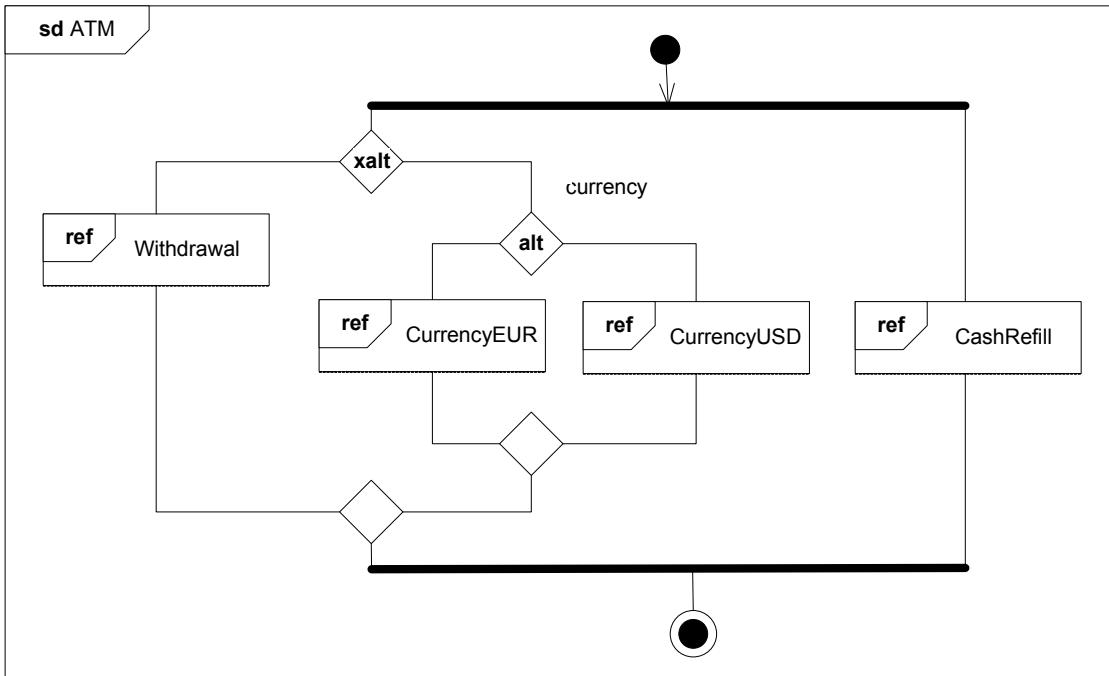


Figure 9.10: Mandatory alternatives (`xalt`)

The semantics of ATM now can be described by an expression on the form:

$$ATM = ((W \text{ xalt } (E \text{ alt } U)) \text{ par } C)$$

In terms of our semantic model this is captured by traces residing in so called *interaction obligations* discriminated by the `xalt` operation. Thus, we have for ATM, two such interaction obligations, one for withdrawal and one for foreign currency. Any correct implementation must support both. The currency obligation may, however, be refined to support only euros or only US dollars.

Summary 3 Semantically, we represent an interaction as a set of interaction obligations $O = \{o_1, \dots, o_n\}$, where $o_j = (p_j, n_j)$, and p_j and n_j are the sets of positive and negative traces,

respectively. An implementation satisfying the specification must fulfill each interaction obligation. Each interaction obligation represents potential variations of mandatory behavior that must be kept separate from other interaction obligations representing variations of other mandatory behavior. The traces within the same interaction obligation serve the same overall purpose.

9.4 STAIRS Spelled out: Supplementing

Supplementing categorizes inconclusive traces as either positive or negative recognizing that early descriptions normally lack completeness. Supplementing supports the incremental process of requirements capture. The initial requirements concentrate on the most obvious normal situations and the most obvious exceptional ones. Supplementing supports this by allowing less obvious situations to be treated later. Hence, in the course of interaction development the overall picture may be filled in with more situations.

In our ATM example specified in Fig. 9.7, we may supplement the services by offering more kinds of foreign currency such as Danish kroner, or Japanese yen. We may likewise offer completely new services such as paying bills.

Furthermore, we may supplement the detailed production traces with more unwanted scenarios like when the user leaves without his card or forgets to key in the right number of digits in his personal identification number. We illustrate this with a Venn-diagram in Fig. 9.11 where the ovals and their subdivisions represents sets of traces. The traces of interest are explicitly named.

Summary 4 Supplementing means reducing the set of inconclusive traces by defining more traces as either positive or negative. Any originally positive trace remains positive, and any originally negative trace remains negative.

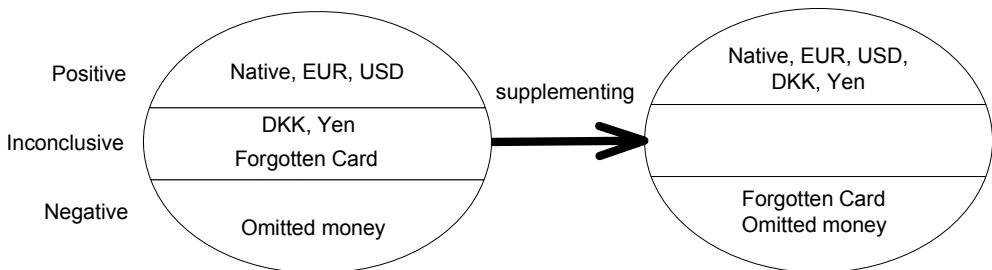


Figure 9.11: Supplementing

9.5 STAIRS Spelled out: Narrowing

When the designers have reached a description that they consider sufficiently complete, they will focus on making the descriptions suitable for implementation. Typically an implementation may decline to produce every positive potential trace. We define narrowing to mean reducing under-specification by eliminating positive traces without really changing the effect of the system.

Narrowing is a relation between descriptions such that the refined description has less variability/under-specification than the former. In our context of interactions, reducing the variability/under-specification means to move traces from the sets of positive traces to the set of negative. A narrowing cannot eliminate traces of the negative trace set since that would mean that some traces specified as illegal would suddenly be acceptable. This would be simply ignoring the specification. In our ATM example specified in Fig. 9.10, narrowing could mean that some ATMs would eliminate a subset of the possible foreign currencies. We illustrate this by a Venn-diagram in Fig. 9.12.

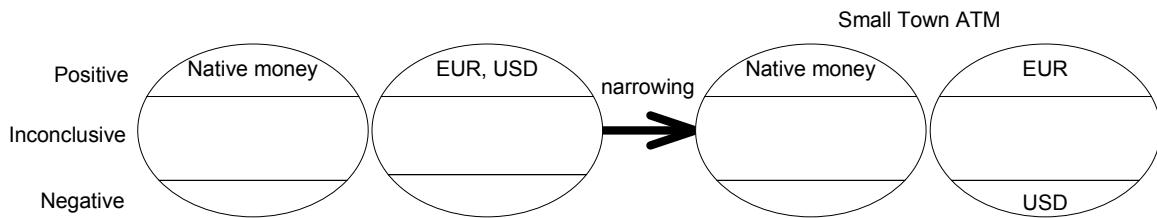


Figure 9.12: Narrowing (Small Town ATM)

Summary 5 *Narrowing means, within one or more interaction obligations, reducing the set of positive traces, and at the same time, moving any trace deleted from the set of positive traces to the set of negative traces. Any inconclusive trace remains inconclusive and any negative trace remains negative.*

9.6 STAIRS Spelled out: Detailing

Detailing involves introducing a more detailed description without significantly altering the externally observable behavior. In Fig. 9.13 we have chosen to decompose the ATM as it actually consists of a number of components.

The resulting activity of the diagram in Fig. 9.13 is that of the :ATM lifeline of Fig. 9.8. This is done through the decomposition mechanism of UML 2.0. In addition we have detailed the outcome result by specifying that the money is delivered in distinct notes. We show the simple message translation in the separate diagram in Fig. 9.14. Clearly, the external behavior of ATM_Withdrawal is the same as the external behavior of :ATM in Fig. 9.8 given the Notes_Translator translation on the outcome. This shows how STAIRS supports the decomposition of the money message in Withdrawal into the different notes values messages in ATM_Withdrawal. The Notes_Translator diagram documents this decomposition and is in contrast to ATM_Withdrawal not supposed to be implemented.

Summary 6 *Detailing means that the sets of positive, negative and inconclusive traces are refined with respect to a translation between the more detailed and the given interaction.*

9.7 Formal Foundation

Here, we demonstrate how the concepts in this paper may be formalized. For more details, we refer to [HRS04].

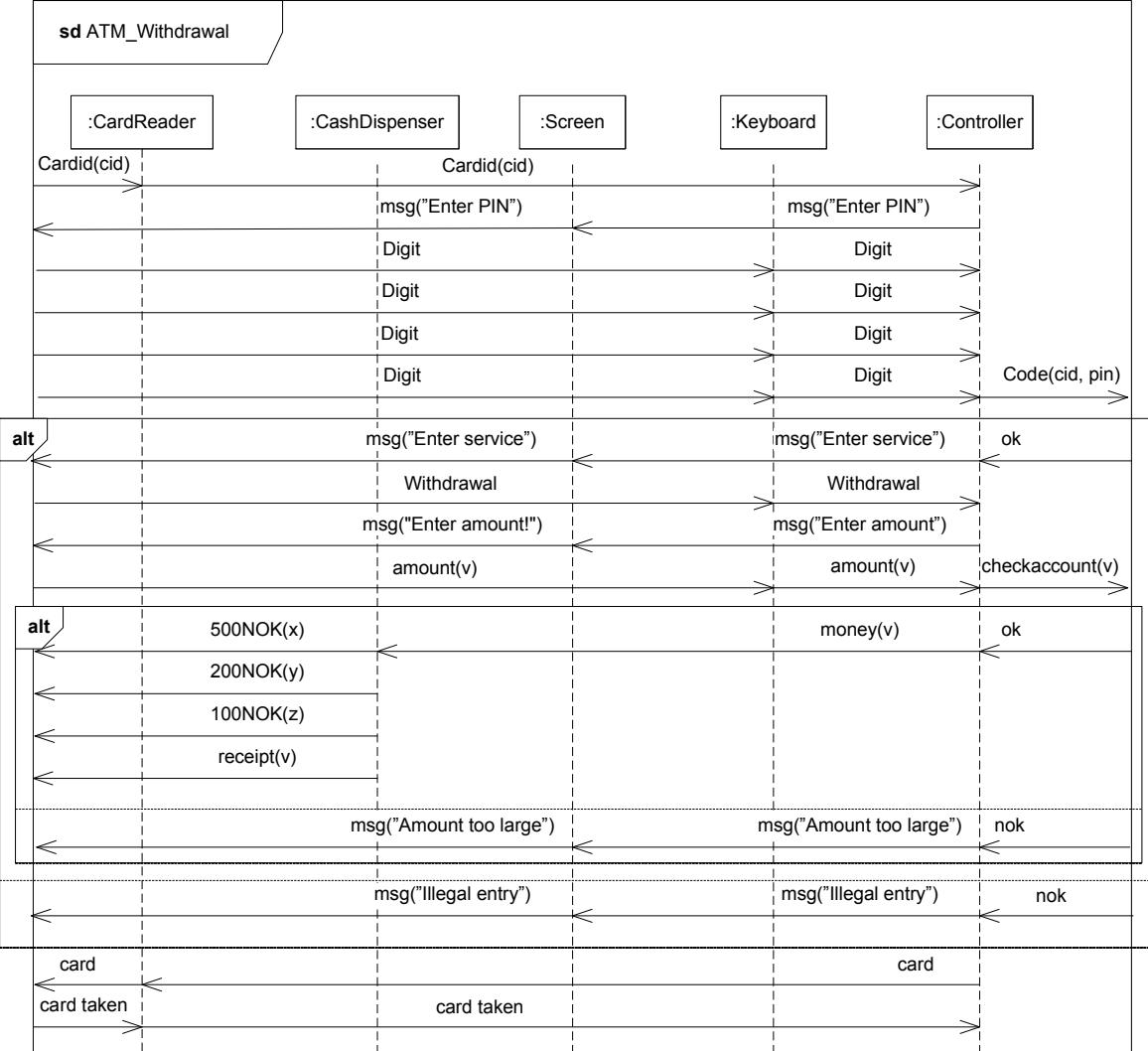


Figure 9.13: Detailing the ATM

9.7.1 Representing Runs by Traces

As explained in section 9.3, a trace is a sequence of events, used to represent a system run. In each trace, a send event (tagged by an !) should always be ordered before the corresponding receive event (tagged by ?). We let H denote the set of all traces that complies with this requirement.

A message is a triple (s, tr, re) of a signal s , a transmitter tr , and a receiver re . M denotes the set of all messages. The transmitters and receivers are lifelines. L denotes the set of all lifelines. An event is a pair of kind and message

$$(k, m) \in \{!, ?\} \times M$$

E denotes the set of all events. We define the functions

$$k. _ \in E \rightarrow \{!, ?\}, \quad tr. _, re. _ \in E \rightarrow L$$

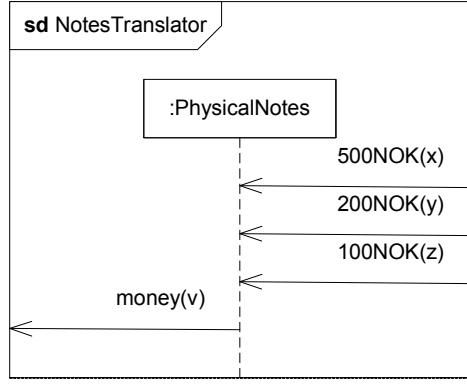


Figure 9.14: NotesTranslator

to yield the kind, transmitter and receiver of an event, respectively.

For concatenation of sequences, filtering of sequences, and filtering of pairs of sequences, we have the functions \cap , \circledcirc , and \circledast , respectively.

Concatenating two sequences implies gluing them together. Hence, $a_1 \cap a_2$ denotes a sequence that equals a_1 if a_1 is infinite. Otherwise, $a_1 \cap a_2$ denotes a sequence that is prefixed by a_1 and suffixed by a_2 . In both cases, the length of $a_1 \cap a_2$ is equal to the sum of the lengths of a_1 and a_2 .

The filtering function \circledcirc is used to filter away elements. By $B \circledcirc a$ we denote the sequence obtained from the sequence a by removing all elements in a that are not in the set of elements B . For example, we have that

$$\{1, 3\} \circledcirc \langle 1, 1, 2, 1, 3, 2 \rangle = \langle 1, 1, 1, 3 \rangle$$

The filtering function \circledast may be understood as a generalization of \circledcirc . The function \circledast filters pairs of sequences with respect to pairs of elements in the same way as \circledcirc filters sequences with respect to elements. For any set of pairs of elements P and pair of sequences t , by $P \circledast t$ we denote the pair of sequences obtained from t by

- truncating the longest sequence in t at the length of the shortest sequence in t if the two sequences are of unequal length;
- for each $j \in [1 \dots k]$, where k is the length of the shortest sequence in t , selecting or deleting the two elements at index j in the two sequences, depending on whether the pair of these elements is in the set P .

For example, we have that

$$\{(1, f), (1, g)\} \circledast (\langle 1, 1, 2, 1, 2 \rangle, \langle f, f, f, g, g \rangle) = (\langle 1, 1, 1 \rangle, \langle f, f, g \rangle)$$

9.7.2 Semantics of Sequence Diagrams

The semantics of sequence diagrams is defined by a function $\llbracket \] \rrbracket$ that for any sequence diagram d yields a set $\llbracket d \rrbracket$ of interaction obligations. As explained in section 9.3.4, an interaction obligation

is a pair (p, n) of sets of traces where the first set is interpreted as the set of positive traces and the second set is the set of negative traces. The term obligation is used to explicitly convey that any implementation of a specification is obliged to fulfill each specified alternative.

For a sequence diagram consisting of a single event e , its semantics is given by:

$$\llbracket e \rrbracket \stackrel{\text{def}}{=} \{(\{\langle e \rangle\}, \emptyset)\}$$

More complex sequence diagrams are constructed through the application of various operators. We focus on the operators that we find most essential, namely negation (neg), potential choice (alt), mandatory choice (xalt), parallel execution (par), and weak sequencing (seq).

As can be expected, we have associativity of alt, xalt, par and seq. We also have commutativity of alt, xalt and par. Proofs can be found in [HHR04].

Negation

The neg construct defines negative traces:

$$\llbracket \text{neg } d \rrbracket \stackrel{\text{def}}{=} \{(\{\langle \rangle\}, p \cup n) \mid (p, n) \in \llbracket d \rrbracket\}$$

Notice that a negative trace cannot be made positive by reapplying neg. Negative traces remain negative. Negation is an operation that characterizes traces absolutely and not relatively. The intuition is that the focus of the neg construct is on characterizing the positive traces in the operand as negative. Negative traces will always propagate as negative to the outermost level. The neg construct defines the empty trace as positive. This facilitates the embedding of negs in sequence diagrams also specifying positive behavior.

Potential Choice

The alt construct defines potential traces. The semantics is the union of the trace sets for both positive and negative:

$$\llbracket d_1 \text{ alt } d_2 \rrbracket \stackrel{\text{def}}{=} \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in \llbracket d_1 \rrbracket \wedge (p_2, n_2) \in \llbracket d_2 \rrbracket\}$$

Mandatory Choice

The xalt construct defines mandatory choices. All implementations must be able to handle every interaction obligation.

$$\llbracket d_1 \text{ xalt } d_2 \rrbracket \stackrel{\text{def}}{=} \llbracket d_1 \rrbracket \cup \llbracket d_2 \rrbracket$$

Parallel Execution

The par construct represents a parallel merge. In order to define par, we first define parallel execution on trace sets:

$$s_1 \parallel s_2 \stackrel{\text{def}}{=} \{h \in H \mid \exists p \in \{1, 2\}^\infty : \begin{aligned} & \pi_2((\{1\} \times E) \circledast (p, h)) \in s_1 \wedge \\ & \pi_2((\{2\} \times E) \circledast (p, h)) \in s_2\} \end{aligned}$$

In this definition, we make use of an oracle, the infinite sequence p , to resolve the non-determinism in the interleaving. It determines the order in which events from traces in s_1 and s_2 are sequenced. π_2 is a projection operator returning the second element of a pair.

The **par** construct itself is defined as:

$$\llbracket d_1 \mathbf{par} d_2 \rrbracket \stackrel{\text{def}}{=} \{o_1 \parallel o_2 \mid o_1 \in \llbracket d_1 \rrbracket \wedge o_2 \in \llbracket d_2 \rrbracket\}$$

where parallel execution of interaction obligations is defined as:

$$(p_1, n_1) \parallel (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \parallel p_2, (n_1 \parallel p_2) \cup (n_1 \parallel n_2) \cup (p_1 \parallel n_2))$$

Note how any trace involving a negative trace will remain negative in the resulting interaction obligation.

Weak Sequencing

Weak sequencing is the implicit composition mechanism combining constructs of a sequence diagram. First, we define weak sequencing of trace sets:

$$s_1 \succsim s_2 \stackrel{\text{def}}{=} \{h \in s_1 \parallel s_2 \mid \exists h_1 \in s_1, h_2 \in s_2 : \forall l \in L : e.l \circledcirc h = e.l \circledcirc h_1 \cap e.l \circledcirc h_2\}$$

where $e.l$ denotes the set of events that may take place on the lifeline l . Note that weak sequencing degenerates to parallel execution if the operands have disjoint sets of lifelines.

The **seq** construct itself is defined as:

$$\llbracket d_1 \mathbf{seq} d_2 \rrbracket \stackrel{\text{def}}{=} \{o_1 \succsim o_2 \mid o_1 \in \llbracket d_1 \rrbracket \wedge o_2 \in \llbracket d_2 \rrbracket\}$$

where weak sequencing of interaction obligations is defined as:

$$(p_1, n_1) \succsim (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \succsim p_2, (n_1 \succsim p_2) \cup (n_1 \succsim n_2) \cup (p_1 \succsim n_2))$$

Note how anything involving a negative trace remains negative.

9.7.3 Refinement

Refinement means to add information to a specification such that the specification becomes closer to an implementation. Supplementing and narrowing are special cases of this general notion. Detailing is defined in terms of lifeline decomposition and interface refinement that are both defined in terms of the basic notion of refinement.

An interaction obligation (p_2, n_2) is a refinement of an interaction obligation (p_1, n_1) , written $(p_1, n_1) \rightsquigarrow (p_2, n_2)$, i

$$n_1 \subseteq n_2 \quad \wedge \quad p_1 \subseteq p_2 \cup n_2$$

A sequence diagram d' is a refinement of a sequence diagram d , written $d \rightsquigarrow d'$, i

$$\forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow o'$$

The refinement semantics supports the classical notions of compositional refinement providing a firm foundation for compositional analysis, verification and testing. In [HHR04] we prove that refinement as defined above is transitive as well as monotonic with respect to the operators defined in section 9.7.2.

Supplementing

Supplementing categorizes inconclusive behavior as either positive or negative. An interaction obligation (p_2, n_2) supplements an interaction obligation (p_1, n_1) , written $(p_1, n_1) \rightsquigarrow_s (p_2, n_2)$, i

$$(n_1 \subset n_2 \quad \wedge \quad p_1 \subseteq p_2) \quad \vee \quad (n_1 \subseteq n_2 \quad \wedge \quad p_1 \subset p_2)$$

Narrowing

Narrowing reduces the allowed (positive) behavior to match the problem better. An interaction obligation (p_2, n_2) narrows an interaction obligation (p_1, n_1) , written $(p_1, n_1) \rightsquigarrow_n (p_2, n_2)$, i

$$p_2 \subset p_1 \quad \wedge \quad n_2 = n_1 \cup (p_1 \setminus p_2)$$

Black-Box Refinement

Black-box refinement may be understood as refinement restricted to the externally visible behavior. We define the function

$$\text{ext} \in H \times \mathbb{P}(L) \rightarrow H$$

to yield the trace obtained from the trace given as first argument by filtering away those events that are internal with respect to the set of lifelines given as second argument, i.e.:

$$\text{ext}(h, l) \stackrel{\text{def}}{=} \{e \in E \mid (k.e = ? \wedge \text{tr}.e \notin l) \vee (k.e = ! \wedge \text{re}.e \notin l)\} \circledcirc h$$

The ext operator is overloaded to sets of traces and pairs of sets of traces in the standard pointwise manner, e.g.:

$$\text{ext}(s, l) \stackrel{\text{def}}{=} \{\text{ext}(h, l) \mid h \in s\}$$

A sequence diagram d' is a black-box refinement of a sequence diagram d , written $d \rightsquigarrow_b d'$, i

$$\forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : \text{ext}(o, \text{ll}(d)) \rightsquigarrow \text{ext}(o', \text{ll}(d'))$$

where the function ll yields the set of lifelines of a sequence diagram.

Detailing

When we increase the granularity of sequence diagrams we call this a detailing of the specification. The granularity can be altered in two different ways: either by decomposing the lifelines such that their inner parts and their internal behavior are displayed in the diagram or by changing the data-structure of messages such that they convey more detailed information.

Black-box refinement is sufficiently general to formalize lifeline decompositions that are not externally visible. However, many lifeline decompositions are externally visible. As an example of a lifeline decomposition that is externally visible, consider the decomposition of the ATM in Fig. 9.13. The messages that originally (in Fig. 9.8) had :ATM as sender/receiver, now have the different components of the ATM (such as :CardReader or :Screen) as sender/receiver.

To allow for this, we extend the definition of black-box refinement with the notion of a lifeline substitution. The resulting refinement relation is called lifeline decomposition. A lifeline substitution is a partial function of type $L \rightarrow L$. LS denotes the set of all such substitutions. We define the function

$$subst \in D \times LS \rightarrow D$$

such that $subst(d, ls)$ yields the sequence diagram obtained from d by substituting every lifeline l in d for which ls is defined with the lifeline $ls(l)$.

We then define that a sequence diagram d' is a lifeline decomposition of a sequence diagram d with respect to a lifeline substitution ls , written $d \rightsquigarrow_l^{ls} d'$, i

$$d \rightsquigarrow_b subst(d', ls)$$

Changing the data-structure of messages may be understood as black-box refinement modulo a translation of the externally visible behavior. This translation is specified by a sequence diagram t , and we refer to this as an interface refinement.

We define that a sequence diagram d' is an interface refinement of a sequence diagram d with respect to a sequence diagram t , written $d \rightsquigarrow_i^t d'$, i

$$d \rightsquigarrow_b (t \text{ seq } d')$$

Detailing may then be defined as the transitive and reflexive closure of lifeline decomposition and interface refinement.

9.8 Conclusions

We have presented STAIRS, a formal approach to the step-wise, incremental development of interactions. It is based on trace semantics. Traces are sequences of events. Events are representations of sending and receiving messages. STAIRS meets the requirements of Section 9.2 in the following sense:

1. Different potential behaviors are expressed through lifeline independence and by alt combined fragments. Semantically, each potential behavior is represented by a trace.
2. Different mandatory behaviors are expressed using combined fragments with xalt. Semantically, different mandatory behaviors are separated by placing them in the different interaction obligations.

3. The potential and the mandatory behavior constitute the positive behavior. Negative behavior may be specified by the neg-construct. Also negative behavior is represented semantically by a set of traces in every interaction obligation.
4. The classical notion of refinement is supported. Firstly, under-specification in the form of potential behavior may be reduced (narrowing). This corresponds to refinement by strengthening the post-condition in traditional pre/post specification [Jon86]. Secondly, the scope of the specification may be enlarged (supplementing). This corresponds to refinement by weakening the pre-condition in traditional pre/post specification [Jon86]. Thirdly, the granularity and data-structure of messages may be altered (detailling). This corresponds to classical data-refinement [Hoa72], or more exactly, to the more recent form of interface refinement as e.g. in TLA [AL95] and Focus [BS01].
5. Incremental development of interactions in the form of supplementing, narrowing and detailing has been formalized.
6. The underlying semantics supports the classical notations of compositional refinement providing a firm foundation for compositional analysis, verification and testing. In [HRRS04] we show that the basic notions of supplementing and narrowing are reflexive, transitive and monotonic with respect to the operators specified in Section 9.7. The same holds for detailing modulo the specified translation.

9.8.1 Related Work

To consider not only positive traces, but also negative ones, has been suggested before. In [Hau95] the proposed methodology stated that specifying negative scenarios could be even more practical and powerful than only specifying the possible or mandatory ones. It was made clear that the MSC-92 standard [ITU93] was not sufficient to express the intention behind the scenarios and that the MSC documents had to be supplemented with informal statements about the intended interpretation of the set of traces expressed by the different MSCs.

The algebraic semantics of MSC-92 [ITU94] gave rise to a canonical logical expression restricted to the strict sequencing operator and a choice operator. When the MSC standard evolved with more advanced structuring mechanisms, the formal semantics as given in [ITU98] and [Ren98] was based on sets of traces, but it was still expressed in algebraic terms. The MSC approach to sequence diagram semantics is an interleaving semantics based on a fully compositional paradigm. The set of traces denoting the semantics of a message sequence chart can be calculated from its constituent parts based on definitions of the semantics of the structuring concepts as operators. This is very much the approach that we base our semantics on as we calculate our semantics of an interaction fragment from the semantics of its internal fragments. The notion of negative traces, and the explicit distinction between mandatory and potential behavior is beyond the MSC language and its semantics. The Eindhoven school of MSC researchers led by Sjouke Mauw concentrated mainly on establishing the formal properties of the logical systems used for defining the semantics, and also how this could be applied to make tools.

The need for describing also the intention behind the scenarios motivated the so-called “two-layer” approaches. In [CPRO95] they showed how MSC could be combined with languages for temporal logics such as CTL letting the scenarios constitute the atoms for the higher level of

modal descriptions. With this one could describe that certain scenarios should appear or should never appear.

Damm and Harel brought this further through their augmented MSC language LSC (Live Sequence Charts) [DH99]. This may also be characterized as a two-layer approach as it takes the basic message sequence charts as starting point and add modal characteristics upon those. The modal expressiveness is strong in LSC since charts, locations, messages and conditions are orthogonally characterized as either mandatory or provisional. Since LSC also includes a notion of subchart, the combinatory complexity can be quite high. The “inline expressions” of MSC-96 (corresponding to combined fragments in UML 2.0) and MSC documents as in MSC-2000 [Hau01] (corresponds to classifier in UML 2.0) are, however, not included in LSC. Mandatory charts are called universal. Their interpretation is that provided their initial condition holds, these charts must happen. Mandatory as in LSC should not be confused with mandatory as in STAIRS, since the latter only specifies traces that must be present in an implementation while the first specifies all allowed traces. Hence, mandatory as in STAIRS does not distinguish between universal or existential interpretation, but rather gives a restriction on what behaviors that must be kept during a refinement. Provisional charts are called existential and they may happen if their initial condition holds. Through mandatory charts it is of course indirectly also possible to define scenarios that are forbidden or negative. Their semantics is said to be a conservative extension of the original MSC semantics, but their construction of the semantics is based on a two-stage procedure. The first stage defines a symbolic transition system from an LSC and from that a set of runs accepted by the LSC is produced. These runs represent traces where each basic element is a snapshot of a corresponding system.

The motivation behind LSC is explicitly to relate sequence diagrams to other system descriptions, typically defined with state machines. Harel has also been involved in the development of a tool-supported methodology that uses LSC as a way to prescribe systems as well as verifying the correspondence between manually described LSCs and State Machines [HM03].

Our approach is similar to LSC since it is basically interleaving. STAIRS is essentially one-stage as the modal distinction between the positive and negative traces in principle is present in every fragment. The final modality results directly from the semantic compositions. With respect to language, we consider almost only what is UML 2.0, while LSC is a language extension of its own. LSC could in the future become a particular UML profile. Furthermore, our focus is on refinement of sequence diagrams as a means for system development and system validation. This means that in our approach the distinction between mandatory and provisional is captured through interaction obligations.

The work by Krüger [Krü00] addresses similar concerns as the ones introduced in this article and covered by the LSC-approach of Harel. Just as with LSC MSCs can be given interpretations as existential or universal. The exact and negative interpretations are also introduced. Krüger also proposes notions of refinement for MSCs. Binding references, interface refinement, property refinement and structural refinement are refinement relations between MSCs at different level of abstraction. Narrowing as described in STAIRS corresponds closely to property refinement in [Krü00] and detailing corresponds to interface refinement and structural refinement. However, Krüger does not distinguish between intended non-determinism and non-determinism as a result of under-specification in the refinement relations.

Although this paper presents STAIRS in the setting of UML 2.0 sequence diagrams, the

underlying principles apply just as well to MSC given that the MSC language is extended with an `xalt` construct similar to the one proposed above for UML 2.0. STAIRS may also be adapted to support LSC. STAIRS is complementary to software development processes based on use-cases, and classical object-oriented approaches such as the Unified Process [JBR99]. STAIRS provides formal foundation for the basic incremental steps of such processes.

Acknowledgements. The research on which this paper reports has partly been carried out within the context of the IKT-2010 project SARDAS (15295/431) funded by the Research Council of Norway. We thank Mass Soldal Lund, Fredrik Seehusen and Ina Schieferdecker for helpful feedback.

References

- [AL95] M. Abadi and L. Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17:507–533, 1995.
- [BS01] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.
- [CPRO95] P. Combes, S. Pickin, B. Renard, and F. Olsen. MSCs to express service requirements as properties on an SDL model: Application to service interaction detection. In *7th SDL Forum (SDL'95)*, pages 243–256. North-Holland, 1995.
- [DH99] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. In *Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, pages 293–311. Kluwer, 1999.
- [dR85] W-P. de Roever. The quest for compositionality: A survey of assertion-based proof systems for concurrent programs: Part 1. In *Formal Models in Programming*, pages 181–205. North-Holland, 1985.
- [Hau95] Ø. Haugen. Using MSC-92 effectively. In *7th SDL Forum (SDL'95)*, pages 37–49. North-Holland, 1995.
- [Hau01] Ø. Haugen. MSC-2000 interaction diagrams for the new millennium. *Computer Networks*, 35:721–732, 2001.
- [HHR04] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309, Department of Informatics, University of Oslo, 2004.
- [HM03] D. Harel and R. Marelly. Specifying and executing behavioral requirements: The play-in/play-out approach. *Software and System Modeling*, 2:82–107, 2003.
- [HMPW03] Ø. Haugen, B. Møller-Pedersen, and T. Weigert. Structural modeling with UML 2.0. In *UML for Real*, pages 53–76. Kluwer, 2003.
- [Hoa72] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–282, 1972.
- [HS03] Ø. Haugen and K. Stølen. STAIRS — Steps to analyze interactions with refinement semantics. In *Sixth International Conference on UML (UML2003)*, number 2863 in Lecture Notes in Computer Science, pages 388–402. Springer, 2003.
- [ITU93] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1993.
- [ITU94] International Telecommunication Union. *Recommendation Z.120 Annex B: Algebraic Semantics of Message Sequence Charts*, 1994.

- [ITU98] International Telecommunication Union. *Recommendation Z.120 Annex B: Formal Semantics of Message Sequence Charts*, 1998.
- [JBR99] I. Jacobson, G. Booch, and J Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.
- [Jon72] C. B. Jones. Formal development of correct algorithms: An example based on Earley's recogniser. In *ACM Conference on Proving Assertions about Programs*, number 7 in SIGPLAN Notices, pages 150–169, 1972.
- [Jon81] C. B. Jones. *Development Methods for Computer Programs Including a Notion of Interference*. PhD thesis, Oxford University, 1981.
- [Jon86] C. B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall, 1986.
- [Krü00] I. Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.
- [Mil71] R. Milner. An algebraic definition of simulation between programs. In *International Joint Conference on Artificial Intelligence*, pages 481–489. Kaufmann, 1971.
- [OMG04] Object Management Group. *UML 2.0 Superstructure Specification*, document: ptc/04-10-02 edition, 2004.
- [Ren98] M. A. Reniers. *Message Sequence Chart: Syntax and Semantics*. PhD thesis, Eindhoven University of Technology, 1998.

Chapter 10

Why Timed Sequence Diagrams Require Three-Event Semantics

Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen

Publication.

Published as Technical Report 309, Department of Informatics, University of Oslo, 2006. Extended and revised version of: Why timed sequence diagrams require three-event semantics. In *Scenarios: Models, Transformations and Tools*, volume 3466 of *LNCS*, pages 1–25. Springer, 2005.

Abstract.

STAIRS is an approach to the compositional development of sequence diagrams supporting the specification of mandatory as well as potential behavior. In order to express the necessary distinction between black-box and glass-box refinement, an extension of the semantic framework with three event messages is introduced. A concrete syntax is also proposed. The proposed extension is especially useful when describing time constraints. The resulting approach, referred to as Timed STAIRS, is formally underpinned by denotational trace semantics. A trace is a sequence built from three kinds of events: events for transmission, reception and consumption. We argue that such traces give the necessary expressiveness to capture the standard UML interpretation of sequence diagrams as well as the black-box interpretation found in classical formal methods.

10.1 Introduction to STAIRS

Sequence diagrams have been used informally for several decades. The first standardization of sequence diagrams came in 1992 [ITU93] – often referred to as MSC-92. Later we have seen several dialects and variations. The sequence diagrams of UML 1.4 [OMG00] were comparable to those of MSC-92, while the recent UML 2.0 [OMG04] has upgraded sequence diagrams to conform well to MSC-2000 [ITU99].

Sequence diagrams show how messages are sent between objects or other instances to perform a task. They are used in a number of different situations. They are for example used by an individual designer to get a better grip of a communication scenario or by a group to achieve a common understanding of the situation. Sequence diagrams are also used during the more detailed design phase where the precise inter-process communication must be set up according to formal protocols. When testing is performed, the behavior of the system can be described as sequence diagrams and compared with those of the earlier phases.

Sequence diagrams seem to have the ability to be understood and produced by professionals of computer systems design as well as potential end-users and stakeholders of the (future) systems. Even though sequence diagrams are intuitive – a property which is always exploited – which diagrams to make is not always so intuitive. It is also the case that intuition is not always the best guide for a precise interpretation of a complicated scenario. Therefore we have brought forth an approach for reaching a sensible and fruitful set of sequence diagrams, supported by formal reasoning. We called this approach STAIRS – Steps To Analyze Interactions with Refinement Semantics [HS03].

STAIRS distinguishes between positive and negative traces and accepts that some traces may be inconclusive meaning that they have not yet or should not be characterized as positive or negative. STAIRS views the process of developing the interactions as a process of learning through describing. From a fuzzy, rough sketch, the aim is to reach a precise and detailed description applicable for formal handling. To come from the rough and fuzzy to the precise and detailed, STAIRS distinguishes between three sub-activities: (1) supplementing, (2) narrowing and (3) detailing.

Supplementing categorizes inconclusive behavior as either positive or negative. The initial requirements concentrate on the most obvious normal situations and the most obvious exceptional ones. Supplementing supports this by allowing less obvious situations to be treated later. Narrowing reduces the allowed behavior to match the problem better. Detailing involves introducing a more detailed description without significantly altering the externally observable behavior.

STAIRS distinguishes between potential alternatives and mandatory or obligatory alternatives. A special composition operator named `xalt` facilitates the specification of mandatory alternatives.

Figure 10.1 shows our STAIRS example – an interaction overview diagram description of the making of a dinner at an ethnic restaurant.

The dinner starts with a salad and continues with a main course that consists of an entree and a side order, which are made in parallel. For the side order there is a simple choice between three alternatives and the restaurant is not obliged to have any particular of them available. Supplementing side orders could be to offer soya beans in addition, while narrowing would mean that the restaurant could choose only to serve rice and never potatoes nor fries. It would still be consistent with the specification and a valid refinement. On the other hand, the entree

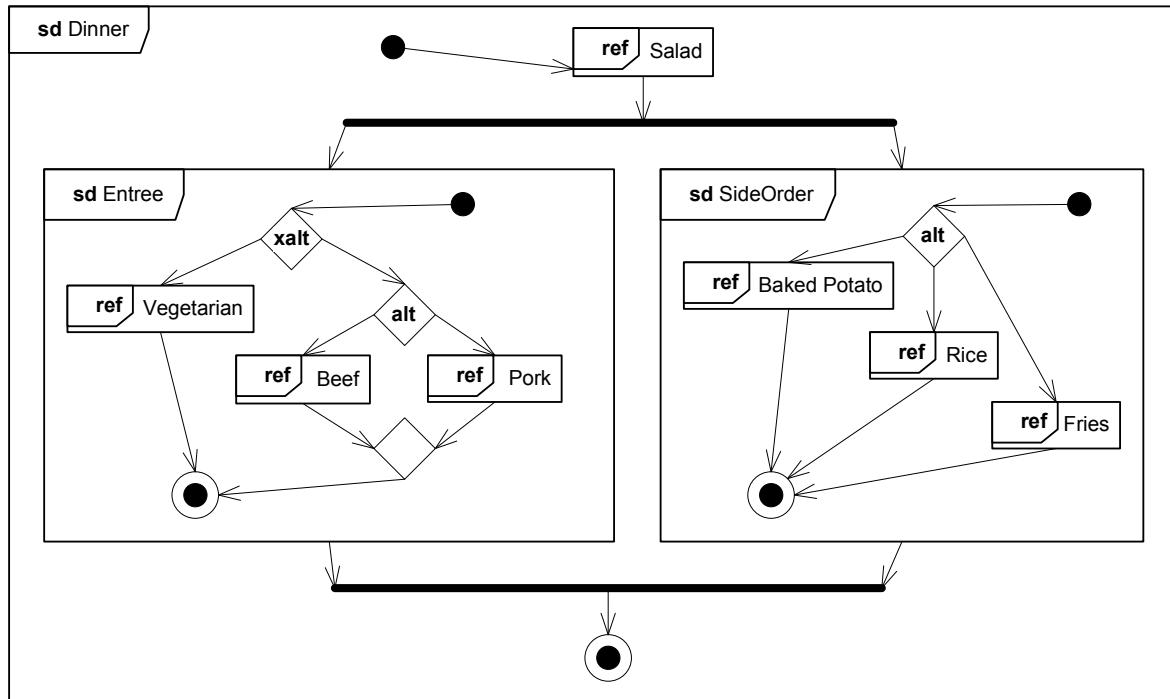


Figure 10.1: Interaction overview diagram of a dinner

has more absolute requirements. The restaurant is obliged to offer vegetarian as well as meat, but it does not have to serve both beef and pork. This means that Indian as well as Jewish restaurants are refinements (narrowing) of our dinner concept, while a pure vegetarian restaurant is not valid according to our specification.

The remainder of the paper is divided into six sections: Section 10.2 motivates the need for a three event semantics for sequence diagrams. Section 10.3 introduces the formal machinery; in particular, it defines the syntax and semantics of sequence diagrams. Section 10.4 defines two special interpretations of sequence diagrams, referred to as the standard and the black-box interpretation, respectively. Section 10.5 demonstrates the full power of Timed STAIRS as specification formalism. Section 10.6 introduces glass-box and black-box refinement and demonstrates the use of these notions. Section 10.7 provides a brief conclusion and compares Timed STAIRS to other approaches known from the literature.

10.2 Motivating Timed STAIRS

STAIRS works well for its purpose. However, there are certain aspects that cannot be expressed within the framework as presented in [HS03]. For instance time constraints and the difference between glass-box and black-box view of a system. This section motivates the need for this extra expressiveness.

Let us now look closer at the details of making the Beef in Figure 10.1. From Figure 10.2¹ it is

¹This sequence diagram is not a complete specification of Beef. The supplementing has not yet been finished.

intuitive to assume that the working of the Cook making Beef can be explained by the following scheme: The Cook receives an order for main dish (of type Beef) and then turns on the heat and waits until the heat is adequate. Then he fetches the sirloin meat from the refrigerator before putting it on the grill. Then he fetches the sirloin from the stove (hopefully when it is adequately grilled). He then sends the steak to the customer.

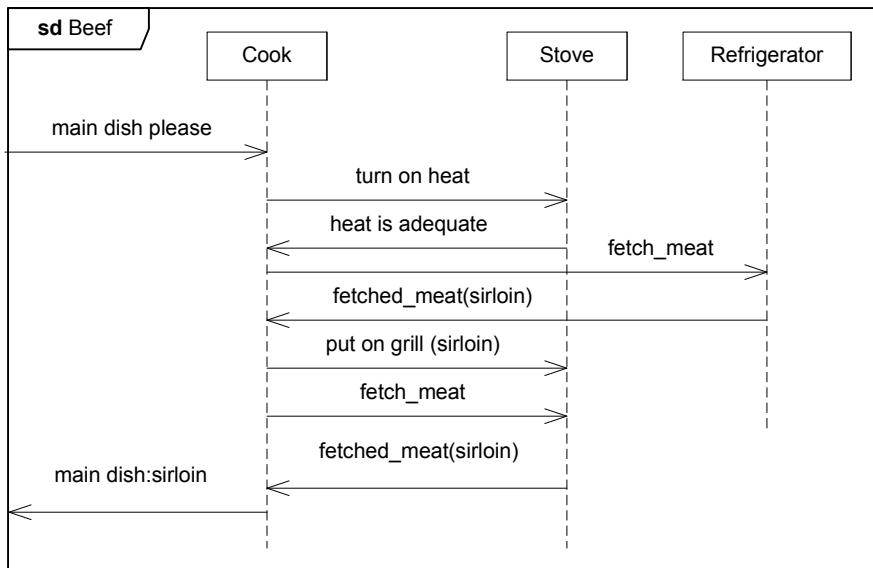


Figure 10.2: Sequence diagram of Beef

We reached this explanation of the procedures of the cook from looking locally at the cook's lifeline in the Beef diagram. The input event led to one or more outputs, before he again would wait for an input. We found it natural to assume that the input event meant that the cook handled this event, consumed it and acted upon it. This intuition gives rise to what we here will call the standard interpretation of sequence diagrams where an input event is seen as consumption of the event, and where the directly following output events of the trace are causally linked to the consumption. Thus, we can by considering each separate lifeline locally determine the transitions of a state machine describing the lifeline.

Our description of how the beef is made comes from a quiet day, or early in the evening when there were not so many customers and the kitchen had no problems to take care of each order immediately. Furthermore our description was probably made for one of those restaurants where the customers can look into the kitchen through glass. It was a glass-box description. We want, however, to be able to describe the situation later in the evening when the restaurant is crammed with customers and in a restaurant where there is only a black door to the kitchen. We would like to assume that even though the restaurant is full, the kitchen will handle our order immediately, but alas this is of course not the case. We can only observe the kitchen as a black-box. We observe the waiters coming through the door as messengers – orders one way

From a methodological point of view, the diagram should be “closed” with an assert when the supplementing has been finished. This to state that what is still left as inconclusive behavior should from now on be understood as negative. Otherwise, we do not get the semantics intended by Figure 10.1.

and dishes the other. From these observations we could make estimates of the efficiency of the kitchen. Notice that the efficiency of the kitchen cannot be derived from when the customers placed the orders because the waiters may stop at several tables before they enter the kitchen. Comparing black-box observations of the kitchen with our glass-box one, we realize that in the glass-box description no event was attached to passing through the door. The order was sent by the customer and consumed by the chef. The passing through the door represents that the kitchen is receiving the message but not necessarily doing something with it. As long as you are not interested in timing matters, the difference is seldom practically significant, but when time matters, the difference between when a message is received and when it is consumed is crucial. How is the kitchen organized to handle the orders in a swift and fair manner?

Motivated by this we will use three events to represent the communication of a message: the sending event, the receiving event and the consumption event, and each of these events may have a timestamp associated. We will introduce concrete syntax for sequence diagrams to capture this and the distinction is also reflected in the semantics. This will give us sufficient expressiveness to describe a black-box interpretation as well as the standard glass-box interpretation.

10.3 Formal Foundation

In the following we define the notion of sequence diagram. In particular, we formalize the meaning of sequence diagrams in denotational trace semantics.

10.3.1 Mathematical Background on Sequences

We will define the semantics of sequence diagrams by using sequences of events.

\mathbb{N} denotes the set of natural numbers, while \mathbb{N}_0 denotes the set of natural numbers including 0.

By A^∞ and A^ω we denote the set of all infinite sequences and the set of all finite and infinite sequences over the set A , respectively. We define the functions

$$\#_a : A^\omega \rightarrow \mathbb{N}_0 \cup \{\infty\}, \quad [a]_n : A^\omega \times \mathbb{N} \rightarrow A, \quad a \sqsubseteq b : A^\omega \times A^\omega \rightarrow \text{Bool}$$

to yield the length, the n th element of a sequence, and the prefix ordering on sequences. Hence, $\#a$ yields the number of elements in a , $a[n]$ yields a 's n th element if $n \leq \#a$, and $a_1 \sqsubseteq a_2$ evaluates to true if a_1 is an initial segment of a_2 , or if $a_1 = a_2$.

We also need functions for concatenation, truncation and filtering:

$$\begin{aligned} a \cdot b &: A^\omega \times A^\omega \rightarrow A^\omega, & [a]_n &: A^\omega \times \mathbb{N}_0 \rightarrow A^\omega, \\ a \circ b &: \mathbb{P}(A) \times A^\omega \rightarrow A^\omega, & [a \circ b]_n &: \mathbb{P}(A \times B) \times (A^\omega \times B^\omega) \rightarrow A^\omega \times B^\omega \end{aligned}$$

Concatenating two sequences implies gluing them together. Hence, $a_1 \cdot a_2$ denotes a sequence of length $\#a_1 + \#a_2$ that equals a_1 if a_1 is infinite, and is prefixed by a_1 and suffixed by a_2 , otherwise. For any $0 \leq i \leq \#a$, we define $a|_i$ to denote the prefix of a of length i .

The filtering function \circ is used to filter away elements. By $B \circ a$ we denote the sequence obtained from the sequence a by removing all elements in a that are not in the set of elements B . For example, we have that

$$\{1, 3\} \circ \langle 1, 1, 2, 1, 3, 2 \rangle = \langle 1, 1, 1, 3 \rangle$$

The filtering function \circledcirc may be understood as a generalization of \circledast . The function \circledcirc filters pairs of sequences with respect to pairs of elements in the same way as \circledast filters sequences with respect to elements. For any set of pairs of elements P and pair of sequences t , by $P \circledcirc t$ we denote the pair of sequences obtained from t by

- truncating the longest sequence in t at the length of the shortest sequence in t if the two sequences are of unequal length;
- for each $j \in [1 \dots k]$, where k is the length of the shortest sequence in t , selecting or deleting the two elements at index j in the two sequences, depending on whether the pair of these elements is in the set P .

For example, we have that

$$\{(1, f), (1, g)\} \circledcirc (\langle 1, 1, 2, 1, 2 \rangle, \langle f, f, f, g, g \rangle) = (\langle 1, 1, 1 \rangle, \langle f, f, g \rangle)$$

For a formal definition of \circledcirc , see [BS01].

10.3.2 Syntax of Sequence Diagrams

A message is a triple (s, tr, re) of a signal s , a transmitter tr , and a receiver re . \mathcal{M} denotes the set of all messages. The transmitters and receivers are lifelines. \mathcal{L} denotes the set of all lifelines.

We distinguish between three kinds of events; a transmission event tagged by an exclamation mark “!”, a reception event tagged by a tilde “~”, or a consumption event tagged by a question mark “?”. \mathcal{K} denotes $\{!, ~, ?\}$.

Every event occurring in a sequence diagram is decorated with a unique timestamp. \mathcal{T} denotes the set of timestamp tags. We use logical formulas with timestamp tags as free variables to impose constraints on the timing of events. By $\mathbb{F}(v)$ we denote the set of logical formulas whose free variables are contained in the set of timestamp tags v .

\mathcal{E} denotes the set of all events. Formally, an event is a triple of kind, message and timestamp tag

$$\mathcal{E} = \mathcal{K} \times \mathcal{M} \times \mathcal{T}$$

We define the functions

$$k. _ \in \mathcal{E} \rightarrow \mathcal{K}, \quad m. _ \in \mathcal{E} \rightarrow \mathcal{M}, \quad t. _ \in \mathcal{E} \rightarrow \mathcal{T}, \quad tr. _, re. _ \in \mathcal{E} \rightarrow \mathcal{L}$$

to yield the kind, message, timestamp tag, transmitter and receiver of an event, respectively. We also overload tr and re to yield the transmitter and receiver of a message.

We define the functions

$$tt. _ \in \mathcal{D} \rightarrow \mathbb{P}(\mathcal{T}), \quad ll. _ \in \mathcal{D} \rightarrow \mathbb{P}(\mathcal{L}), \quad ev. _ \in \mathcal{D} \rightarrow \mathbb{P}(\mathcal{E}), msg. _ \in \mathcal{D} \rightarrow \mathbb{P}(\mathcal{M})$$

to yield the timestamp tags, lifelines, events and messages of a sequence diagram, respectively.

The set of syntactically correct sequence diagrams \mathcal{D} is defined inductively. \mathcal{D} is the least set such that:

- $\mathcal{E} \subset \mathcal{D}$
- $d \in \mathcal{D} \Rightarrow \text{neg } d \in \mathcal{D} \wedge \text{assert } d \in \mathcal{D}$
- $d \in \mathcal{D} \wedge I \subseteq \mathbb{N}_0 \cup \{\infty\} \Rightarrow (\text{loop } I d) \in \mathcal{D}$
- $d_1, d_2 \in \mathcal{D} \Rightarrow d_1 \text{ alt } d_2 \in \mathcal{D} \wedge d_1 \text{ xalt } d_2 \in \mathcal{D} \wedge d_1 \text{ par } d_2 \in \mathcal{D}$
- $d_1, \dots, d_n \in \mathcal{D} \Rightarrow \text{seq } [d_1, \dots, d_n] \in \mathcal{D}$
- $d \in \mathcal{D} \wedge C \in \mathbb{F}(tt.d) \Rightarrow d \text{ tc } C \in \mathcal{D}$

The base case implies that any event is a sequence diagram. Any other sequence diagram is constructed from the basic ones through the application of operators for negation, assertion, loop, potential choice, mandatory choice, weak sequencing, parallel execution and time constraint. The full set of operators as defined by UML 2.0 [OMG04] is somewhat more comprehensive, and it is beyond the scope of this paper to treat them all. We focus on the operators that we find most essential.

We only consider sequence diagrams that are considered syntactically correct in UML 2.0. Also, we do not handle extra global combined fragments. This means that for all operators except from seq and par we assume that the operand(s) consist only of complete messages, i.e. messages where both the transmission, the reception and the consumption event is within the same operand. Formally, for all operands d_i of an operator different from seq and par, we require:

$$\forall m \in msg.d_i : \#\{ e \in ev.d_i \mid k.e = ! \wedge m.e = m \} \quad (10.1)$$

$$= \#\{ e \in ev.d_i \mid k.e = \sim \wedge m.e = m \}$$

$$\forall m \in msg.d_i : \#\{ e \in ev.d_i \mid k.e = ! \wedge m.e = m \} \quad (10.2)$$

$$= \#\{ e \in ev.d_i \mid k.e = ? \wedge m.e = m \}$$

where $\{\}$ denotes a multi-set and $\#$ is overloaded to yield the number of elements in a multi-set. A multi-set is needed here as the same message (consisting of a signal, a transmitter and a receiver) may occur more than once in the same diagram.

All single-event diagrams are considered syntactically correct. For all diagrams consisting of more than one event, we require that a message is complete if both the transmitter and the receiver lifelines are present in the diagram:

$$\begin{aligned} \forall m \in msg.d : (\#ev.d > 1 \wedge tr.m \in ll.d \wedge re.m \in ll.d) \Rightarrow \\ \#\{ e \in ev.d \mid k.e = ! \wedge m.e = m \} = \#\{ e \in ev.d \mid k.e = \sim \wedge m.e = m \} \end{aligned} \quad (10.3)$$

$$\begin{aligned} \forall m \in msg.d : (\#ev.d > 1 \wedge re.m \in ll.d) \Rightarrow \\ \#\{ e \in ev.d \mid k.e = \sim \wedge m.e = m \} = \#\{ e \in ev.d \mid k.e = ? \wedge m.e = m \} \end{aligned} \quad (10.4)$$

10.3.3 Representing Executions by Traces

We are mainly interested in communication scenarios. The actual content of messages is not significant for the purpose of this paper. Hence, we do not give any semantic interpretation of messages as such. The same holds for events except that the timestamp tag is assigned a timestamp in the form of a real number. \mathbb{R} denotes the set of all timestamps. Hence:²

$$\llbracket \mathcal{E} \rrbracket \stackrel{\text{def}}{=} \{(k, m, t \mapsto r) \mid (k, m, t) \in \mathcal{E} \wedge r \in \mathbb{R}\} \quad (10.5)$$

We define the function

$$r._- \in \llbracket \mathcal{E} \rrbracket \rightarrow \mathbb{R}$$

to yield the timestamp of an event.

A trace h is an element of $\llbracket \mathcal{E} \rrbracket^\omega$ that satisfies a number of well-formedness conditions. We use traces to represent executions. We require the events in h to be ordered by time: the timestamp of the i th event is less than or equal to the timestamp of the j th event if $i < j$. Formally:

$$\forall i, j \in [1.. \#h] : i < j \Rightarrow r.h[i] \leq r.h[j] \quad (10.6)$$

This means that two events may happen at the same time.

The same event takes place only once in the same execution. Hence, we also require:

$$\forall i, j \in [1.. \#h] : i \neq j \Rightarrow h[i] \neq h[j] \quad (10.7)$$

We also need to make sure that time will eventually progress beyond any finite point in time. The following constraint states that for each lifeline l represented by infinitely many events in the trace h , and for any possible timestamp t there must exist an l -event in h whose timestamp is greater than t :

$$\forall l \in \mathcal{L} : (\#e.l \circledcirc h = \infty \Rightarrow \forall t \in \mathbb{R} : \exists i \in \mathbb{N} : r.(e.l \circledcirc h)[i] > t) \quad (10.8)$$

where $e.l$ denotes the set of events that may take place on the lifeline l . Formally:

$$e.l \stackrel{\text{def}}{=} \{e \in \llbracket \mathcal{E} \rrbracket \mid (k.e = ! \wedge tr.e = l) \vee (k.e \in \{\sim, ?\} \wedge re.e = l)\} \quad (10.9)$$

For any single message, transmission must happen before reception, which must happen before consumption. However, in a particular sequence diagram we may have only the transmitter or the receiver lifeline present. Thus we get the following well-formedness requirements on traces, stating that if at any point in the trace we have a transmission event, up to that point we must have had at least as many transmissions as receptions of that particular message, and similarly for reception events with respect to consumptions:

$$\forall i \in [1.. \#h] : k.h[i] = ! \Rightarrow \quad (10.10)$$

$$\#(\{!\} \times \{m.h[i]\} \times U) \circledcirc h|_i > \#(\{\sim\} \times \{m.h[i]\} \times U) \circledcirc h|_i \\ \forall i \in [1.. \#h] : k.h[i] = \sim \Rightarrow \quad (10.11)$$

$$\#(\{\sim\} \times \{m.h[i]\} \times U) \circledcirc h|_i > \#(\{?\} \times \{m.h[i]\} \times U) \circledcirc h|_i$$

²The functions k, m, t, tr, re on \mathcal{E} are lifted to $\llbracket \mathcal{E} \rrbracket$ in the obvious manner.

where $U \stackrel{\text{def}}{=} \{t \mapsto r \mid t \in \mathcal{T} \wedge r \in \mathbb{R}\}$.

\mathcal{H} denotes the set of all well-formed traces.

We define three basic composition operators on trace sets, namely parallel execution, weak sequencing, and time constraint denoted by \parallel , \gtrsim , and \wr , respectively.

Informally, $s_1 \parallel s_2$ is the set of all traces such that

- all events from one trace in s_1 and one trace in s_2 are included (and no other events),
- the ordering of events from each of the traces is preserved.

Formally:

$$\begin{aligned} s_1 \parallel s_2 &\stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty : \\ &\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1 \wedge \\ &\quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in s_2\} \end{aligned} \quad (10.12)$$

In this definition, we make use of an oracle, the infinite sequence p , to resolve the non-determinism in the interleaving. It determines the order in which events from traces in s_1 and s_2 are sequenced. π_2 is a projection operator returning the second element of a pair.

For $s_1 \gtrsim s_2$ we have the constraint that events on one lifeline from one trace in s_1 should come before events from one trace in s_2 on the same lifeline:

$$\begin{aligned} s_1 \gtrsim s_2 &\stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \forall l \in \mathcal{L} : \\ &\quad e.l \circledcirc h = e.l \circledcirc h_1 \cap e.l \circledcirc h_2\} \end{aligned} \quad (10.13)$$

Time constraint is defined as

$$s \wr C \stackrel{\text{def}}{=} \{h \in s \mid h \models C\} \quad (10.14)$$

where $h \models C$ holds if for all possible assignments of timestamps to timestamp tags done by h , there is an assignment of timestamps to the remaining timestamp tags in C (possibly none) such that C evaluates to true. For example, if

$$h = \langle (k_1, m_1, t_1 \mapsto r_1), (k_2, m_2, t_2 \mapsto r_2), (k_3, m_3, t_2 \mapsto r_3) \rangle, \quad C = t_1 < t_2$$

then $h \models C$ if $r_1 < r_2$ and $r_1 < r_3$.

10.3.4 Interaction Obligations

The semantics of sequence diagrams will eventually be defined as sets of interaction obligations. An interaction obligation is a pair (p, n) of sets of traces where the first set is interpreted as the set of positive traces and the second set is the set of negative traces. The term obligation is used to explicitly convey that any implementation of a specification is obliged to fulfill each specified alternative.

\mathcal{O} denotes the set of all interaction obligations. Parallel execution, weak sequencing and time constraint are overloaded from sets of traces to interaction obligations as follows:

$$(p_1, n_1) \parallel (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \parallel p_2, (n_1 \parallel (p_2 \cup n_2)) \cup (n_2 \parallel p_1)) \quad (10.15)$$

$$(p_1, n_1) \succsim (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \succsim p_2, (n_1 \succsim (n_2 \cup p_2)) \cup (p_1 \succsim n_2)) \quad (10.16)$$

$$(p, n) \wr C \stackrel{\text{def}}{=} (p \wr C, n \cup (p \wr \neg C)) \quad (10.17)$$

An obligation pair (p, n) is contradictory if $p \cap n \neq \emptyset$.

The operators for parallel execution, weak sequencing and time constraint are also overloaded to sets of interaction obligations:

$$O_1 \parallel O_2 \stackrel{\text{def}}{=} \{o_1 \parallel o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\} \quad (10.18)$$

$$O_1 \succsim O_2 \stackrel{\text{def}}{=} \{o_1 \succsim o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\} \quad (10.19)$$

$$O \wr C \stackrel{\text{def}}{=} \{o \wr C \mid o \in O\} \quad (10.20)$$

In order to define the semantics of infinite loops in sequence diagrams, we introduce the notion of “chain”. Intuitively, an infinite loop corresponds to infinitely many weak sequencing steps. A chain of interaction obligations is an infinite sequence of obligations such that each element is a sequential composition of the previous obligation in the chain and some other appropriate obligation. For a set O of interaction obligations, its chains is defined as:

$$\begin{aligned} \text{chains}(O) \stackrel{\text{def}}{=} & \{ \bar{o} \in \mathcal{O}^\infty \mid \\ & \bar{o}[1] \in O \wedge \\ & \forall j \in \mathbb{N} : \exists o \in O : \bar{o}[j+1] = \bar{o}[j] \succsim o \} \end{aligned} \quad (10.21)$$

From a chain \bar{o} of interaction obligations, we obtain a chain of traces by selecting one positive trace from each obligation in the chain \bar{o} , and such that each trace in the chain is an extension (by means of weak sequencing) of the previous trace in the chain. For a chain \bar{o} of interaction obligations, we define its positive chains of traces as:

$$\begin{aligned} \text{pos}(\bar{o}) \stackrel{\text{def}}{=} & \{ \bar{t} \in \mathcal{H}^\infty \mid \forall j \in \mathbb{N} : \\ & \bar{t}[j] \in \pi_1(\bar{o}[j]) \wedge \\ & \exists t \in \mathcal{H} : \bar{t}[j+1] \in \{ \bar{t}[j] \} \succsim \{ t \} \} \end{aligned} \quad (10.22)$$

For a chain \bar{o} of interaction obligations, we get a negative chain of traces by selecting the traces such that the first one is negative in some obligation $\bar{o}[i]$ and all the following traces belong to the negative trace sets of the corresponding obligations. By starting from some obligation $\bar{o}[i]$ and not just from $\bar{o}[1]$, we take into account that a negative trace may have been positive during a finite number of initial iterations. As for $\text{pos}(\bar{o})$, each trace in the chain is a weak sequencing extension of the previous trace in the chain. According to definition (10.16), once we have selected a negative trace, all extensions of this trace will also be negative. Hence, we get the following definition for the negative chains of traces:

$$\begin{aligned} \text{negs}(\bar{o}) \stackrel{\text{def}}{=} & \{ \bar{t} \in \mathcal{H}^\infty \mid \exists i \in \mathbb{N} : \forall j \in \mathbb{N} : \\ & \bar{t}[j] \in \pi_2(\bar{o}[j+i-1]) \wedge \\ & \exists t \in \mathcal{H} : \bar{t}[j+1] \in \{ \bar{t}[j] \} \succsim \{ t \} \} \end{aligned} \quad (10.23)$$

For a chain of traces \bar{t} we have that for each $l \in \mathcal{L}$, the sequence

$$e.l \odot \bar{t}[1], e.l \odot \bar{t}[2], e.l \odot \bar{t}[3], \dots$$

constitutes a chain whose elements are ordered by \sqsubseteq . We use $\sqcup_l \bar{t}$ to denote the least upper bound of this chain of sequences (with respect to \sqsubseteq). Since sequences may be infinite such least upper bounds always exist.

For a chain of traces \bar{t} , we define its set of approximations $\sqcup \bar{t}$ as:

$$\sqcup \bar{t} \stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \forall l \in \mathcal{L} : e.l \odot h = \sqcup_l \bar{t}\} \quad (10.24)$$

For a chain of interaction obligations \bar{o} , we then define the obligation $\sqcup \bar{o}$ as:

$$\sqcup \bar{o} \stackrel{\text{def}}{=} (\cup_{\bar{t} \in pos(\bar{o})} \sqcup \bar{t}, \cup_{\bar{t} \in negs(\bar{o})} \sqcup \bar{t}) \quad (10.25)$$

For a set of interaction obligations, we define a loop construct $\mu_n O$, where n denotes the number of times the loop is iterated. $\mu_n O$ is defined inductively as follows:

$$\mu_0 O \stackrel{\text{def}}{=} \{(\{\langle\rangle\}, \emptyset)\} \quad (10.26)$$

$$\mu_1 O \stackrel{\text{def}}{=} O \quad (10.27)$$

$$\mu_n O \stackrel{\text{def}}{=} O \gtrsim \mu_{n-1} O \quad \text{for } 1 < n < \infty \quad (10.28)$$

$$\mu_\infty O \stackrel{\text{def}}{=} \{\sqcup \bar{o} \mid \bar{o} \in chains(O)\} \quad (10.29)$$

Finally, to facilitate defining the common alternative choice operator alt , we define an operator for inner union of sets of interaction obligations:

$$O_1 \uplus O_2 \stackrel{\text{def}}{=} \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2\} \quad (10.30)$$

\biguplus is the generalization of \uplus to arbitrary many sets:

$$\biguplus_{i \in I} O_i \stackrel{\text{def}}{=} \{\biguplus_{i \in I} o_i \mid o_i \in O_i \text{ for all } i \in I\} \quad (10.31)$$

where I is an index set and

$$\biguplus_{i \in I} (p_i, n_i) \stackrel{\text{def}}{=} (\bigcup_{i \in I} p_i, \bigcup_{i \in I} n_i) \quad (10.32)$$

10.3.5 Semantics of Sequence Diagrams

The semantics of sequence diagrams is defined by a function

$$[\![_]\!] \in \mathcal{D} \rightarrow \mathbb{P}(\mathcal{O})$$

that for any sequence diagram d yields a set $[\![d]\!]$ of interaction obligations.

An event is represented by infinitely many unary positive traces – one for each possible assignment of timestamp to its timestamp tag:

$$\llbracket (k, m, t) \rrbracket \stackrel{\text{def}}{=} \{(\{\langle(k, m, t \mapsto r)\rangle \mid r \in \mathbb{R}\}, \emptyset) \} \quad \text{if } (k, m, t) \in \mathcal{E} \quad (10.33)$$

The neg construct defines negative traces:

$$\llbracket \text{neg } d \rrbracket \stackrel{\text{def}}{=} \{(\{\langle\rangle\}, p \cup n) \mid (p, n) \in \llbracket d \rrbracket\} \quad (10.34)$$

Notice that a negative trace cannot be made positive by reapplying neg. Negative traces remain negative. Negation is an operation that characterizes traces absolutely and not relatively. The intuition is that the focus of the neg construct is on characterizing the positive traces in the operand as negative. Negative traces will always propagate as negative to the outermost level. The neg construct defines the empty trace as positive. This facilitates the embedding of negs in sequence diagrams also specifying positive behavior.

The assert construct makes all inconclusive traces negative. Otherwise the sets of positive and negative traces are left unchanged:

$$\llbracket \text{assert } d \rrbracket \stackrel{\text{def}}{=} \{(p, n \cup (\mathcal{H} \setminus p)) \mid (p, n) \in \llbracket d \rrbracket\} \quad (10.35)$$

Note that contradictory obligation pairs remain contradictory.

Constructs of sequence diagrams are graphically either (almost) rectangular or a point (an event). In operands or diagrams, such free constructs are ordered according to their upper left corner. The ordering operator is the weak sequencing operator. In our formal syntax given in Section 10.3.2 all constructs are explicit.

Weak sequencing is defined by the seq construct:

$$\begin{aligned} \llbracket \text{seq } [d] \rrbracket &\stackrel{\text{def}}{=} \llbracket d \rrbracket \\ \llbracket \text{seq } [D, d] \rrbracket &\stackrel{\text{def}}{=} \llbracket \text{seq } [D] \rrbracket \succ \llbracket d \rrbracket \end{aligned} \quad (10.36)$$

for d a syntactically correct sequence diagram and D a non-empty list of such sequence diagrams.

The alt construct defines potential traces. The semantics is the union of the trace sets for both positive and negative:

$$\llbracket d_1 \text{ alt } d_2 \rrbracket \stackrel{\text{def}}{=} \llbracket d_1 \rrbracket \uplus \llbracket d_2 \rrbracket \quad (10.37)$$

The xalt construct defines mandatory choices. All implementations must be able to handle every interaction obligation:

$$\llbracket d_1 \text{ xalt } d_2 \rrbracket \stackrel{\text{def}}{=} \llbracket d_1 \rrbracket \cup \llbracket d_2 \rrbracket \quad (10.38)$$

Notice that the sets of negative traces are not combined as is the case with the alt. This is due to the fact that we want to allow behaviors that are positive in one interaction obligation to be negative in another interaction obligation. The intuition behind this is as follows: All positive behaviors in an interaction obligation serve the same overall purpose, e.g. different ways of making beef. Alternative ways of making beef can be introduced by the alt operator.

Hence, a behavior cannot be present in both the positive and negative trace sets of an interaction obligation as this would lead to a contradictory specification. However, behaviors specified by different interaction obligations are meant to serve different purposes, e.g. make beef dish and make vegetarian dish. There is nothing wrong about stating that a behavior which is positive in one interaction obligation is negative in another. E.g. steak beef would definitely be positive in a beef context and negative in a vegetarian context. By insisting on separate negative sets of interaction obligations we achieve this wanted property.

The `par` construct represents a parallel merge. Any trace involving a negative trace will remain negative in the resulting interaction obligation:

$$\llbracket d_1 \mathsf{par} d_2 \rrbracket \stackrel{\text{def}}{=} \llbracket d_1 \rrbracket \parallel \llbracket d_2 \rrbracket \quad (10.39)$$

The `tc` construct defines the effect of a time constraint. The positive traces of the operand that do not fulfill the constraint become negative in the result. The negative traces of the operand remain negative regardless of whether they fulfill the constraint:

$$\llbracket d \mathsf{tc} C \rrbracket \stackrel{\text{def}}{=} \llbracket d \rrbracket \wr C \quad (10.40)$$

The semantics of the loop construct is the same as the inner union of the semantics for doing the contents of the loop for each possible number in the set I , where $I \subseteq \mathbb{N}_0 \cup \{\infty\}$:

$$\llbracket \mathsf{loop} I d \rrbracket \stackrel{\text{def}}{=} \biguplus_{i \in I} \mu_i \llbracket d \rrbracket \quad (10.41)$$

We use inner union \biguplus (as in `alt`) and not ordinary union (as in `xalt`) since we do not want to require an implementation to implement the loop for all possible values in the set I .

10.4 Two Abstractions

An example to illustrate the importance of distinguishing between the message reception and the message consumption event when dealing with timed specifications goes as follows: A restaurant chain specifies in a sequence diagram (see Figure 10.3) that it should never take more than 10 minutes to prepare a beef dish. The specification is handed over to the local restaurant owner who takes these requirements as an input to the design process of her/his local restaurant. When testing the time it takes to prepare a beef the restaurant finds that it is in accordance with the timing requirements. However, when the restaurant chain inspector comes to verify that the timing policies of the chain are obeyed in the operational restaurant he finds that it takes much longer time than 10 minutes to prepare the beef. Thus, the inspector claims that the restaurant is not working according to the timing requirements while the restaurant owner claims they are working according to the requirements. Who is right? According to UML both are right as there is no notion of bringing of communication in UML. Whether the message arrival of “main dish please” to the kitchen shall be regarded as message reception or consumption is not defined in the semantics of UML, and hence, it is up to the users of the diagrams to interpret the meaning.

In this section we define two abstractions over the triple event semantics that match the two different views in the example above, namely the standard interpretation and the black-box interpretation.

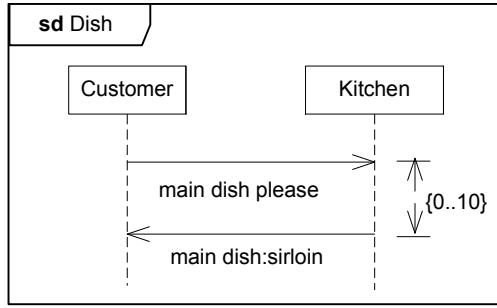


Figure 10.3: Restaurant specification with time constraint

10.4.1 Standard Interpretation

The standard interpretation is meant to represent the traditional way of interpreting graphical sequence diagrams, namely that the input event of a message at a lifeline represents a consumption. We then only take send (!) and consume (?) events into consideration. Thus, we abstract away the fact that a message will arrive and be stored before it is consumed by the object. The standard interpretation sees graphical sequence diagrams like the diagram in Figure 10.3 as “standard diagrams”.

The syntax and semantics of standard diagrams is defined in exactly the same manner as for general sequence diagrams in Section 10.3, with the following exceptions.

Standard diagrams do not contain any reception events, meaning that the syntactical well-formedness criteria (10.1) and (10.4) are no longer relevant. Criteria (10.2) is kept for standard diagrams, while criteria (10.3) is replaced by:

$$\begin{aligned} \forall m \in msg.d : (\#ev.d > 1 \wedge tr.m \in ll.d \wedge re.m \in ll.d) \Rightarrow & \quad (10.42) \\ \#\{\{e \in ev.d \mid k.e = ! \wedge m.e = m\} = \#\{\{e \in ev.d \mid k.e = ? \wedge m.e = m\}\} \end{aligned}$$

Also, the semantics of events is redefined as follows:

$$\begin{aligned} \llbracket (k, m, t) \rrbracket &\stackrel{\text{def}}{=} \{(\{h' \sim \langle (k, m, t \mapsto r) \rangle \sim h'' \in \mathcal{H} \mid \\ &\quad h', h'' \in E(l, \sim)^\omega \wedge \#h' < \infty \wedge r \in \mathbb{R}\}, \\ &\quad \emptyset)\} \end{aligned} \quad (10.43)$$

where $l = tr.m$ if $k = !$ and $l = re.m$ if $k = ?$, and $E(l, \sim)$ is the set of all events $e \in \llbracket \mathcal{E} \rrbracket$ such that $re.e = l$ and $k.e = \sim$.

This definition says essentially that in a standard diagram, reception events may happen anywhere on the relevant lifeline (as long as the well-formedness conditions (10.10) and (10.11) are obeyed) since they are considered irrelevant in this setting.

If we apply the standard interpretation to the diagram in Figure 10.3, every positive trace h is such that

$$\begin{aligned} \{e \in \llbracket \mathcal{E} \rrbracket \mid k.e \neq \sim\} \circledcirc h = \\ \langle (!, m, t_1 \mapsto r_1), (? , m, t_3 \mapsto r_3), (!, n, t_4 \mapsto r_4), (? , n, t_6 \mapsto r_6) \rangle \end{aligned}$$

where $r_4 \leq r_3 + 10$, m stands for “main dish please” and n stands for “main dish:sirloin”. The implicit reception of m can happen at any point between the corresponding transmission and consumption events, and similarly for n (and any other message).

10.4.2 Black-Box Interpretation

The black-box interpretation represents the view where the input event of a message at a lifeline represents a reception event. The black-box interpretation sees graphical sequence diagrams like the diagram in Figure 10.3 as “black-box diagrams”.

As with standard diagrams, the syntax and semantics of black-box diagrams is defined in exactly the same manner as for general sequence diagrams in Section 10.3, with the following exceptions.

Black-box diagrams do not contain any consumption events, meaning that the syntactical well-formedness criteria (10.2) and (10.4) are no longer relevant. Criteria (10.1) and (10.3) are kept for black-box diagrams.

Also, the semantics of events is redefined as follows:

$$\llbracket (k, m, t) \rrbracket \stackrel{\text{def}}{=} \{(\{h' \smallfrown \langle (k, m, t \mapsto r) \rangle \smallfrown h'' \in \mathcal{H} \mid h', h'' \in E(l, ?)^\omega \wedge \#h' < \infty \wedge r \in \mathbb{R}\}, \emptyset)\} \quad (10.44)$$

where $l = tr.m$ if $k = !$ and $l = re.m$ if $k = \sim$, and $E(l, ?)$ is the set of all events $e \in \llbracket \mathcal{E} \rrbracket$ such that $re.e = l$ and $k.e = ?$.

If we apply the black-box interpretation to the diagram in Figure 10.3, every positive trace h is such that

$$\{e \in \llbracket \mathcal{E} \rrbracket \mid k.e \neq ?\} \circledcirc h = \langle (!, m, t_1 \mapsto r_1), (\sim, m, t_2 \mapsto r_2), (!, n, t_4 \mapsto r_4), (\sim, n, t_5 \mapsto r_5) \rangle$$

where $r_4 \leq r_2 + 10$. Note that we do not impose any constraint on the implicit consumption events, except that the consumption cannot take place before its reception (if it takes place at all).

10.5 The General Case

We have shown that input events are most naturally (standard) interpreted as consumption when they appear on lifelines that represent atomic processes and their concrete implementations should be derived from the lifelines. We have also shown that there are reasons, e.g. timing constraints, that sometimes make it necessary to consider the input event as representing the reception. Moreover, we have seen that timing constraints may also make good sense when applied to consumption events.

In fact we believe that notation for both reception and consumption events are necessary, but that most often for any given message a two-event notation will suffice. Sometimes the message will end in the reception and sometimes in the consumption, but seldom there is a need to make both the reception and the consumption explicit. There are, however, exceptions where all

three events must be present to convey the exact meaning of the scenario. Hence, we will in the following introduce graphical notation in order to be able to explicitly state whether a message input event at a lifeline shall be interpreted as a reception event or a consumption event. That is, whether standard or black-box interpretation shall be applied.

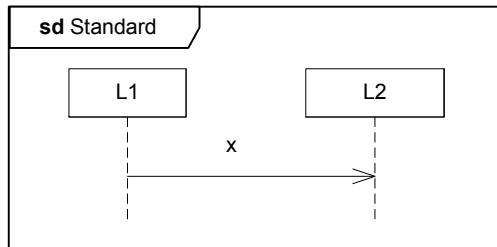


Figure 10.4: Graphical syntax for specifying standard interpretation

Figure 10.4 shows the graphical notation to specify that a message input event at a lifeline shall be interpreted as a consumption event. Syntactically this notation is equal to the one applied for ordinary two-event sequence diagrams.

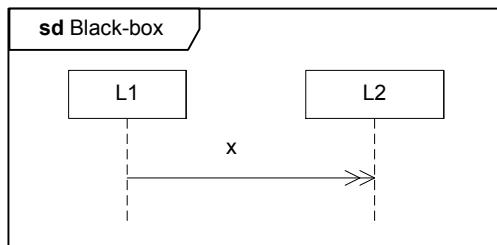


Figure 10.5: Graphical syntax for specifying black-box interpretation

We express that a message input event at a lifeline shall be interpreted as a reception event and thus be given black-box interpretation by the double arrowhead as shown in Figure 10.5. We will in the following give some examples of the full approach describing reception events as well as consumption events explicitly.

Let us return to the dinner example where we may assume that the cook is not really one single person, but actually a chef and his apprentice. We may decompose the cook lifeline into new sequence diagrams where the chef and apprentice constitute the internal lifelines. We have shown this in Figure 10.6 for the preparation of beef shown originally in Figure 10.2. Let us assume that the apprentice wants to go and get the meat before heating the stove. His priorities may be so because heating the stove is more of a burden, or because the refrigerator is closer at hand. For our purposes we would like to describe a scenario that highlights that the apprentice fetches the meat before heating the stove even though he received the order to turn on the heat first.

In Figure 10.6 we have shown some explicit reception events, but we have chosen not to show explicitly the corresponding consumptions. This is because our needs were to describe the relationship between the receptions and the actions (outputs) of the apprentice.

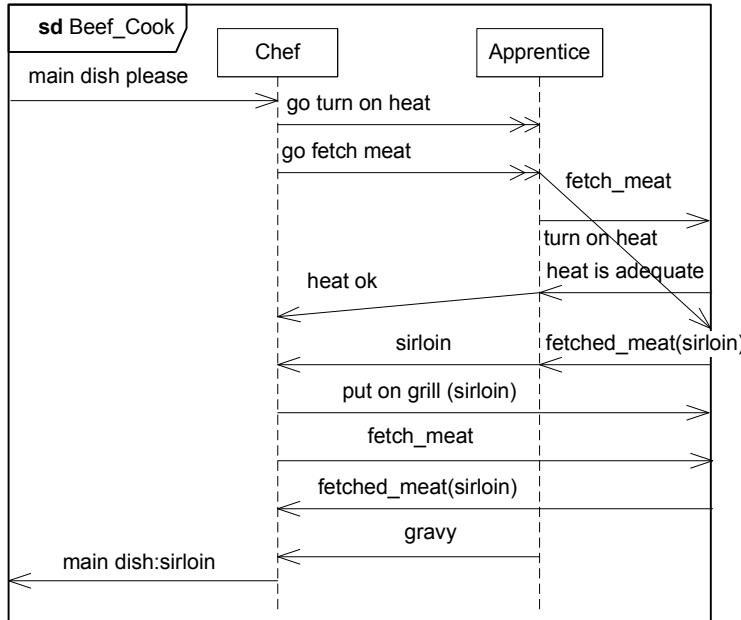


Figure 10.6: Prioritizing to fetch the meat

The consumptions were considered less important. The disadvantage of this is that we cannot from Figure 10.6 deduce whether the apprentice actually fetched the meat because he received the order “go fetch meat” or the order “go turn on heat”. The reader should appreciate that the “fetch_meat” message crosses the other messages only due to the need to graphically let the formal gates match the events on the decomposed Cook lifeline shown in Figure 10.2. Semantically there is no ordering between gates. For a formal treatment of gates, see Appendix 10.A.

If we want to give an even more detailed account of the apprentice’s options, we may introduce both reception and consumption events. We have done so in Figure 10.7.

In Figure 10.7 we see that the chef instructs the apprentice to go turn on heat and to go and fetch meat. The apprentice makes independent decisions for the order of consumption. Here he has decided to consume the order to go fetch meat before consuming go turn on heat. Now we can easily see that the apprentice reacts adequately to the consumptions. It is of course rather risky for the apprentice not to react immediately to the chef’s order to turn on the heat, but we may remedy this by timetagging the message receptions of “go turn on heat” and “go fetch meat”. Then we specify that the scenario is only valid if these receptions are sufficiently close together in time by a formula including these time tags.

As the examples in this section demonstrate, we have cases where we need to explicitly describe both reception and consumption events in the same diagram, but seldom for the same message. This means that general diagrams may contain standard, black-box as well as three-event arrows. The semantics of such diagrams is given by the definitions in Section 10.3 with two exceptions:

- The semantics of a consumption event of a standard arrow should be as for consumption events in the standard case (see Section 10.4.1).

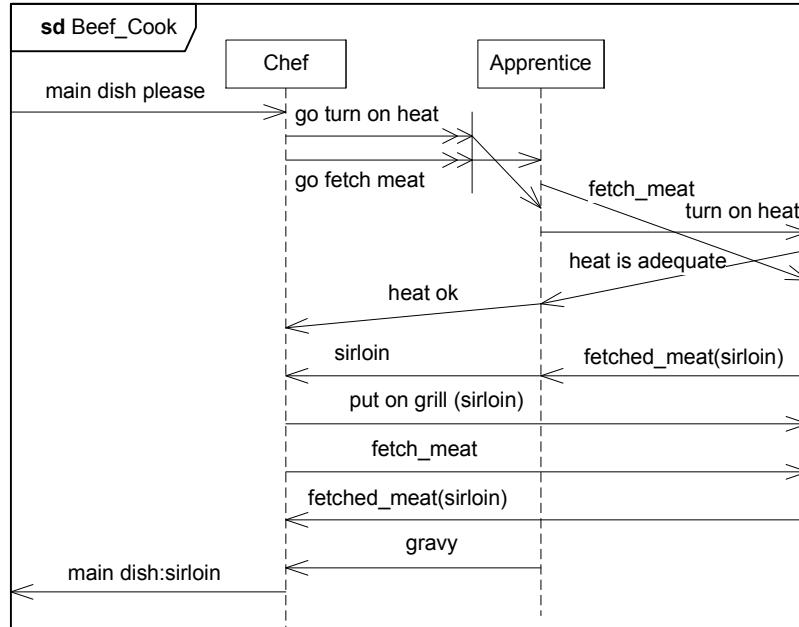


Figure 10.7: The whole truth

- The semantics of a receive event of a black-box arrow should be as for receive events in the black-box case (see Section 10.4.2).

10.6 Refinement

Refinement means to add information to a specification such that the specification becomes closer to an implementation. The set of potential traces will be narrowed and situations that we have not yet considered will be supplemented. We define formally two forms of refinement — glass-box refinement which takes the full semantics of the diagram into account, and black-box refinement which only considers changes that are externally visible.

Negative traces must always remain negative in a refinement, while positive traces may remain positive or become negative if the trace has been cancelled out. Inconclusive traces may go anywhere.

10.6.1 Definition of Glass-Box Refinement

An interaction obligation (p_2, n_2) is a refinement of an interaction obligation (p_1, n_1) , written $(p_1, n_1) \rightsquigarrow_r (p_2, n_2)$, i

$$n_1 \subseteq n_2 \quad \wedge \quad p_1 \subseteq p_2 \cup n_2 \tag{10.45}$$

A set of interaction obligations O' is a glass-box refinement of a set O , written $O \rightsquigarrow_g O'$, i

$$\forall o \in O : \exists o' \in O' : o \rightsquigarrow_r o' \tag{10.46}$$

A sequence diagram d' is then a glass-box refinement of a sequence diagram d , written $d \rightsquigarrow_g d'$, i

$$\llbracket d \rrbracket \rightsquigarrow_g \llbracket d' \rrbracket \quad (10.47)$$

The refinement semantics supports the classical notion of compositional refinement providing a firm foundation for compositional analysis, verification and testing. In Appendix 10.D we prove that refinement as defined above is reflexive and transitive. Reflexivity means that a sequence diagram is viewed as a refinement of itself. Transitivity is important, as it means that a sequence diagram resulting from successive refinement steps will still be a valid refinement of the original specification.

Also, in Appendix 10.E we prove that refinement is monotonic with respect to the composition operators presented in Section 10.3.5, except from assert. For assert, we have monotonicity in the special case of narrowing defined below. Monotonicity is important, as it ensures compositionality in the sense that the different operands of a specification may be refined separately.

10.6.2 Supplementing and Narrowing

Supplementing and narrowing are special cases of the general notion of refinement. Supplementing categorizes inconclusive behavior as either positive or negative. An interaction obligation (p_2, n_2) supplements an interaction obligation (p_1, n_1) , written $(p_1, n_1) \rightsquigarrow_s (p_2, n_2)$, i

$$(n_1 \subset n_2 \wedge p_1 \subseteq p_2) \vee (n_1 \subseteq n_2 \wedge p_1 \subset p_2) \quad (10.48)$$

Narrowing reduces the allowed behavior to match the problem better. An interaction obligation (p_2, n_2) narrows an interaction obligation (p_1, n_1) , written $(p_1, n_1) \rightsquigarrow_n (p_2, n_2)$, i

$$p_2 \subset p_1 \wedge n_2 = n_1 \cup (p_1 \setminus p_2) \quad (10.49)$$

10.6.3 Example of Glass-Box Refinement

We want to refine the Beef_Cook diagram presented in Figure 10.7. In a glass-box refinement we are interested in the complete traces described by the diagram, and a selection and/or a supplement of these traces.

Figure 10.8 is a glass-box refinement of Figure 10.7. In this diagram we state that we no longer want gravy, but Bearnaise sauce instead. Defining gravy as negative is a narrowing, as it means to reduce the set of positive traces of the original specification. The traces with Bearnaise sauce was earlier considered inconclusive (i.e. neither positive nor negative), but are now defined as positive. This is an example of supplementing. In addition, the diagram in Figure 10.8 permits using no sauce at all. This is because the neg fragment also introduces the empty trace ($\langle \rangle$) as positive. We summarize these changes in Figure 10.9.

10.6.4 Definition of Black-Box Refinement

Black-box refinement may be understood as refinement restricted to the externally visible behavior. We define the function

$$ext \in \mathcal{H} \times \mathbb{P}(\mathcal{L}) \rightarrow \mathcal{H}$$

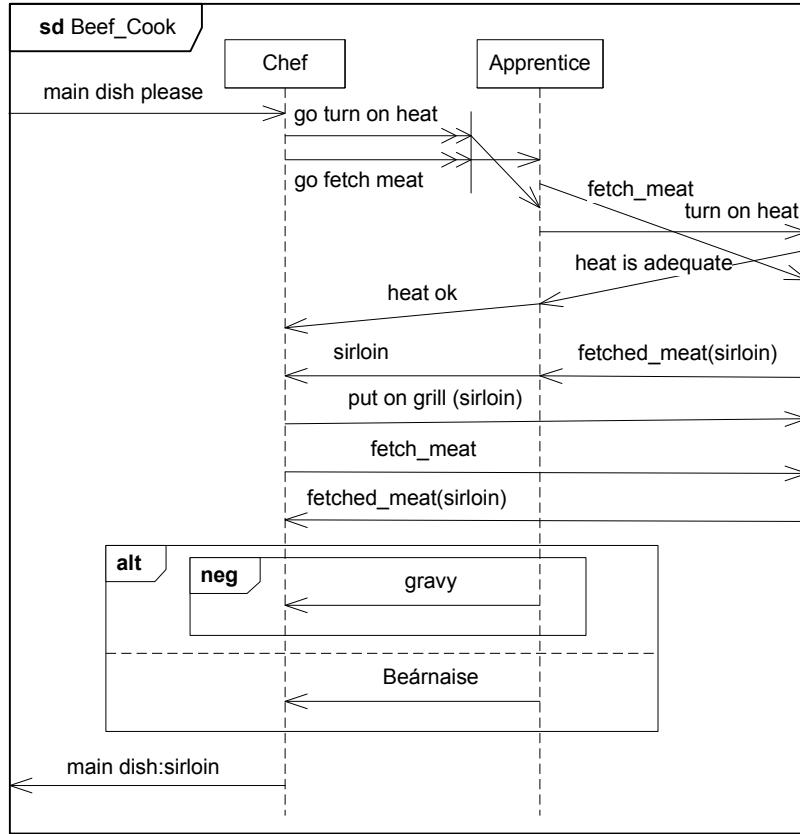


Figure 10.8: Glass-box refinement of Beef_Cook

to yield the trace obtained from the trace given as first argument by filtering away those events that are internal with respect to the set of lifelines given as second argument, i.e.:

$$\text{ext}(h, l) \stackrel{\text{def}}{=} \{e \in [\mathcal{E}] \mid (tr.e \notin l \vee re.e \notin l) \wedge k.e \neq ?\} \circledcirc h \quad (10.50)$$

The *ext* operator is overloaded to sets of traces, to pairs of sets of traces, and sets of pairs of sets of traces in the standard pointwise manner:

$$\text{ext}(s, l) \stackrel{\text{def}}{=} \{\text{ext}(h, l) \mid h \in s\} \quad (10.51)$$

$$\text{ext}((p, n), l) \stackrel{\text{def}}{=} (\text{ext}(p, l), \text{ext}(n, l)) \quad (10.52)$$

$$\text{ext}(O, l) \stackrel{\text{def}}{=} \{\text{ext}((p, n), l) \mid (p, n) \in O\} \quad (10.53)$$

A sequence diagram d' is a black-box refinement of a sequence diagram d , written $d \rightsquigarrow_b d'$,

$$\text{ext}([\mathbb{d}], ll.d) \rightsquigarrow_g \text{ext}([\mathbb{d}'], ll.d') \quad (10.54)$$

Notice that the *ext* operator also filters away all consumption events regardless of lifeline, as was the case with black-box interpretation of sequence diagrams. Thus, black-box refinement is mainly relevant in the context of black-box interpretation (even though it may also be applied to standard diagrams).

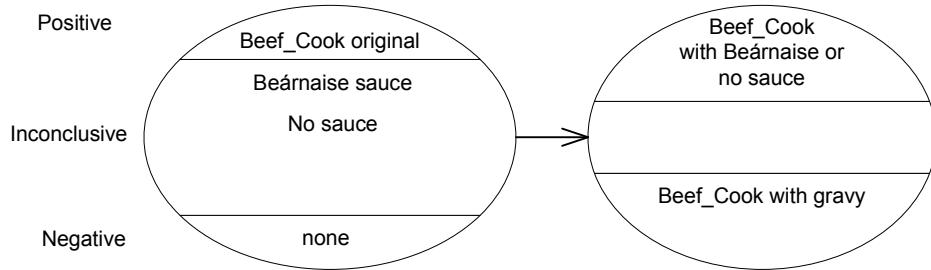


Figure 10.9: Summary glass-box refinement

10.6.5 Example of Black-Box Refinement

It is obvious from the definition of black-box refinement that any glass-box refinement is also a black-box refinement. What would be a black-box refinement in our Beef_Cook context, but not a glass-box refinement? If we in a refinement of the specification in Figure 10.7 had just replaced the gravy with Bearnaise, this change would not affect the externally visible behavior of Beef_Cook as it is defined, and would therefore be a legal black-box refinement. However, it would not be a glass-box refinement since the traces involving gravy have been lost (they are now inconclusive), and this violates the definition.

10.6.6 Detailing

When we increase the granularity of sequence diagrams we call this a detailing of the specification. The granularity can be altered in two different ways: either by decomposing the lifelines such that their inner parts and their internal behavior are displayed in the diagram or by changing the data-structure of messages such that they convey more detailed information.

Black-box refinement is sufficiently general to formalize lifeline decompositions that are not externally visible. However, many lifeline decompositions are externally visible. As an example of a lifeline decomposition that is externally visible, consider the decomposition of Beef_Cook in Figure 10.6. The messages that originally (in Figure 10.2) had the Cook as sender/receiver, now have the chef or the apprentice as sender/receiver.

To allow for this, we extend the definition of black-box refinement with the notion of a lifeline substitution. The resulting refinement relation is called lifeline decomposition. A lifeline substitution is a partial function of type $\mathcal{L} \rightarrow \mathcal{L}$. \mathcal{LS} denotes the set of all such substitutions. We define the function

$$\text{subst} \in \mathcal{D} \times \mathcal{LS} \rightarrow \mathcal{D}$$

such that $\text{subst}(d, ls)$ yields the sequence diagram obtained from d by substituting every lifeline l in d for which ls is defined with the lifeline $ls(l)$.

We then define that a sequence diagram d' is a lifeline decomposition of a sequence diagram d with respect to a lifeline substitution ls , written $d \rightsquigarrow_l^{ls} d'$, i

$$d \rightsquigarrow_b \text{subst}(d', ls) \tag{10.55}$$

Changing the data-structure of messages may be understood as black-box refinement modulo

a translation of the externally visible behavior. This translation is specified by a sequence diagram t , and we refer to this as an interface refinement.

We define that a sequence diagram d' is an interface refinement of a sequence diagram d with respect to a sequence diagram t , written $d \rightsquigarrow_i^t d'$, i

$$d \rightsquigarrow_b \text{seq } [t, d'] \quad (10.56)$$

Detailing may then be understood as the transitive and reflexive closure of lifeline decomposition and interface refinement.

10.6.7 Refinement Through Time Constraints

Having given examples of refinement in terms of pure event manipulation and trace selection, we go on to present an example where time constraints represent the refinement constructs. We

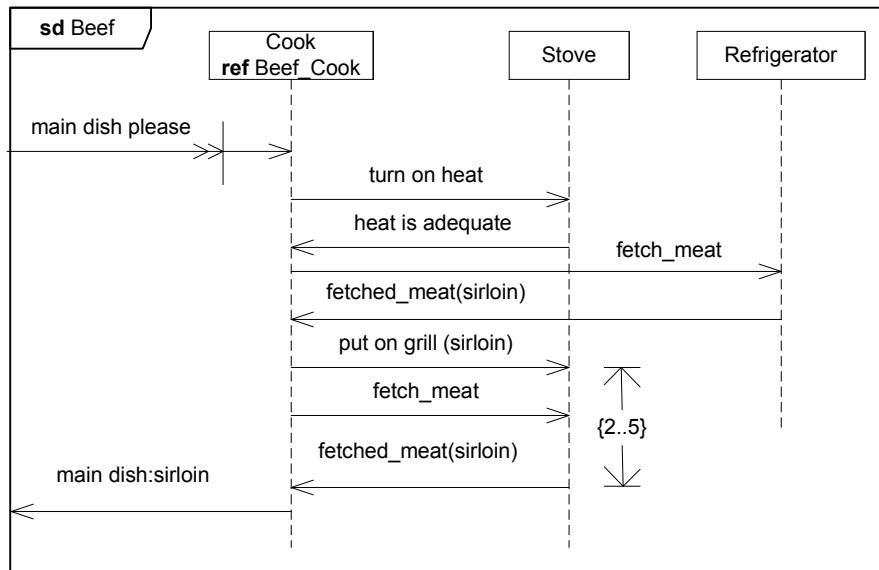


Figure 10.10: Imposing constraints on timing

will now introduce two time refinements as indicated in Figures 10.10 and 10.11. First we would like to make sure that beers are neither burned nor raw when fetched from the stove. To make sure that this constraint holds we will put the time constraint on the consumption event of the “put on grill” message. This is because it is the time that the beef is actually present on the stove that matters with respect to how much it is grilled, not the time the beef lies on a plate beside the stove waiting for free space on the stove. All behaviors that do not meet this time constraint are considered negative according to definition (10.40) of time constraint semantics. Traces that originally were positive are because of the new time constraint now defined as negative. Thus, this step constitutes a glass-box refinement according to definition (10.47). In fact, it is a narrowing as defined by definition (10.49). Since the consumption events and transmit events locally define the object behavior, it is only the behavior of the stove that is affected by this refinement step, and not the environment. Using double arrowhead on the “put on grill” message we would

not be able to express the intended refinement because it is necessary to talk about message consumption. On the other hand, comparing Figure 10.10 with the original diagram in Figure 10.2, we have replaced a standard arrow with a three-event arrow. This is a valid refinement, as it means to make explicit one of the implicit reception events that are already present in the semantics of Figure 10.2. This change was made in order to make Figure 10.10 a valid lifeline decomposition of the Kitchen lifeline in Figure 10.11, which we will describe next.

We would now like to limit the overall time it takes to prepare a beef. This represents a customer requirement on the kitchen as illustrated in Figure 10.11. However, the customer does not care about the details of beef preparation, just that it is prepared in time. As seen from Figure 10.11 this can be interpreted as a time constraint on the reception event. In the same manner as with the glass-box refinement above, the introduction of the time constraint is a narrowing that “moves” traces from the set of positive traces to the set of negative traces. We are not concerned about where the beef spends its time in the kitchen during the preparation process, just that it is prepared in time.

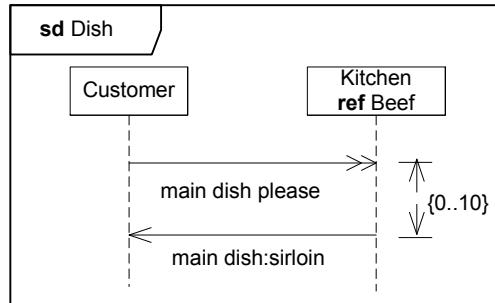


Figure 10.11: Customer requirement on the beef preparation time

10.7 Conclusions and Related Work

We have presented Timed STAIRS, a formal approach to the step-wise, incremental development of timed sequence diagrams. It is based on trace semantics. Traces are sequences of events. Events are representations of sending, receiving and consuming messages.

Three event semantics of sequence diagrams has been considered before. In [EMR97] the event ordering imposed by the MSCs is used to determine the physical architecture needed for implementing the specified behavior such as a FIFO buffer between each of the processes. In Timed STAIRS we implicitly assume that every message has one associated input buffer unless something else is explicitly specified in the diagrams. Thus, we do not deduce the communication architecture from the sequence diagrams but instead make it an option for the designer to explicitly specify the wanted architecture in the diagrams. The main rationale for introducing the three event semantics in Timed STAIRS is to be able to distinguish between reception and consumption of messages in order to specify time-constraints on black-box behavior as well as message consumption. Hence, the purpose of the three event semantics is quite different from [EMR97] where time and black-box behavior is not considered.

To consider not only positive traces, but also negative ones, has been suggested before. In [Hau95] the proposed methodology stated that specifying negative scenarios could be even more practical and powerful than only specifying the possible or mandatory ones. It was made clear that the MSC-92 standard [ITU93] was not sufficient to express the intention behind the scenarios and that the MSC documents had to be supplemented with informal statements about the intended interpretation of the set of traces expressed by the different MSCs.

The algebraic semantics of MSC-92 [ITU94] gave rise to a canonical logical expression restricted to the strict sequencing operator and a choice operator. When the MSC standard evolved with more advanced structuring mechanisms, the formal semantics as given in [ITU98] and [Ren98] was based on sets of traces, but it was still expressed in algebraic terms. The MSC approach to sequence diagram semantics is an interleaving semantics based on a fully compositional paradigm. The set of traces denoting the semantics of a message sequence chart can be calculated from its constituent parts based on definitions of the semantics of the structuring concepts as operators. This is very much the approach that we base our semantics on as we calculate our semantics of an interaction fragment from the semantics of its internal fragments. The notion of negative traces, and the explicit distinction between mandatory and potential behavior is beyond the MSC language and its semantics. The Eindhoven school of MSC researchers led by Sjouke Mauw concentrated mainly on establishing the formal properties of the logical systems used for defining the semantics, and also how this could be applied to make tools.

The need for describing also the intention behind the scenarios motivated the so-called “two-layer” approaches. In [CPRO95] they showed how MSC could be combined with languages for temporal logics such as CTL letting the scenarios constitute the atoms for the higher level of modal descriptions. With this one could describe that certain scenarios should appear or should never appear.

Damm and Harel brought this further through their augmented MSC language LSC (Live Sequence Charts) [DH99]. This may also be characterized as a two-layer approach as it takes the basic message sequence charts as starting point and add modal characteristics upon those. The modal expressiveness is strong in LSC since charts, locations, messages and conditions are orthogonally characterized as either mandatory or provisional. Since LSC also includes a notion of subchart, the combinatory complexity can be quite high. The “inline expressions” of MSC-96 (corresponding to combined fragments in UML 2.0) and MSC documents as in MSC-2000 [Hau01] (corresponds to classifier in UML 2.0) are, however, not included in LSC. Mandatory charts are called universal. Their interpretation is that provided their initial condition holds, these charts must happen. Mandatory as in LSC should not be confused with mandatory as in Timed STAIRS, since the latter only specifies traces that must be present in an implementation while the first specifies all allowed traces. Hence, mandatory as in Timed STAIRS does not distinguish between universal or existential interpretation, but rather gives a restriction on what behaviors that must be kept during a refinement. Provisional charts are called existential and they may happen if their initial condition holds. Through mandatory charts it is of course indirectly also possible to define scenarios that are forbidden or negative. Their semantics is said to be a conservative extension of the original MSC semantics, but their construction of the semantics is based on a two-stage procedure. The first stage defines a symbolic transition system from an LSC and from that a set of executions accepted by the LSC is produced. These executions represent traces where each basic element is a snapshot of a corresponding system.

The motivation behind LSC is explicitly to relate sequence diagrams to other system descriptions, typically defined with state machines. Harel has also been involved in the development of a tool-supported methodology that uses LSC as a way to prescribe systems as well as verifying the correspondence between manually described LSCs and State Machines [HM03].

Our approach is similar to LSC since it is basically interleaving. Timed STAIRS is essentially one-stage as the modal distinction between the positive and negative traces in principle is present in every fragment. The final modality results directly from the semantic compositions. With respect to language, we consider almost only what is UML 2.0, while LSC is a language extension of its own. LSC could in the future become a particular UML profile. Furthermore, our focus is on refinement of sequence diagrams as a means for system development and system validation. This means that in our approach the distinction between mandatory and provisional is captured through the interaction obligations.

The work by Krüger [Kru00] addresses similar concerns as the ones introduced in this article and covered by the LSC-approach of Harel. Just as with LSC MSCs can be given interpretations as existential or universal. The exact and negative interpretations are also introduced. Krüger also proposes notions of refinement for MSCs. Binding references, interface refinement, property refinement and structural refinement are refinement relations between MSCs at different level of abstraction. Narrowing as described in Timed STAIRS corresponds closely to property refinement in [Kru00] and detailing corresponds to interface refinement and structural refinement. However, Krüger does not distinguish between intended non-determinism and non-determinism as a result of under-specification in the refinement relation.

Although this paper presents Timed STAIRS in the setting of UML 2.0 sequence diagrams, the underlying principles apply just as well to MSC given that the MSC language is extended with an `xalt` construct similar to the one proposed above for UML 2.0. Timed STAIRS may also be adapted to support LSC. Timed STAIRS is complementary to software development processes based on use-cases, and classical object-oriented approaches such as the Unified Process [JBR99]. Timed STAIRS provides formal foundation for the basic incremental steps of such processes.

Acknowledgements. The research on which this paper reports has partly been carried out within the context of the IKT-2010 project SARDAS (15295/431) funded by the Research Council of Norway. We thank Mass Soldal Lund, Atle Refsdal, Ina Schieferdecker and Fredrik Seehusen for helpful feedback.

References

- [BS01] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.
- [CPRO95] P. Combes, S. Pickin, B. Renard, and F. Olsen. MSCs to express service requirements as properties on an SDL model: Application to service interaction detection. In *7th SDL Forum (SDL'95)*, pages 243–256. North-Holland, 1995.
- [DH99] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. In *Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, pages 293–311. Kluwer, 1999.

- [EMR97] A. Engels, S. Mauw, and M.A. Reniers. A hierarchy of communication models for message sequence charts. In *Formal Description Techniques and Protocol Specification, Testing and Verification*, pages 75–90. Chapman & Hall, 1997.
- [Hau95] Ø. Haugen. Using MSC-92 effectively. In *7th SDL Forum (SDL'95)*, pages 37–49. North-Holland, 1995.
- [Hau01] Ø. Haugen. MSC-2000 interaction diagrams for the new millennium. *Computer Networks*, 35:721–732, 2001.
- [HM03] D. Harel and R. Marelly. Specifying and executing behavioral requirements: The play-in/play-out approach. *Software and System Modeling*, 2:82–107, 2003.
- [HS03] Ø. Haugen and K. Stølen. STAIRS — Steps to analyze interactions with refinement semantics. In *Sixth International Conference on UML (UML'2003)*, number 2863 in Lecture Notes in Computer Science, pages 388–402. Springer, 2003.
- [ITU93] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1993.
- [ITU94] International Telecommunication Union. *Recommendation Z.120 Annex B: Algebraic Semantics of Message Sequence Charts*, 1994.
- [ITU98] International Telecommunication Union. *Recommendation Z.120 Annex B: Formal Semantics of Message Sequence Charts*, 1998.
- [ITU99] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1999.
- [JBR99] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.
- [Krü00] I. Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.
- [Lam93] L. Lamport. How to write a proof. *American Mathematical Monthly*, 102(7):600–608, August–September 1993.
- [OMG00] Object Management Group. *Unified Modeling Language, Version 1.4*, 2000.
- [OMG04] Object Management Group. *UML 2.0 Superstructure Specification*, document: ptc/04-10-02 edition, 2004.
- [Ren98] M. A. Reniers. *Message Sequence Chart: Syntax and Semantics*. PhD thesis, Eindhoven University of Technology, 1998.

10.A Extending STAIRS to Handle Gates

In the definition of a sequence diagram in Section 10.3 we do not take the notion of “gates” into consideration. This is just to keep the presentation simple. In the following we extend our approach to also handle gates.

10.A.1 Syntax

As before, a message is a triple (s, tr, re) of a signal s , a transmitter tr , and a receiver re , but contrary to earlier we allow gates as “receivers” and “transmitters”. Hence, the transmitters and receivers are now lifelines or gates, and not just lifelines as previously. A gate occurring in the position of a receiver is an output gate, and a gate occurring in the position of a transmitter is an input gate. However, the transmitter and the receiver of a message cannot both be gates, since

no message should go directly from an input gate to an output gate. \mathcal{L} and \mathcal{G} denote the set of all lifelines and gates, respectively. \mathcal{L} and \mathcal{G} are assumed to be disjoint.

Events are defined in exactly the same way as before, with the exception that we impose two constraints to make sure that events take place at lifelines only, i.e., not at gates. Formally, an event is a triple of kind, message and timestamp tag

$$(k, (s, tr, re), t) \in \mathcal{K} \times \mathcal{M} \times \mathcal{T}$$

such that

- $k = ! \Rightarrow tr \in \mathcal{L}$
- $k = \sim \vee k = ? \Rightarrow re \in \mathcal{L}$

The set of syntactically correct sequence diagrams \mathcal{D} is defined inductively in the same way as before with the following constraints on gates:

- The operands of alt, xalt and par cannot communicate directly via gates. Hence, the input ports of the first operand are disjoint from the output ports of the second operand, and the other way around.
- The operands of seq have disjoint sets of input gates and disjoint sets of output gates.
- If two operands of seq share a gate³ then the corresponding messages contain the same signal.

In addition we introduce an operator for gate substitution. If $d \in \mathcal{D}$ is a sequence diagram, $gates(d)$ its set of gates, $g \in gates(d)$ and h is a gate that does not occur in d then $d[g]_h \in \mathcal{D}$ is a sequence diagram with gates:

$$(gates(d) \setminus \{g\}) \cup \{h\}$$

10.A.2 Semantics

For any sequence diagram d , let the function gm be a set of maplets such that the following holds:

- $d \in \mathcal{E} \wedge tr.d, re.d \in \mathcal{L} \Rightarrow gm(d) = \emptyset$
- $d \in \mathcal{E} \wedge tr.d \in \mathcal{G} \Rightarrow gm(d) = \{tr.d \mapsto re.d\}$
- $d \in \mathcal{E} \wedge re.d \in \mathcal{G} \Rightarrow gm(d) = \{re.d \mapsto tr.d\}$
- $d = \text{neg } d' \Rightarrow gm(d) = gm(d')$
- $d = \text{loop } d' \Rightarrow gm(d) = gm(d')$

³A gate is shared if it is an input gate of one operand and an output gate of another operand. Thus, a shared gate is a way to represent the same as a connector in the graphical syntax of UML 2.0.

- $d = \text{assert } d' \Rightarrow gm(d) = gm(d')$
- $d = d' \text{ tc } C \Rightarrow gm(d) = gm(d')$
- $d = d_1 \text{ alt } d_2 \Rightarrow gm(d) = gm(d_1) \cup gm(d_2)$
- $d = d_1 \text{ xalt } d_2 \Rightarrow gm(d) = gm(d_1) \cup gm(d_2)$
- $d = d_1 \text{ par } d_2 \Rightarrow gm(d) = gm(d_1) \cup gm(d_2)$
- $d = \text{seq } [d_1, \dots, d_n] \Rightarrow$
 $gm(d) = \{(g \mapsto l) \in gm(d_i) \mid i \in [1 \dots n] \wedge$
 $\neg \exists l', i' : i \neq i' \wedge (g \mapsto l') \in gm(d_{i'})\}$
- $d = d'[^g_h] \Rightarrow \exists l \in L : gm(d) = (gm(d') \setminus \{g \mapsto l\}) \cup \{h \mapsto l\}$

We may think of $gm(d)$ as a function providing a matching lifeline for each gate in its domain. In the case of seq, the definition states that if there is a shared gate (i.e. a gate that exists in two operands), this gate should *not* be taken as a gate of the total diagram (see example below).

For any interaction obligation o and substitution $gm(d)$, we define $o \cdot gm(d)$ to denote the interaction obligation obtained from o by replacing any occurrence of any gate g for which $gm(d)$ is defined in any of o 's traces by $(gm(d))(g)$.

The semantics of a sequence diagram can then be defined as before with the exception that

- sequential composition is updated to

$$\begin{aligned} \llbracket \text{seq } [d] \rrbracket &\stackrel{\text{def}}{=} \llbracket d \rrbracket \\ \llbracket \text{seq } [D, d] \rrbracket &\stackrel{\text{def}}{=} \llbracket \text{seq } [D] \rrbracket \cdot gm(d) \succsim \llbracket d \rrbracket \cdot gm(\text{seq } [D]) \end{aligned} \quad (10.57)$$

with $O \cdot gm(d)$ defined as

$$O \cdot gm(d) \stackrel{\text{def}}{=} \{o \cdot gm(d) \mid o \in O\} \quad (10.58)$$

- substitution is defined as

$$\llbracket d[^g_h] \rrbracket \stackrel{\text{def}}{=} \llbracket d \rrbracket[^g_h] \quad (10.59)$$

$$O[^g_h] \stackrel{\text{def}}{=} \{o \cdot \{g \mapsto h\} \mid o \in O\} \quad (10.60)$$

10.A.3 Example

In the following, we demonstrate how the formal definitions work on a concrete example with gates. The definitions handle three-event, timed sequence diagrams, but in order to simplify the presentation and focus on how gates are handled, in this example we use the two-event, untimed version.

The sequence diagram d in Figure 10.12 has one output gate, o . Its semantics and gate matching function are defined as:

$$\llbracket d \rrbracket = \{\{\langle \langle \langle !, (m, L_1, o) \rangle \rangle \}, \emptyset\} gm(d) = \{o \mapsto L_1\}$$

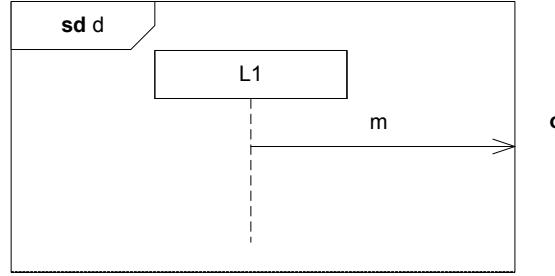


Figure 10.12: Sequence diagram with one output gate

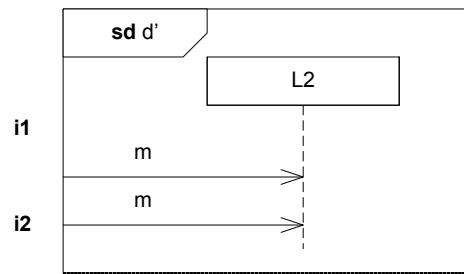


Figure 10.13: Sequence diagram with two input gates

The sequence diagram d' in Figure 10.13 has two input gates, i_1 and i_2 . Its semantics and gate matching function are defined as:

$$\llbracket d' \rrbracket = \{(\{\langle (?,(m,i_1,L_2)), (?,(m,i_2,L_2)) \rangle\}, \emptyset)\}$$

$$gm(d') = \{i_1 \mapsto L_2, i_2 \mapsto L_2\}$$

Two instances of d may be composed with d' as follows:

$$G = \text{seq} [d_{[c_1]}^o, d_{[c_2]}^o, d'^{[i_1]}_{[c_1]} d'^{[i_2]}_{[c_2]}]$$

This is illustrated in Figure 10.14.

The semantics of G may be calculated as follows:

$$\llbracket d_{[c_1]}^o \rrbracket = \{(\{\langle (!,(m,L_1,c_1)) \rangle\}, \emptyset)\}$$

$$gm(d_{[c_1]}^o) = \{c_1 \mapsto L_1\}$$

$$\llbracket d_{[c_2]}^o \rrbracket = \{(\{\langle (!,(m,L_1,c_2)) \rangle\}, \emptyset)\}$$

$$gm(d_{[c_2]}^o) = \{c_2 \mapsto L_1\}$$

$$\llbracket \text{seq} [d_{[c_1]}^o, d_{[c_2]}^o] \rrbracket = \{(\{\langle (!,(m,L_1,c_1)), (!,(m,L_1,c_2)) \rangle\}, \emptyset)\}$$

$$gm(\text{seq} [d_{[c_1]}^o, d_{[c_2]}^o]) = \{c_1 \mapsto L_1, c_2 \mapsto L_1\}$$

$$\llbracket d'^{[i_1]}_{[c_1]} d'^{[i_2]}_{[c_2]} \rrbracket = \{(\{\langle (?,(m,c_1,L_2)), (?,(m,c_2,L_2)) \rangle\}, \emptyset)\}$$

$$gm(d'^{[i_1]}_{[c_1]} d'^{[i_2]}_{[c_2]}) = \{c_1 \mapsto L_2, c_2 \mapsto L_2\}$$

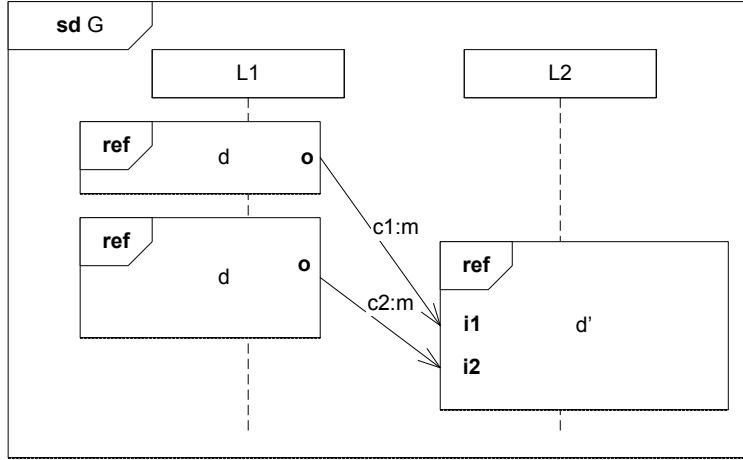


Figure 10.14: Composed sequence diagram

And finally:

$$\begin{aligned} \llbracket G \rrbracket &= \{(\{\langle\langle !, (m, L_1, L_2)), (!, (m, L_1, L_2))\rangle\}, \emptyset)\} \succsim \\ &\quad \{(\{\langle\langle ?, (m, L_1, L_2)), (?, (m, L_1, L_2))\rangle\}, \emptyset)\} \\ &= \{(\{\langle\langle !, (m, L_1, L_2)), (!, (m, L_1, L_2)), (?, (m, L_1, L_2)), (?, (m, L_1, L_2))\rangle, \\ &\quad \langle !, (m, L_1, L_2)), (?, (m, L_1, L_2)), (!, (m, L_1, L_2)), (?, (m, L_1, L_2))\rangle\}, \emptyset)\} \end{aligned}$$

The gate matching function of G is the empty set, since both of the actual gates (c_1 and c_2) have a mapping for both $\text{seq } [d_{c_1}^o, d_{c_2}^o]$ and $d'_{c_1}^{i_1} [d'_{c_2}^{i_2}]$:

$$gm(G) = \emptyset$$

This means that G has no gates, as can be seen from Figure 10.14.

10.B Trace Completeness

In section 10.3.2 we formulated a number of syntactical well-formedness criteria for the sequence diagrams considered in this paper. Criteria (10.3) and (10.4) required that for all diagrams consisting of more than one event, a message should be complete if both the transmitter and the receiver lifelines are present in the diagram. In this section we formulate a corresponding semantic notion of trace completeness. We also demonstrate how trace completeness follows for all syntactically correct sequence diagrams, a fact that will be used when proving associativity of seq and par in Section 10.C.

In order to increase the readability of the following definitions and later proofs, we use the notation $\#!m \in h$ to denote the number of send-events with respect to the message m in h , and similarly for receive and consume events. Formally:

$$\# * m \in h \stackrel{\text{def}}{=} \#\{h[i] \mid i \in [1..h] \wedge k.h[i] = * \wedge m.h[i] = m\} \quad (10.61)$$

where $*$ is one of $!$, \sim , or $?$.

We now introduce the notion of trace completeness. For a trace h to be complete, we require that

1. if h contains both send and receive events of a message m , it contains equally many. Formally:

$$(\#!m \in h) > 0 \wedge (\#\sim m \in h) > 0 \Rightarrow (\#!m \in h) = (\#\sim m \in h) \quad (10.62)$$

2. if h contains more than one event, it contains equally many receive and consume events for all messages m . Formally:

$$\#h > 1 \Rightarrow (\#\sim m \in h) = (\#?m \in h) \quad (10.63)$$

We now define $\text{traces}(d)$ to yield all positive and negative traces in the interaction obligations in $\llbracket d \rrbracket$:

$$\text{traces}(d) = \{h \in \mathcal{H} \mid \exists s \in \text{tracesets}(d) : h \in s\} \quad (10.64)$$

where $\text{tracesets}(d)$ denotes all positive and negative trace-sets in the interaction obligations in $\llbracket d \rrbracket$, formally defined by:

$$\text{tracesets}(d) = \{s \in \mathbb{P}(\mathcal{H}) \mid \exists s' \in \mathbb{P}(\mathcal{H}) : (s, s') \in \llbracket d \rrbracket \vee (s', s) \in \llbracket d \rrbracket\} \quad (10.65)$$

The following two observations relate trace completeness to the syntactical well-formedness criteria from Section 10.3.2.

Observation 1 Let d be a syntactically correct sequence diagram. From the syntactical well-formedness criteria (10.1)–(10.4) and the definition of $\llbracket d \rrbracket$, it follows that all traces in $\text{traces}(d)$ are complete as defined by definitions (10.62) and (10.63).

Observation 2 Let d_1 and d_2 be syntactically correct sequence diagrams such that also $d_1 \text{ par } d_2$ and $\text{seq } [d_1, d_2]$ are syntactically correct. From the syntactical well-formedness criteria (10.1)–(10.4) and the definition of $\llbracket d \rrbracket$, it follows that for all traces $h_1 \in \text{traces}(d_1)$ and $h_2 \in \text{traces}(d_2)$, their combination is complete as defined by definitions (10.62) and (10.63), i.e. for all messages m :

1. $((\#!m \in h_1) + (\#!m \in h_2)) > 0 \wedge ((\#\sim m \in h_1) + (\#\sim m \in h_2)) > 0$
 $\Rightarrow ((\#!m \in h_1) + (\#!m \in h_2)) = ((\#\sim m \in h_1) + (\#\sim m \in h_2))$
2. $(\#h_1 + \#h_2) > 1$
 $\Rightarrow ((\#\sim m \in h_1) + (\#\sim m \in h_2)) = ((\#?m \in h_1) + (\#?m \in h_2))$

10.C Associativity, Commutativity and Distributivity

In this section we present several lemmas and theorems of associativity, commutativity and distributivity for various operators defined and used in this paper. In particular, we prove that `alt`, `xalt` and `par` are associative and commutative, and that `seq` is associative.

First, in Section 10.C.1 we prove some helpful lemmas regarding well-formedness of traces. Section 10.C.2 provides the necessary lemmas with respect to trace sets, while Sections 10.C.3 and 10.C.4 contain lemmas on interaction obligations and sets of interaction obligations, respectively. Finally, Section 10.C.5 provides the associativity and commutativity theorems for the sequence diagram operators.

Many of the proofs in these and the following sections are structured in a way similar to that of [Lam93], where the hierarchical structure is used to demonstrate how each proof step is justified.

10.C.1 Lemmas on Well-Formedness

First, we define $h_1 \lhd h_2$ to denote the fact that h_1 is a subtrace (possibly non-continuous) of h_2 . Formally:

$$h_1 \lhd h_2 \stackrel{\text{def}}{=} \exists p \in \{1, 2\}^\infty : \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h_2)) = h_1 \quad (10.66)$$

Lemma 1

- A : 1. $h_1 \lhd h_2$
- 2. h_1 violates criteria (10.6), i.e.
 $\neg(\forall i, j \in [1.. \#h_1] : i < j \Rightarrow r.h_1[i] \leq r.h_1[j])$
- P : h_2 violates criteria (10.6), i.e.
 $\neg(\forall i, j \in [1.. \#h_2] : i < j \Rightarrow r.h_2[i] \leq r.h_2[j])$
- $\langle 1 \rangle 1$. $\exists i, j \in [1.. \#h_1] : i < j \wedge r.h_1[i] > r.h_1[j]$
P : Assumption 2 and \neg -rules.
- $\langle 1 \rangle 2$. $\exists i', j' \in [1.. \#h_2] : i' < j' \wedge r.h_2[i'] > r.h_2[j']$
P : $\langle 1 \rangle 1$ and assumption 1.
- $\langle 1 \rangle 3$. $\neg(\forall i', j' \in [1.. \#h_2] : i' < j' \Rightarrow r.h_2[i'] \leq r.h_2[j'])$
P : $\langle 1 \rangle 2$ and \neg -rules.
- $\langle 1 \rangle 4$. Q.E.D.

□

Lemma 2

- A : 1. $h_1 \lhd h_2$
- 2. h_1 violates criteria (10.7), i.e.
 $\neg(\forall i, j \in [1.. \#h_1] : i \neq j \Rightarrow h_1[i] \neq h_1[j])$
- P : h_2 violates criteria (10.7), i.e.
 $\neg(\forall i, j \in [1.. \#h_2] : i \neq j \Rightarrow h_2[i] \neq h_2[j])$
- $\langle 1 \rangle 1$. $\exists i, j \in [1.. \#h_1] : i \neq j \wedge h_1[i] = h_1[j]$
P : Assumption 2 and \neg -rules.
- $\langle 1 \rangle 2$. $\exists i', j' \in [1.. \#h_2] : i' \neq j' \wedge h_2[i'] = h_2[j']$
P : $\langle 1 \rangle 1$ and assumption 1.

$\langle 1 \rangle 3. \neg(\forall i', j' \in [1..h_2] : i' \neq j' \Rightarrow h_2[i'] \neq h_2[j'])$

P : $\langle 1 \rangle 2$ and \neg -rules.

$\langle 1 \rangle 4. \text{Q.E.D.}$

□

10.C.2 Lemmas on Trace Sets

Lemma 3 *Commutativity of parallel execution*

P : $s_1 \parallel s_2 = s_2 \parallel s_1$

$$\begin{aligned} P &: \\ s_1 \parallel s_2 &= \{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty : \\ &\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1 \wedge \\ &\quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in s_2\} \quad \text{by definition (10.12)} \\ &= \{h \in \mathcal{H} \mid \exists p \in \{2, 1\}^\infty : \\ &\quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1 \wedge \\ &\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_2\} \quad \text{as the symbols in } p \text{ are arbitrary} \\ &= \{h \in \mathcal{H} \mid \exists p \in \{2, 1\}^\infty : \\ &\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_2 \wedge \\ &\quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1\} \quad \text{by the commutativity of } \wedge \\ &= \{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty : \\ &\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_2 \wedge \\ &\quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1\} \quad \text{since } \{1, 2\}^\infty = \{2, 1\}^\infty \\ &= s_2 \parallel s_1 \quad \text{by definition (10.12)} \end{aligned}$$

Lemma 4 *Associativity of parallel execution for non-empty trace-sets*

A : 1. $(s_1 \parallel s_2) \neq \emptyset$

2. $(s_2 \parallel s_3) \neq \emptyset$

P : $(s_1 \parallel s_2) \parallel s_3 = s_1 \parallel (s_2 \parallel s_3)$

P :

The two sides of the equation reduce to the same formula.

Left side:

$$\begin{aligned} &(s_1 \parallel s_2) \parallel s_3 \\ &= \{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty : \\ &\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in (s_1 \parallel s_2) \wedge \\ &\quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in s_3\} \quad \text{by definition (10.12)} \\ &= \{h \in \mathcal{H} \mid \exists p \in \{1, 2, 3\}^\infty : \\ &\quad \pi_2((\{1, 2\} \times [\mathcal{E}]) \oplus (p, h)) \in (s_1 \parallel s_2) \wedge \\ &\quad \pi_2((\{3\} \times [\mathcal{E}]) \oplus (p, h)) \in s_3\} \quad \text{as the symbols in } p \text{ are arbitrary} \\ &= \{h \in \mathcal{H} \mid \exists p \in \{1, 2, 3\}^\infty : \\ &\quad \pi_2((\{1, 2\} \times [\mathcal{E}]) \oplus (p, h)) \in \\ &\quad \{h' \in \mathcal{H} \mid \exists p' \in \{1, 2\}^\infty : \\ &\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p', h')) \in s_1 \wedge \\ &\quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p', h')) \in s_2\} \wedge \\ &\quad \pi_2((\{3\} \times [\mathcal{E}]) \oplus (p, h)) \in s_3\} \quad \text{by definition (10.12)} \end{aligned}$$

$$\begin{aligned}
&= \{h \in \mathcal{H} \mid \exists p \in \{1, 2, 3\}^\infty, p' \in \{1, 2\}^\infty : \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p', \pi_2((\{1, 2\} \times [\mathcal{E}]) \oplus (p, h)))) \in s_1 \wedge \\
&\quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p', \pi_2((\{1, 2\} \times [\mathcal{E}]) \oplus (p, h)))) \in s_2 \wedge \text{ by definition (10.12)} \\
&\quad \pi_2((\{3\} \times [\mathcal{E}]) \oplus (p, h)) \in s_3\} \quad \text{and assumption 1} \\
&= \{h \in \mathcal{H} \mid \exists p \in \{1, 2, 3\}^\infty : \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1 \wedge \\
&\quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in s_2 \wedge \\
&\quad \pi_2((\{3\} \times [\mathcal{E}]) \oplus (p, h)) \in s_3\} \quad \text{constructing } p \text{ with } p' \text{ as a subsequence}
\end{aligned}$$

Right side:

$$\begin{aligned}
&s_1 \parallel (s_2 \parallel s_3) \\
&= \{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty : \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1 \wedge \\
&\quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in (s_2 \parallel s_3)\} \quad \text{by definition (10.12)} \\
&= \{h \in \mathcal{H} \mid \exists p \in \{1, 2, 3\}^\infty : \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1 \wedge \\
&\quad \pi_2((\{2, 3\} \times [\mathcal{E}]) \oplus (p, h)) \in (s_2 \parallel s_3)\} \quad \text{as the symbols in } p \text{ are arbitrary} \\
&= \{h \in \mathcal{H} \mid \exists p \in \{1, 2, 3\}^\infty : \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1 \wedge \\
&\quad \pi_2((\{2, 3\} \times [\mathcal{E}]) \oplus (p, h)) \in \\
&\quad \quad \{h' \in \mathcal{H} \mid \exists p' \in \{2, 3\}^\infty : \\
&\quad \quad \quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p', h')) \in s_2 \wedge \\
&\quad \quad \quad \pi_2((\{3\} \times [\mathcal{E}]) \oplus (p', h')) \in s_3\}\} \quad \text{by definition (10.12)} \\
&= \{h \in \mathcal{H} \mid \exists p \in \{1, 2, 3\}^\infty, p' \in \{2, 3\}^\infty : \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1 \wedge \\
&\quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p', \pi_2((\{2, 3\} \times [\mathcal{E}]) \oplus (p, h)))) \in s_2 \wedge \text{ by definition (10.12)} \\
&\quad \pi_2((\{3\} \times [\mathcal{E}]) \oplus (p', \pi_2((\{2, 3\} \times [\mathcal{E}]) \oplus (p, h)))) \in s_3\} \quad \text{and assumption 2} \\
&= \{h \in \mathcal{H} \mid \exists p \in \{1, 2, 3\}^\infty : \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1 \wedge \\
&\quad \pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in s_2 \wedge \\
&\quad \pi_2((\{3\} \times [\mathcal{E}]) \oplus (p, h)) \in s_3\} \quad \text{constructing } p \text{ with } p' \text{ as a subsequence}
\end{aligned}$$

□

Lemma 5 (To be used when proving associativity of parallel execution in lemma 7)

Let d_1 , d_2 and d_3 be syntactically well-formed sequence diagrams such that also $d_1 \text{ par } d_2$, $d_2 \text{ par } d_3$ and $(d_1 \text{ par } d_2) \text{ par } d_3$ are syntactically well-formed.

$$\begin{aligned}
A &: 1. s_1 \in \text{tracesets}(d_1) \\
&2. s_2 \in \text{tracesets}(d_2) \\
&3. s_3 \in \text{tracesets}(d_3) \\
P &: s_2 \parallel s_3 = \emptyset \Rightarrow (s_1 \parallel s_2) \parallel s_3 = \emptyset \\
\langle 1 \rangle A &: s_2 \parallel s_3 = \emptyset \\
P &: (s_1 \parallel s_2) \parallel s_3 = \emptyset
\end{aligned}$$

$\langle 2 \rangle 1.$ C : $s_1 = \emptyset, s_2 = \emptyset, s_3 = \emptyset$ or $s_1 \parallel s_2 = \emptyset$

P : $(s_1 \parallel s_2) \parallel s_3 = \emptyset$ by definition (10.12) of \parallel .

$\langle 2 \rangle 2.$ C : $s_1 \neq \emptyset, s_2 \neq \emptyset, s_3 \neq \emptyset$ and $s_1 \parallel s_2 \neq \emptyset$

$\langle 3 \rangle 1.$ $h \notin \mathcal{H}$ for arbitrary

$h \in \{h \in [\mathcal{E}]^\omega \mid \exists p \in \{1, 2, 3\}^\infty :$

$\pi_2((\{1, 2\} \times [\mathcal{E}]) \oplus (p, h)) \in (s_1 \parallel s_2) \wedge$

$\pi_2((\{3\} \times [\mathcal{E}]) \oplus (p, h)) \in s_3\}$

$\langle 4 \rangle 1.$ Choose $h_{12} \in s_1 \parallel s_2, h_3 \in s_3$ and $p \in \{1, 2, 3\}^\infty$ such that

$\pi_2((\{1, 2\} \times [\mathcal{E}]) \oplus (p, h)) = h_{12}$ and

$\pi_2((\{3\} \times [\mathcal{E}]) \oplus (p, h)) = h_3$

P : $\langle 2 \rangle 2$ and $\langle 3 \rangle 1.$

$\langle 4 \rangle 2.$ Choose $h_1 \in s_1, h_2 \in s_2$ and $p' \in \{1, 2\}^\infty$ such that

$\pi_2((\{1\} \times [\mathcal{E}]) \oplus (p', h_1)) = h_1$ and

$\pi_2((\{2\} \times [\mathcal{E}]) \oplus (p', h_1)) = h_2$

P : $\langle 4 \rangle 1$ and definition (10.12) of \parallel .

$\langle 4 \rangle 3.$ $\{h \in \mathcal{H} \mid \exists p \in \{2, 3\}^\infty :$

$\pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in s_2 \wedge$

$\pi_2((\{3\} \times [\mathcal{E}]) \oplus (p, h)) \in s_3\} = \emptyset$

P : $\langle 1 \rangle 1$ and definition (10.12) of \parallel .

$\langle 4 \rangle 4.$ Choose $h_{23} \lhd h$ such that

$h_{23} \in \{h \in [\mathcal{E}]^\omega \mid \exists p'' \in \{2, 3\}^\infty :$

$\pi_2((\{2\} \times [\mathcal{E}]) \oplus (p'', h_{23})) = h_2 \wedge$

$\pi_2((\{3\} \times [\mathcal{E}]) \oplus (p'', h_{23})) = h_3\}$

$\langle 5 \rangle 1.$ $h_2 \lhd h$

$\langle 6 \rangle 1.$ $h_2 \lhd h_{12}$

P : $\langle 4 \rangle 2$ and definition (10.66) of \lhd .

$\langle 6 \rangle 2.$ $h_{12} \lhd h$

P : $\langle 4 \rangle 1$ and definition (10.66) of \lhd .

$\langle 6 \rangle 3.$ Q.E.D.

P : $\langle 6 \rangle 1, \langle 6 \rangle 2$ and definition (10.66) of \lhd .

$\langle 5 \rangle 2.$ $h_3 \lhd h$

P : $\langle 4 \rangle 1$ and definition (10.66) of \lhd .

$\langle 5 \rangle 3.$ Q.E.D.

P : $\langle 5 \rangle 1, \langle 5 \rangle 2, \langle 4 \rangle 4$ and definition (10.66) of \lhd .

$\langle 4 \rangle 5.$ $h_{23} \notin \mathcal{H}$

P : $\langle 4 \rangle 1, \langle 4 \rangle 2, \langle 4 \rangle 3$ and $\langle 4 \rangle 4.$

$\langle 4 \rangle 6.$ $h \notin \mathcal{H}$

$\langle 5 \rangle 1.$ C : h_{23} violates criteria (10.6), i.e.

$\neg(\forall i, j \in [1.. \#h_{23}] : i < j \Rightarrow r.h_{23}[i] \leq r.h_{23}[j])$

P : h violates criteria (10.6) by $\langle 4 \rangle 4$ and lemma 1 on well-formedness.

$\langle 5 \rangle 2.$ C : h_{23} violates criteria (10.7), i.e.

$\neg(\forall i, j \in [1.. \#h_{23}] : i \neq j \Rightarrow h_{23}[i] \neq h_{23}[j])$

P : h violates criteria (10.7) by $\langle 4 \rangle 4$ and lemma 2 on well-formedness.

$\langle 5 \rangle 3.$ C : h_{23} violates criteria (10.8), i.e.

- $\neg(\forall l \in \mathcal{L} : \#e.l \circledcirc h_{23} = \infty \Rightarrow \forall t \in \mathbb{R} : \exists i \in \mathbb{N} : r.(e.l \circledcirc h_{23})[i] > t)$
 $\langle 6 \rangle 1. \forall l \in \mathcal{L} : \#e.l \circledcirc h_2 = \infty \Rightarrow \forall t \in \mathbb{R} : \exists i \in \mathbb{N} : r.(e.l \circledcirc h_2)[i] > t$
 P : $\langle 4 \rangle 2$ and assumption 2.
 $\langle 6 \rangle 2. \forall l \in \mathcal{L} : \#e.l \circledcirc h_3 = \infty \Rightarrow \forall t \in \mathbb{R} : \exists i \in \mathbb{N} : r.(e.l \circledcirc h_3)[i] > t$
 P : $\langle 4 \rangle 1$ and assumption 3.
 $\langle 6 \rangle 3. \forall l \in \mathcal{L} : \#e.l \circledcirc h_{23} = \infty \Rightarrow \forall t \in \mathbb{R} : \exists i \in \mathbb{N} : r.(e.l \circledcirc h_{23})[i] > t$
 P : $\langle 6 \rangle 1, \langle 6 \rangle 2$ and $\langle 4 \rangle 4$.
 $\langle 6 \rangle 4. \text{Q.E.D.}$
 P : Case impossible by $\langle 6 \rangle 3$.
 $\langle 5 \rangle 4. C : h_{23} \text{ violates criteria (10.10), i.e.}$
 $\neg(\forall i \in [1.. \#h_{23}] : k.h_{23}[i] = ! \Rightarrow \#(\{!\} \times \{m.h_{23}[i]\} \times U) \circledcirc h_{23}|_i > \#(\{\sim\} \times \{m.h_{23}[i]\} \times U) \circledcirc h_{23}|_i)$
 $\langle 6 \rangle 1. \text{Choose } i \in [1.. \#h_{23}] \text{ such that } k.h_{23}[i] = ! \text{ and}$
 $\#(\{!\} \times \{m.h_{23}[i]\} \times U) \circledcirc h_{23}|_i \leq \#(\{\sim\} \times \{m.h_{23}[i]\} \times U) \circledcirc h_{23}|_i$
 P : $\langle 5 \rangle 4$ and \neg -rules.
 $\langle 6 \rangle 2. L : m = m.h_{23}[i]$
 P : $\langle 6 \rangle 1$.
 $\langle 6 \rangle 3. (\#\{!m \in h_2\} + \#\{!m \in h_3\}) > 0$
 P : $\langle 4 \rangle 4, \langle 6 \rangle 1$ and $\langle 6 \rangle 2$.
 $\langle 6 \rangle 4. (\#\{\sim m \in h_2\} + \#\{\sim m \in h_3\}) > 0$
 P : $\langle 4 \rangle 4, \langle 6 \rangle 1$ and $\langle 6 \rangle 2$.
 $\langle 6 \rangle 5. (\#\{!m \in h_2\} + \#\{!m \in h_3\}) = (\#\{\sim m \in h_2\} + \#\{\sim m \in h_3\})$
 P : $\langle 6 \rangle 3, \langle 6 \rangle 4, \langle 4 \rangle 1, \langle 4 \rangle 2$, assumptions 2 and 3 and observation 2, part 2.
 $\langle 6 \rangle 6. (\#\{!m \in h\}) = (\#\{\sim m \in h\})$
 P : $\langle 6 \rangle 3, \langle 6 \rangle 4, \langle 4 \rangle 1, \langle 4 \rangle 2$ and observation 1.
 $\langle 6 \rangle 7. (\#\{!m \in h_1\}) = (\#\{\sim m \in h_1\})$
 P : $\langle 6 \rangle 5, \langle 6 \rangle 6, \langle 4 \rangle 1$ and $\langle 4 \rangle 2$.
 $\langle 6 \rangle 8. \exists i' \in [1.. \#h] : k.h[i'] = ! \wedge \#(\{!\} \times \{m.h[i']\} \times U) \circledcirc h|_{i'} \leq \#(\{\sim\} \times \{m.h[i']\} \times U) \circledcirc h|_{i'}$
 P : $\langle 6 \rangle 1, \langle 6 \rangle 7$ and $\langle 4 \rangle 4$.
 $\langle 6 \rangle 9. \text{Q.E.D.}$
 P : h violates criteria (10.10) by $\langle 6 \rangle 8$.
 $\langle 5 \rangle 5. C : h_{23} \text{ violates criteria (10.11), i.e.}$
 $\neg(\forall i \in [1.. \#h] : k.h_{23}[i] = \sim \Rightarrow \#(\{\sim\} \times \{m.h_{23}[i]\} \times U) \circledcirc h_{23}|_i > \#(\{?\} \times \{m.h_{23}[i]\} \times U) \circledcirc h_{23}|_i)$
 $\langle 6 \rangle 1. \text{Choose } i \in [1.. \#h_{23}] \text{ such that } k.h_{23}[i] = \sim \text{ and}$
 $\#(\{\sim\} \times \{m.h_{23}[i]\} \times U) \circledcirc h_{23}|_i \leq$

- #($\{?\} \times \{m.h_{23}[i]\} \times U) \circledcirc h_{23}|_i$
- P : $\langle 5 \rangle 5$ and \neg -rules.
- $\langle 6 \rangle 2.$ L : $m = m.h_{23}[i]$
- P : $\langle 6 \rangle 1.$
- $\langle 6 \rangle 3.$ $(\# \sim m \in h_2) + (\# \sim m \in h_3) > 0$
- P : $\langle 4 \rangle 4, \langle 6 \rangle 1$ and $\langle 6 \rangle 2.$
- $\langle 6 \rangle 4.$ $(\#?m \in h_2) + (\#?m \in h_3) > 0$
- P : $\langle 4 \rangle 4, \langle 6 \rangle 1$ and $\langle 6 \rangle 2.$
- $\langle 6 \rangle 5.$ $(\# \sim m \in h_2) + (\# \sim m \in h_3) = (\#?m \in h_2) + (\#?m \in h_3)$
- P : $\langle 6 \rangle 3, \langle 6 \rangle 4, \langle 4 \rangle 1, \langle 4 \rangle 2$, assumptions 2 and 3 and observation 2, part 1.
- $\langle 6 \rangle 6.$ $(\# \sim m \in h) = (\#?m \in h)$
- P : $\langle 6 \rangle 3, \langle 6 \rangle 4, \langle 4 \rangle 1, \langle 4 \rangle 2$ and observation 1.
- $\langle 6 \rangle 7.$ $(\# \sim m \in h_1) = (\#?m \in h_1)$
- P : $\langle 6 \rangle 5, \langle 6 \rangle 6, \langle 4 \rangle 1$ and $\langle 4 \rangle 2.$
- $\langle 6 \rangle 8.$ $\exists i' \in [1.. \#h] : k.h[i'] = \sim \wedge$
- $\#(\{\sim\} \times \{m.h[i']\} \times U) \circledcirc h|_{i'} \leq \#(\{?\} \times \{m.h[i']\} \times U) \circledcirc h|_{i'}$
- P : $\langle 6 \rangle 1, \langle 6 \rangle 7$ and $\langle 4 \rangle 4.$
- $\langle 6 \rangle 9.$ Q.E.D.
- P : h violates criteria (10.11) by $\langle 6 \rangle 8.$
- $\langle 5 \rangle 6.$ Q.E.D.
- P : The cases are exhaustive by $\langle 4 \rangle 5$ and definition of $\mathcal{H}.$
- $\langle 4 \rangle 7.$ Q.E.D.
- P : $\langle 4 \rangle 6.$
- $\langle 3 \rangle 2.$ $\{h \in \mathcal{H} \mid \exists p \in \{12, 3\}^\infty :$
- $\pi_2((\{12\} \times [\mathcal{E}]) \oplus (p, h)) \in (s_1 \parallel s_2) \wedge$
- $\pi_2((\{3\} \times [\mathcal{E}]) \oplus (p, h)) \in s_3\} = \emptyset$
- P : $\langle 3 \rangle 1.$
- $\langle 3 \rangle 3.$ Q.E.D.
- P : $(s_1 \parallel s_2) \parallel s_3 = \emptyset$ by $\langle 3 \rangle 2$ and definition (10.12) of $\parallel.$
- $\langle 2 \rangle 3.$ Q.E.D.
- P : The cases are exhaustive.
- $\langle 1 \rangle 2.$ Q.E.D.
- P : \Rightarrow -rule.

□

Lemma 6 (To be used when proving associativity of parallel execution in lemma 7)

Let d_1 , d_2 and d_3 be syntactically well-formed sequence diagrams such that also $d_1 \text{ par } d_2$, $d_2 \text{ par } d_3$ and $d_1 \text{ par } (d_2 \text{ par } d_3)$ are syntactically well-formed.

- A : 1. $s_1 \in \text{tracesets}(d_1)$
2. $s_2 \in \text{tracesets}(d_2)$
3. $s_3 \in \text{tracesets}(d_3)$
- P : $s_1 \parallel s_2 = \emptyset \Rightarrow s_1 \parallel (s_2 \parallel s_3) = \emptyset$

$\langle 1 \rangle 1.$ A : $s_1 \parallel s_2 = \emptyset$
 P : $s_1 \parallel (s_2 \parallel s_3) = \emptyset$
 $\langle 2 \rangle 1.$ $s_1 \parallel (s_2 \parallel s_3) = (s_2 \parallel s_3) \parallel s_1$
 P : Lemma 3 (commutativity of \parallel).
 $\langle 2 \rangle 2.$ $(s_2 \parallel s_3) \parallel s_1 = (s_3 \parallel s_2) \parallel s_1$
 P : Lemma 3 (commutativity of \parallel).
 $\langle 2 \rangle 3.$ $s_2 \parallel s_1 = \emptyset$
 P : $\langle 1 \rangle 1$ and lemma 3 (commutativity of \parallel).
 $\langle 2 \rangle 4.$ $(s_3 \parallel s_2) \parallel s_1 = \emptyset$
 P : $\langle 2 \rangle 3$ and lemma 5.
 $\langle 2 \rangle 5.$ Q.E.D.
 P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and $\langle 2 \rangle 4$.
 $\langle 1 \rangle 2.$ Q.E.D.
 P : \Rightarrow -rule.

□

Lemma 7 *Associativity of parallel execution*

Let d_1 , d_2 and d_3 be syntactically well-formed sequence diagrams such that also $d_1 \text{ par } d_2$, $d_2 \text{ par } d_3$, $(d_1 \text{ par } d_2) \text{ par } d_3$ and $d_1 \text{ par } (d_2 \text{ par } d_3)$ are syntactically well-formed.

A : 1. $s_1 \in \text{tracesets}(d_1)$
 2. $s_2 \in \text{tracesets}(d_2)$
 3. $s_3 \in \text{tracesets}(d_3)$
 P : $(s_1 \parallel s_2) \parallel s_3 = s_1 \parallel (s_2 \parallel s_3)$
 $\langle 1 \rangle 1.$ C : $s_1 \parallel s_2 = \emptyset$
 $\langle 2 \rangle 1.$ $(s_1 \parallel s_2) \parallel s_3 = \emptyset$
 P : Definition (10.12) of \parallel .
 $\langle 2 \rangle 2.$ $s_1 \parallel (s_2 \parallel s_3) = \emptyset$
 P : $\langle 1 \rangle 1$ and lemma 6.
 $\langle 2 \rangle 3.$ Q.E.D.
 $\langle 1 \rangle 2.$ C : $s_2 \parallel s_3 = \emptyset$
 $\langle 2 \rangle 1.$ $s_1 \parallel (s_2 \parallel s_3) = \emptyset$
 P : Definition (10.12) of \parallel .
 $\langle 2 \rangle 2.$ $(s_1 \parallel s_2) \parallel s_3 = \emptyset$
 P : $\langle 1 \rangle 2$ and lemma 5.
 $\langle 2 \rangle 3.$ Q.E.D.
 $\langle 1 \rangle 3.$ C : $s_1 \parallel s_2 \neq \emptyset \wedge s_2 \parallel s_3 \neq \emptyset$
 P : $(s_1 \parallel s_2) \parallel s_3 = s_1 \parallel (s_2 \parallel s_3)$ by lemma 4.
 $\langle 1 \rangle 4.$ Q.E.D.
 P : The cases are exhaustive.

□

Lemma 8 *Associativity of weak sequencing for non-empty trace-sets*

$$A : \begin{aligned} & 1. s_1 \succsim s_2 \neq \emptyset \\ & 2. s_2 \succsim s_3 \neq \emptyset \end{aligned}$$

$$P : (s_1 \succsim s_2) \succsim s_3 = s_1 \succsim (s_2 \succsim s_3)$$

P :

The two sides of the equation reduce to the same formula.

Left side:

$$\begin{aligned} & (s_1 \succsim s_2) \succsim s_3 \\ &= \{h \in \mathcal{H} \mid \exists h_{12} \in (s_1 \succsim s_2), h_3 \in s_3 : \forall l \in \mathcal{L} : \\ & \quad e.l \circledcirc h = e.l \circledcirc h_{12} \cap e.l \circledcirc h_3\} \quad \text{by definition (10.13)} \\ &= \{h \in \mathcal{H} \mid \\ & \quad \exists h_{12} \in \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \forall l' \in \mathcal{L} : \\ & \quad \quad e.l' \circledcirc h = e.l' \circledcirc h_1 \cap e.l' \circledcirc h_2\} \wedge \\ & \quad \exists h_3 \in s_3 : \forall l \in \mathcal{L} : \\ & \quad \quad e.l \circledcirc h = e.l \circledcirc h_{12} \cap e.l \circledcirc h_3\} \quad \text{by definition (10.13)} \\ &= \{h \in \mathcal{H} \mid \\ & \quad \exists h_{12} \in \mathcal{H}, h_1 \in s_1, h_2 \in s_2 : \\ & \quad (\forall l' \in \mathcal{L} : e.l' \circledcirc h_{12} = e.l' \circledcirc h_1 \cap e.l' \circledcirc h_2) \wedge \\ & \quad \exists h_3 \in s_3 : \forall l \in \mathcal{L} : \\ & \quad \quad e.l \circledcirc h = e.l \circledcirc h_{12} \cap e.l \circledcirc h_3\} \quad \text{by definition (10.13)} \text{ and assumption 1} \\ &= \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2, h_3 \in s_3 : \forall l \in \mathcal{L} : \quad \text{using } e.l \circledcirc h_{12} = e.l \circledcirc h_1 \cap e.l \circledcirc h_2 \\ & \quad e.l \circledcirc h = e.l \circledcirc h_1 \cap e.l \circledcirc h_2 \cap e.l \circledcirc h_3\} \quad \text{for all } l \in \mathcal{L} \end{aligned}$$

Right side:

$$\begin{aligned} & s_1 \succsim (s_2 \succsim s_3) \\ &= \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_{23} \in (s_2 \succsim s_3) : \forall l \in \mathcal{L} : \\ & \quad e.l \circledcirc h = e.l \circledcirc h_1 \cap e.l \circledcirc h_{23}\} \quad \text{by definition (10.13)} \\ &= \{h \in \mathcal{H} \mid \exists h_1 \in s_1, \\ & \quad h_{23} \in \{h \in \mathcal{H} \mid \exists h_2 \in s_2, h_3 \in s_3 : \forall l' \in \mathcal{L} : \\ & \quad \quad e.l' \circledcirc h = e.l' \circledcirc h_2 \cap e.l' \circledcirc h_3\} : \\ & \quad \forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_1 \cap e.l \circledcirc h_{23}\} \quad \text{by definition (10.13)} \\ &= \{h \in \mathcal{H} \mid \\ & \quad \exists h_1 \in s_1, h_{23} \in \mathcal{H}, h_2 \in s_2, h_3 \in s_3 : \\ & \quad (\forall l' \in \mathcal{L} : e.l' \circledcirc h_{23} = e.l' \circledcirc h_2 \cap e.l' \circledcirc h_3) \wedge \quad \text{by definition (10.13)} \\ & \quad \forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_1 \cap e.l \circledcirc h_{23}\} \quad \text{and assumption 2} \\ &= \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2, h_3 \in s_3 : \forall l \in \mathcal{L} : \quad \text{using } e.l \circledcirc h_{23} = e.l \circledcirc h_2 \cap e.l \circledcirc h_3 \\ & \quad e.l \circledcirc h = e.l \circledcirc h_1 \cap e.l \circledcirc h_2 \cap e.l \circledcirc h_3\} \quad \text{for all } l \in \mathcal{L} \end{aligned}$$

□

Lemma 9 *(To be used when proving associativity of weak sequencing in lemma 11)*

Let d_1 , d_2 and d_3 be syntactically well-formed sequence diagrams such that also $\text{seq } [d_1, d_2]$, $\text{seq } [d_2, d_3]$ and $\text{seq } [\text{seq } [d_1, d_2], d_3]$ are syntactically well-formed.

- A : 1. $s_1 \in tracesets(d_1)$
 2. $s_2 \in tracesets(d_2)$
 3. $s_3 \in tracesets(d_3)$
- P : $s_2 \succsim s_3 = \emptyset \Rightarrow (s_1 \succsim s_2) \succsim s_3 = \emptyset$
- $\langle 1 \rangle$ 1. A : $s_2 \succsim s_3 = \emptyset$
 P : $(s_1 \succsim s_2) \succsim s_3 = \emptyset$
- $\langle 2 \rangle$ 1. C : $s_1 = \emptyset, s_2 = \emptyset, s_3 = \emptyset$ or $s_1 \succsim s_2 = \emptyset$
 P : $(s_1 \succsim s_2) \succsim s_3 = \emptyset$ by definition (10.13) of \succsim .
- $\langle 2 \rangle$ 2. C : $s_1 \neq \emptyset, s_2 \neq \emptyset, s_3 \neq \emptyset$ and $s_1 \succsim s_2 \neq \emptyset$
 $\langle 3 \rangle$ 1. $h \notin \mathcal{H}$ for arbitrary
 $h \in \{h \in [\mathcal{E}]^\omega \mid \exists h_{12} \in s_1 \succsim s_2, h_3 \in s_3 : \forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_{12} \cap e.l \circledcirc h_3\}$
- $\langle 4 \rangle$ 1. Choose $h_{12} \in s_1 \succsim s_2$ and $h_3 \in s_3$ such that
 $\forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_{12} \cap e.l \circledcirc h_3$
 P : $\langle 2 \rangle$ 2 and $\langle 3 \rangle$ 1.
- $\langle 4 \rangle$ 2. Choose $h_1 \in s_1$ and $h_2 \in s_2$ such that
 $\forall l \in \mathcal{L} : e.l \circledcirc h_{12} = e.l \circledcirc h_1 \cap e.l \circledcirc h_2$
 P : $\langle 4 \rangle$ 1 and definition (10.13) of \succsim .
- $\langle 4 \rangle$ 3. $\{h \in \mathcal{H} \mid \exists h_2 \in s_2, h_3 \in s_3 : \forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_2 \cap e.l \circledcirc h_3\}$
 $= \emptyset$
 P : $\langle 1 \rangle$ 1 and definition (10.13) of \succsim .
- $\langle 4 \rangle$ 4. $\forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_1 \cap e.l \circledcirc h_2 \cap e.l \circledcirc h_3$
 P : $\langle 4 \rangle$ 1 and $\langle 4 \rangle$ 2.
- $\langle 4 \rangle$ 5. Choose $h_{23} \lhd h$ such that $\forall l \in \mathcal{L} : e.l \circledcirc h_{23} = e.l \circledcirc h_2 \cap e.l \circledcirc h_3$
 P : $\langle 4 \rangle$ 4, definition (10.66) of \lhd and definitions of \circledcirc and \cap .
- $\langle 4 \rangle$ 6. $h_{23} \notin \mathcal{H}$
 P : $\langle 4 \rangle$ 1, $\langle 4 \rangle$ 2, $\langle 4 \rangle$ 3 and $\langle 4 \rangle$ 5.
- $\langle 4 \rangle$ 7. $h \notin \mathcal{H}$
- $\langle 5 \rangle$ 1. C : h_{23} violates criteria (10.6), i.e.
 $\neg(\forall i, j \in [1.. \#h_{23}] : i < j \Rightarrow r.h_{23}[i] \leq r.h_{23}[j])$
 P : h violates criteria (10.6) by $\langle 4 \rangle$ 5 and lemma 1 on well-formedness.
- $\langle 5 \rangle$ 2. C : h_{23} violates criteria (10.7), i.e.
 $\neg(\forall i, j \in [1.. \#h_{23}] : i \neq j \Rightarrow h_{23}[i] \neq h_{23}[j])$
 P : h violates critiera (10.7) by $\langle 4 \rangle$ 5 and lemma 2 on well-formedness.
- $\langle 5 \rangle$ 3. C : h_{23} violates criteria (10.8), i.e.
 $\neg(\forall l \in \mathcal{L} : \#e.l \circledcirc h_{23} = \infty$
 $\Rightarrow \forall t \in \mathbb{R} : \exists i \in \mathbb{N} : r.(e.l \circledcirc h_{23})[i] > t)$
- $\langle 6 \rangle$ 1. $\forall l \in \mathcal{L} : \#e.l \circledcirc h_2 = \infty$
 $\Rightarrow \forall t \in \mathbb{R} : \exists i \in \mathbb{N} : r.(e.l \circledcirc h_2)[i] > t$
 P : $\langle 4 \rangle$ 2 and assumption 2.
- $\langle 6 \rangle$ 2. $\forall l \in \mathcal{L} : \#e.l \circledcirc h_3 = \infty$
 $\Rightarrow \forall t \in \mathbb{R} : \exists i \in \mathbb{N} : r.(e.l \circledcirc h_3)[i] > t$
 P : $\langle 4 \rangle$ 1 and assumption 3.
- $\langle 6 \rangle$ 3. $\forall l \in \mathcal{L} : \#e.l \circledcirc h_{23} = \infty$

- $\Rightarrow \forall t \in \mathbb{R} : \exists i \in \mathbb{N} : r.(e.l \odot h_{23})[i] > t$
- P : $\langle 6 \rangle 1, \langle 6 \rangle 2$ and $\langle 4 \rangle 5$.
- $\langle 6 \rangle 4.$ Q.E.D.
- P : Case impossible by $\langle 6 \rangle 3$.
- $\langle 5 \rangle 4.$ C : h_{23} violates criteria (10.10), i.e.
- $$\neg(\forall i \in [1.. \#h_{23}] : k.h_{23}[i] =!) \Rightarrow$$
- $$\#(\{\!\!\{ ! \}\!\!\} \times \{m.h_{23}[i]\} \times U) \odot h_{23}|_i >$$
- $$\#(\{\!\!\{ \sim \}\!\!\} \times \{m.h_{23}[i]\} \times U) \odot h_{23}|_i)$$
- $\langle 6 \rangle 1.$ Choose $i \in [1.. \#h_{23}]$ such that $k.h_{23}[i] =!$ and
- $$\#(\{\!\!\{ ! \}\!\!\} \times \{m.h_{23}[i]\} \times U) \odot h_{23}|_i \leq$$
- $$\#(\{\!\!\{ \sim \}\!\!\} \times \{m.h_{23}[i]\} \times U) \odot h_{23}|_i$$
- P : $\langle 5 \rangle 4$ and \neg -rules.
- $\langle 6 \rangle 2.$ L : $m = m.h_{23}[i]$
- P : $\langle 6 \rangle 1$.
- $\langle 6 \rangle 3.$ $(\#\{m \in h_2\}) + (\#\{m \in h_3\}) > 0$
- P : $\langle 4 \rangle 5, \langle 6 \rangle 1$ and $\langle 6 \rangle 2$.
- $\langle 6 \rangle 4.$ $(\#\{m \in h_2\}) + (\#\{m \in h_3\}) > 0$
- P : $\langle 4 \rangle 5, \langle 6 \rangle 1$ and $\langle 6 \rangle 2$.
- $\langle 6 \rangle 5.$ $(\#\{m \in h_2\}) + (\#\{m \in h_3\}) = (\#\{m \in h_2\}) + (\#\{m \in h_3\})$
- P : $\langle 6 \rangle 3, \langle 6 \rangle 4, \langle 4 \rangle 1, \langle 4 \rangle 2$, assumptions 2 and 3 and observation 2, part 2.
- $\langle 6 \rangle 6.$ $(\#\{m \in h\}) = (\#\{m \in h\})$
- P : $\langle 6 \rangle 3, \langle 6 \rangle 4, \langle 4 \rangle 1, \langle 4 \rangle 2$ and observation 1.
- $\langle 6 \rangle 7.$ $(\#\{m \in h_1\}) = (\#\{m \in h_1\})$
- P : $\langle 6 \rangle 5, \langle 6 \rangle 6, \langle 4 \rangle 1$ and $\langle 4 \rangle 2$.
- $\langle 6 \rangle 8.$ $\exists i' \in [1.. \#h] : k.h[i'] =! \wedge$
- $$\#(\{\!\!\{ ! \}\!\!\} \times \{m.h[i']\} \times U) \odot h|_{i'} \leq \#(\{\!\!\{ \sim \}\!\!\} \times \{m.h[i']\} \times U) \odot h|_{i'}$$
- P : $\langle 6 \rangle 1, \langle 6 \rangle 7$ and $\langle 4 \rangle 5$.
- $\langle 6 \rangle 9.$ Q.E.D.
- P : h violates criteria (10.10) by $\langle 6 \rangle 8$.
- $\langle 5 \rangle 5.$ C : h_{23} violates criteria (10.11), i.e.
- $$\neg(\forall i \in [1.. \#h] : k.h_{23}[i] = \sim \Rightarrow$$
- $$\#(\{\!\!\{ \sim \}\!\!\} \times \{m.h_{23}[i]\} \times U) \odot h_{23}|_i >$$
- $$\#(\{\!\!\{ ? \}\!\!\} \times \{m.h_{23}[i]\} \times U) \odot h_{23}|_i)$$
- $\langle 6 \rangle 1.$ Choose $i \in [1.. \#h_{23}]$ such that $k.h_{23}[i] = \sim$ and
- $$\#(\{\!\!\{ \sim \}\!\!\} \times \{m.h_{23}[i]\} \times U) \odot h_{23}|_i \leq$$
- $$\#(\{\!\!\{ ? \}\!\!\} \times \{m.h_{23}[i]\} \times U) \odot h_{23}|_i$$
- P : $\langle 5 \rangle 5$ and \neg -rules.
- $\langle 6 \rangle 2.$ L : $m = m.h_{23}[i]$
- P : $\langle 6 \rangle 1$.
- $\langle 6 \rangle 3.$ $(\#\{m \in h_2\}) + (\#\{m \in h_3\}) > 0$
- P : $\langle 4 \rangle 5, \langle 6 \rangle 1$ and $\langle 6 \rangle 2$.
- $\langle 6 \rangle 4.$ $(\#\{?m \in h_2\}) + (\#\{?m \in h_3\}) > 0$
- P : $\langle 4 \rangle 5, \langle 6 \rangle 1$ and $\langle 6 \rangle 2$.
- $\langle 6 \rangle 5.$ $(\#\{m \in h_2\}) + (\#\{m \in h_3\}) = (\#\{?m \in h_2\}) + (\#\{?m \in h_3\})$

P : $\langle 6 \rangle 3, \langle 6 \rangle 4, \langle 4 \rangle 1, \langle 4 \rangle 2$, assumptions 2 and 3 and observation 2, part 1.

$\langle 6 \rangle 6. (\# \sim m \in h) = (\#?m \in h)$

P : $\langle 6 \rangle 3, \langle 6 \rangle 4, \langle 4 \rangle 1, \langle 4 \rangle 2$ and observation 1.

$\langle 6 \rangle 7. (\# \sim m \in h_1) = (\#?m \in h_1)$

P : $\langle 6 \rangle 5, \langle 6 \rangle 6, \langle 4 \rangle 1$ and $\langle 4 \rangle 2$.

$\langle 6 \rangle 8. \exists i' \in [1..h] : k.h[i'] = \sim \wedge$
 $\#(\{\sim\} \times \{m.h[i']\} \times U) \odot h|_{i'} \leq \#(\{?\} \times \{m.h[i']\} \times U) \odot h|_{i'}$

P : $\langle 6 \rangle 1, \langle 6 \rangle 7$ and $\langle 4 \rangle 5$.

$\langle 6 \rangle 9. \text{Q.E.D.}$

P : h violates criteria (10.11) by $\langle 6 \rangle 8$.

$\langle 5 \rangle 6. \text{Q.E.D.}$

P : The cases are exhaustive by $\langle 4 \rangle 6$ and definition of \mathcal{H} .

$\langle 4 \rangle 8. \text{Q.E.D.}$

P : $\langle 4 \rangle 7$.

$\langle 3 \rangle 2. \{h \in \mathcal{H} \mid \exists h_{12} \in s_1 \succsim s_2, h_3 \in s_3 : \forall l \in \mathcal{L} : e.l \odot h = e.l \odot h_{12} \cap e.l \odot h_3\}$
 $= \emptyset$

P : $\langle 3 \rangle 1$.

$\langle 3 \rangle 3. \text{Q.E.D.}$

P : $(s_1 \succsim s_2) \succsim s_3 = \emptyset$ by $\langle 3 \rangle 2$ and definition (10.13) of \succsim .

$\langle 2 \rangle 3. \text{Q.E.D.}$

P : The cases are exhaustive.

$\langle 1 \rangle 2. \text{Q.E.D.}$

P : \Rightarrow -rule.

□

Lemma 10 (To be used when proving associativity of weak sequencing in lemma 11)

Let d_1 , d_2 and d_3 be syntactically well-formed sequence diagrams such that also $\text{seq } [d_1, d_2]$, $\text{seq } [d_2, d_3]$ and $\text{seq } [d_1, \text{seq } [d_2, d_3]]$ are syntactically well-formed.

A : 1. $s_1 \in \text{tracesets}(d_1)$
2. $s_2 \in \text{tracesets}(d_2)$
3. $s_3 \in \text{tracesets}(d_3)$

P : $s_1 \succsim s_2 = \emptyset \Rightarrow s_1 \succsim (s_2 \succsim s_3) = \emptyset$

P :

Symmetrical to the proof of lemma 9.

□

Lemma 11 *Associativity of weak sequencing*

Let d_1 , d_2 and d_3 be syntactically well-formed sequence diagrams such that also $\text{seq } [d_1, d_2]$, $\text{seq } [d_2, d_3]$, $\text{seq } [\text{seq } [d_1, d_2], d_3]$ and $\text{seq } [d_1, [\text{seq } [d_2, d_3]]]$ are syntactically well-formed.

A : 1. $s_1 \in \text{tracesets}(d_1)$

2. $s_2 \in \text{tracesets}(d_2)$

3. $s_3 \in \text{tracesets}(d_3)$

P : $(s_1 \succsim s_2) \succsim s_3 = s_1 \succsim (s_2 \succsim s_3)$

$\langle 1 \rangle 1.$ C : $s_1 \succsim s_2 = \emptyset$

$\langle 2 \rangle 1.$ $(s_1 \succsim s_2) \succsim s_3 = \emptyset$

P : Definition (10.13) of \succsim .

$\langle 2 \rangle 2.$ $s_1 \succsim (s_2 \succsim s_3) = \emptyset$

P : $\langle 1 \rangle 1$ and lemma 10.

$\langle 2 \rangle 3.$ Q.E.D.

$\langle 1 \rangle 2.$ C : $s_2 \succsim s_3 = \emptyset$

$\langle 2 \rangle 1.$ $s_1 \succsim (s_2 \succsim s_3) = \emptyset$

P : Definition (10.13) of \succsim .

$\langle 2 \rangle 2.$ $(s_1 \succsim s_2) \succsim s_3 = \emptyset$

P : $\langle 1 \rangle 2$ and lemma 9.

$\langle 2 \rangle 3.$ Q.E.D.

$\langle 1 \rangle 3.$ C : $s_1 \succsim s_2 \neq \emptyset \wedge s_2 \succsim s_3 \neq \emptyset$

P : $(s_1 \succsim s_2) \succsim s_3 = s_1 \succsim (s_2 \succsim s_3)$ by lemma 8.

$\langle 1 \rangle 4.$ Q.E.D.

P : The cases are exhaustive.

□

Lemma 12 *Left distributivity of parallel execution \parallel over union \cup*

P : $s_1 \parallel (s_2 \cup s_3) = (s_1 \parallel s_2) \cup (s_1 \parallel s_3)$

P :

$s_1 \parallel (s_2 \cup s_3)$

= $\{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty :$

$\pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1 \wedge$

$\pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in s_2 \cup s_3\}$ by definition (10.12)

= $\{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty :$

$\pi_2((\{1\} \times [\mathcal{E}]) \oplus (p, h)) \in s_1 \wedge$

$(\pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in s_2 \vee$

$\pi_2((\{2\} \times [\mathcal{E}]) \oplus (p, h)) \in s_3)\}$ by definition of \cup

$$\begin{aligned}
&= \{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty : \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \circledast (p, h)) \in s_1 \wedge \\
&\quad \pi_2((\{2\} \times [\mathcal{E}]) \circledast (p, h)) \in s_2\} \\
&\quad \cup \\
&\quad \{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty : \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \circledast (p, h)) \in s_1 \wedge \\
&\quad \pi_2((\{2\} \times [\mathcal{E}]) \circledast (p, h)) \in s_3\} \\
&= (s_1 \parallel s_2) \cup (s_1 \parallel s_3) \quad \text{by definition (10.12)}
\end{aligned}$$

Lemma 13 Right distributivity of parallel execution \parallel over union \cup

$$P : (s_1 \cup s_2) \parallel s_3 = (s_1 \parallel s_3) \cup (s_2 \parallel s_3)$$

P :

$$\begin{aligned}
&(s_1 \cup s_2) \parallel s_3 \\
&= \{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty : \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \circledast (p, h)) \in s_1 \cup s_2 \wedge \\
&\quad \pi_2((\{2\} \times [\mathcal{E}]) \circledast (p, h)) \in s_3\} \quad \text{by definition (10.12)} \\
&= \{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty : \\
&\quad (\pi_2((\{1\} \times [\mathcal{E}]) \circledast (p, h)) \in s_1 \vee \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \circledast (p, h)) \in s_2) \wedge \\
&\quad \pi_2((\{2\} \times [\mathcal{E}]) \circledast (p, h)) \in s_3\} \quad \text{by definition of } \cup \\
&= \{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty : \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \circledast (p, h)) \in s_1 \wedge \\
&\quad \pi_2((\{2\} \times [\mathcal{E}]) \circledast (p, h)) \in s_3\} \\
&\quad \cup \\
&\quad \{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty : \\
&\quad \pi_2((\{1\} \times [\mathcal{E}]) \circledast (p, h)) \in s_2 \wedge \\
&\quad \pi_2((\{2\} \times [\mathcal{E}]) \circledast (p, h)) \in s_3\} \\
&= (s_1 \parallel s_3) \cup (s_2 \parallel s_3) \quad \text{by definition (10.12)}
\end{aligned}$$

Lemma 14 Left distributivity of weak sequencing \succsim over union \cup

$$P : s_1 \succsim (s_2 \cup s_3) = (s_1 \succsim s_2) \cup (s_1 \succsim s_3)$$

P :

$$\begin{aligned}
&s_1 \succsim (s_2 \cup s_3) \\
&= \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in (s_2 \cup s_3) : \\
&\quad \forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_1 \frown e.l \circledcirc h_2\} \quad \text{by definition (10.13)} \\
&= \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \\
&\quad \forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_1 \frown e.l \circledcirc h_2\} \\
&\quad \cup \\
&\quad \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_3 : \\
&\quad \forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_1 \frown e.l \circledcirc h_2\} \\
&= (s_1 \succsim s_2) \cup (s_1 \succsim s_3) \quad \text{by definition (10.13)}
\end{aligned}$$

Lemma 15 Right distributivity of weak sequencing \succsim over union \cup

$$P : (s_1 \cup s_2) \succsim s_3 = (s_1 \succsim s_3) \cup (s_2 \succsim s_3)$$

P :

$$(s_1 \cup s_2) \succsim s_3$$

$$\begin{aligned} &= \{h \in \mathcal{H} \mid \exists h_1 \in (s_1 \cup s_2), h_2 \in s_3 : \\ &\quad \forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_1 \cap e.l \circledcirc h_2\} \text{ by definition (10.13)} \\ &= \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_3 : \\ &\quad \forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_1 \cap e.l \circledcirc h_2\} \\ &\quad \cup \\ &\quad \{h \in \mathcal{H} \mid \exists h_1 \in s_2, h_2 \in s_3 : \\ &\quad \forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_1 \cap e.l \circledcirc h_2\} \\ &= (s_1 \succsim s_3) \cup (s_2 \succsim s_3) \text{ by definition (10.13)} \end{aligned}$$

Lemma 16 Distributivity of time constraint \wr over union \cup

$$P : (s_1 \cup s_2) \wr C = (s_1 \wr C) \cup (s_2 \wr C)$$

P :

$$(s_1 \cup s_2) \wr C$$

$$\begin{aligned} &= \{h \in (s_1 \cup s_2) \mid h \models C\} \text{ by definition (10.14)} \\ &= \{h \in s_1 \mid h \models C\} \\ &\quad \cup \\ &\quad \{h \in s_2 \mid h \models C\} \\ &= (s_1 \wr C) \cup (s_2 \wr C) \text{ by definition (10.14)} \end{aligned}$$

10.C.3 Lemmas on Interaction Obligations

Lemma 17 Commutativity of parallel execution

$$P : (p_1, n_1) \parallel (p_2, n_2) = (p_2, n_2) \parallel (p_1, n_1)$$

P :

$$(p_1, n_1) \parallel (p_2, n_2)$$

$$\begin{aligned} &= (p_1 \parallel p_2, (n_1 \parallel (p_2 \cup n_2)) \cup (n_2 \parallel p_1)) \text{ by definition (10.15)} \\ &= (p_1 \parallel p_2, n_1 \parallel p_2 \cup n_1 \parallel n_2 \parallel p_1) \text{ by lemma 12 (distributivity of } \parallel \text{ over } \cup \text{)} \\ &= (p_2 \parallel p_1, n_1 \parallel p_2 \cup n_2 \parallel n_1 \cup n_2 \parallel p_1) \text{ by lemma 3 (commutativity of } \parallel \text{)} \\ &= (p_2 \parallel p_1, n_2 \parallel p_1 \cup n_2 \parallel n_1 \cup n_1 \parallel p_2) \text{ by commutativity of } \cup \\ &= (p_2 \parallel p_1, (n_2 \parallel (p_1 \cup n_1)) \cup (n_1 \parallel p_2)) \text{ by lemma 12 (distributivity of } \parallel \text{ over } \cup \text{)} \\ &= (p_2, n_2) \parallel (p_1, n_1) \text{ by definition (10.15)} \end{aligned}$$

Lemma 18 *Associativity of parallel execution*

Let d_1 , d_2 and d_3 be syntactically well-formed sequence diagrams such that also $d_1 \text{ par } d_2$, $d_2 \text{ par } d_3$, $(d_1 \text{ par } d_2) \text{ par } d_3$ and $d_1 \text{ par } (d_2 \text{ par } d_3)$ are syntactically well-formed.

$$\text{P} : \forall (p_1, n_1) \in [d_1], (p_2, n_2) \in [d_2], (p_3, n_3) \in [d_3] : \\ ((p_1, n_1) \parallel (p_2, n_2)) \parallel (p_3, n_3) = (p_1, n_1) \parallel ((p_2, n_2) \parallel (p_3, n_3))$$

P :

The two sides of the equation reduce to the same formula.

Left side:

$$\begin{aligned} & ((p_1, n_1) \parallel (p_2, n_2)) \parallel (p_3, n_3) \\ &= (p_1 \parallel p_2, (n_1 \parallel (p_2 \cup n_2)) \cup (n_2 \parallel p_1)) \parallel (p_3, n_3) && \text{by definition (10.15)} \\ &= ((p_1 \parallel p_2) \parallel p_3, \\ &\quad (((n_1 \parallel (p_2 \cup n_2)) \cup (n_2 \parallel p_1)) \parallel (p_3 \cup n_3)) \cup \\ &\quad (n_3 \parallel (p_1 \parallel p_2))) && \text{by definition (10.15)} \\ &= ((p_1 \parallel p_2) \parallel p_3, \\ &\quad ((n_1 \parallel p_2 \cup n_1 \parallel n_2 \cup n_2 \parallel p_1) \parallel (p_3 \cup n_3)) \cup \\ &\quad (n_3 \parallel (p_1 \parallel p_2))) && \text{by lemma 12} \\ &= ((p_1 \parallel p_2) \parallel p_3, \\ &\quad ((n_1 \parallel p_2 \cup n_1 \parallel n_2 \cup n_2 \parallel p_1) \parallel p_3) \cup \\ &\quad ((n_1 \parallel p_2 \cup n_1 \parallel n_2 \cup n_2 \parallel p_1) \parallel n_3) \cup \\ &\quad (n_3 \parallel (p_1 \parallel p_2))) && \text{(distributivity of } \parallel \text{ over } \cup\text{)} \\ &= ((p_1 \parallel p_2) \parallel p_3, \\ &\quad (n_1 \parallel p_2) \parallel p_3 \cup (n_1 \parallel n_2) \parallel p_3 \cup (n_2 \parallel p_1) \parallel p_3 \cup \\ &\quad (n_1 \parallel p_2) \parallel n_3 \cup (n_1 \parallel n_2) \parallel n_3 \cup (n_2 \parallel p_1) \parallel n_3 \cup \\ &\quad n_3 \parallel (p_1 \parallel p_2))) && \text{by lemma 12} \\ & && \text{(distributivity of } \parallel \text{ over } \cup\text{)} \end{aligned}$$

Right side:

$$\begin{aligned} & (p_1, n_1) \parallel ((p_2, n_2) \parallel (p_3, n_3)) \\ &= (p_1, n_1) \parallel (p_2 \parallel p_3, (n_2 \parallel (p_3 \cup n_3)) \cup (n_3 \parallel p_2)) && \text{by definition (10.15)} \\ &= (p_1 \parallel (p_2 \parallel p_3), \\ &\quad (n_1 \parallel ((p_2 \parallel p_3) \cup ((n_2 \parallel (p_3 \cup n_3)) \cup (n_3 \parallel p_2)))) \cup \\ &\quad (((n_2 \parallel (p_3 \cup n_3)) \cup (n_3 \parallel p_2)) \parallel p_1)) && \text{by definition (10.15)} \\ &= (p_1 \parallel (p_2 \parallel p_3), \\ &\quad (n_1 \parallel ((p_2 \parallel p_3) \cup (n_2 \parallel p_3 \cup n_2 \parallel n_3 \cup n_3 \parallel p_2))) \cup \\ &\quad ((n_2 \parallel p_3 \cup n_2 \parallel n_3 \cup n_3 \parallel p_2) \parallel p_1)) && \text{by lemma 12} \\ &= (p_1 \parallel (p_2 \parallel p_3), \\ &\quad n_1 \parallel (p_2 \parallel p_3) \cup n_1 \parallel (n_2 \parallel p_3) \cup n_1 \parallel (n_2 \parallel n_3) \cup \\ &\quad n_1 \parallel (n_3 \parallel p_2) \cup (n_2 \parallel p_3) \parallel p_1 \cup (n_2 \parallel n_3) \parallel p_1 \cup \\ &\quad (n_3 \parallel p_2) \parallel p_1) && \text{by lemma 12} \\ &= (p_1 \parallel (p_2 \parallel p_3), \\ &\quad n_1 \parallel (p_2 \parallel p_3) \cup n_1 \parallel (n_2 \parallel p_3) \cup (n_2 \parallel p_3) \parallel p_1) \cup \\ &\quad n_1 \parallel (n_3 \parallel p_2) \cup n_1 \parallel (n_2 \parallel n_3) \cup (n_2 \parallel n_3) \parallel p_1 \cup \\ &\quad (n_3 \parallel p_2) \parallel p_1) && \text{by commutativity of } \cup \end{aligned}$$

$$\begin{aligned}
&= ((p_1 \parallel p_2) \parallel p_3, \\
&\quad (n_1 \parallel p_2) \parallel p_3 \cup (n_1 \parallel n_2) \parallel p_3 \cup (n_2 \parallel p_1) \parallel p_3 \cup \\
&\quad (n_1 \parallel p_2) \parallel n_3 \cup (n_1 \parallel n_2) \parallel n_3 \cup (n_2 \parallel p_1) \parallel n_3 \cup \\
&\quad n_3 \parallel (p_1 \parallel p_2)) \quad \text{by lemmas 3 and 7} \\
&\quad \text{(commutativity and} \\
&\quad \text{associativity of } \parallel)
\end{aligned}$$

□

Lemma 19 *Associativity of weak sequencing*

Let d_1 , d_2 and d_3 be syntactically well-formed sequence diagrams such that also $\text{seq } [d_1, d_2]$, $\text{seq } [d_2, d_3]$, $\text{seq } [\text{seq } [d_1, d_2], d_3]$ and $\text{seq } [d_1, \text{seq } [d_2, d_3]]$ are syntactically well-formed.

$$\begin{aligned}
P &: \forall (p_1, n_1) \in [d_1], (p_2, n_2) \in [d_2], (p_3, n_3) \in [d_3] : \\
&\quad ((p_1, n_1) \succsim (p_2, n_2)) \succsim (p_3, n_3) = (p_1, n_1) \succsim ((p_2, n_2) \succsim (p_3, n_3))
\end{aligned}$$

P :

The two sides of the equation reduce to the same formula.

Left side:

$$\begin{aligned}
&((p_1, n_1) \succsim (p_2, n_2)) \succsim (p_3, n_3) \\
&= (p_1 \succsim p_2, (n_1 \succsim (n_2 \cup p_2)) \cup (p_1 \succsim n_2)) \succsim (p_3, n_3) \quad \text{by definition (10.16)} \\
&= ((p_1 \succsim p_2) \succsim p_3, \\
&\quad (((n_1 \succsim (n_2 \cup p_2)) \cup (p_1 \succsim n_2)) \succsim (n_3 \cup p_3)) \cup \\
&\quad ((p_1 \succsim p_2) \succsim n_3)) \quad \text{by definition (10.16)} \\
&= ((p_1 \succsim p_2) \succsim p_3, \\
&\quad ((n_1 \succsim n_2 \cup n_1 \succsim p_2 \cup p_1 \succsim n_2) \succsim (n_3 \cup p_3)) \cup \\
&\quad ((p_1 \succsim p_2) \succsim n_3)) \quad \text{by lemma 14} \\
&= ((p_1 \succsim p_2) \succsim p_3, \\
&\quad ((n_1 \succsim n_2 \cup n_1 \succsim p_2 \cup p_1 \succsim n_2) \succsim n_3) \cup \\
&\quad ((n_1 \succsim n_2 \cup n_1 \succsim p_2 \cup p_1 \succsim n_2) \succsim p_3) \cup \\
&\quad ((p_1 \succsim p_2) \succsim n_3)) \quad \text{by lemma 14} \\
&= ((p_1 \succsim p_2) \succsim p_3, \\
&\quad (n_1 \succsim n_2) \succsim n_3 \cup (n_1 \succsim p_2) \succsim n_3 \cup (p_1 \succsim n_2) \succsim n_3 \cup \\
&\quad (n_1 \succsim n_2) \succsim p_3 \cup (n_1 \succsim p_2) \succsim p_3 \cup (p_1 \succsim n_2) \succsim p_3 \cup \\
&\quad (p_1 \succsim p_2) \succsim n_3) \quad \text{by lemma 15} \\
&= ((p_1 \succsim p_2) \succsim p_3, \\
&\quad (n_1 \succsim n_2 \cup n_1 \succsim p_3 \cup p_2 \succsim n_3 \cup p_2 \succsim p_3) \cup \\
&\quad (p_1 \succsim (n_2 \succsim n_3 \cup n_2 \succsim p_3 \cup p_2 \succsim n_3))) \quad \text{by definition (10.16)}
\end{aligned}$$

Right side:

$$\begin{aligned}
&(p_1, n_1) \succsim ((p_2, n_2) \succsim (p_3, n_3)) \\
&= (p_1, n_1) \succsim (p_2 \succsim p_3, (n_2 \succsim (n_3 \cup p_3)) \cup (p_2 \succsim n_3)) \quad \text{by definition (10.16)} \\
&= (p_1 \succsim (p_2 \succsim p_3), \\
&\quad (n_1 \succsim (((n_2 \succsim (n_3 \cup p_3)) \cup (p_2 \succsim n_3)) \cup (p_2 \succsim p_3))) \cup \\
&\quad (p_1 \succsim ((n_2 \succsim (n_3 \cup p_3)) \cup (p_2 \succsim n_3)))) \quad \text{by definition (10.16)} \\
&= (p_1 \succsim (p_2 \succsim p_3), \\
&\quad (n_1 \succsim (n_2 \succsim n_3 \cup n_2 \succsim p_3 \cup p_2 \succsim n_3 \cup p_2 \succsim p_3)) \cup \\
&\quad (p_1 \succsim (n_2 \succsim n_3 \cup n_2 \succsim p_3 \cup p_2 \succsim n_3))) \quad \text{by lemma 14} \\
&= (p_1 \succsim (p_2 \succsim p_3), \\
&\quad n_1 \succsim (n_2 \succsim n_3) \cup n_1 \succsim (n_2 \succsim p_3) \cup n_1 \succsim (p_2 \succsim n_3) \cup \\
&\quad n_1 \succsim (p_2 \succsim p_3) \cup p_1 \succsim (n_2 \succsim n_3) \cup p_1 \succsim (n_2 \succsim p_3) \cup \\
&\quad p_1 \succsim (p_2 \succsim n_3)) \quad \text{by lemma 14} \\
&\quad \text{(distributivity of } \succsim \text{ over } \cup \text{)}
\end{aligned}$$

$$\begin{aligned}
&= (p_1 \asymp (p_2 \asymp p_3), \\
&\quad n_1 \asymp (n_2 \asymp n_3) \cup n_1 \asymp (p_2 \asymp n_3) \cup p_1 \asymp (n_2 \asymp n_3) \cup \\
&\quad n_1 \asymp (n_2 \asymp p_3) \cup n_1 \asymp (p_2 \asymp p_3) \cup p_1 \asymp (n_2 \asymp p_3) \cup \\
&\quad p_1 \asymp (p_2 \asymp n_3)) && \text{by commutativity of } \cup \\
&= ((p_1 \asymp p_2) \asymp p_3, \\
&\quad (n_1 \asymp n_2) \asymp n_3 \cup (n_1 \asymp p_2) \asymp n_3 \cup (p_1 \asymp n_2) \asymp n_3 \cup \\
&\quad (n_1 \asymp n_2) \asymp p_3 \cup (n_1 \asymp p_2) \asymp p_3 \cup (p_1 \asymp n_2) \asymp p_3 \cup \\
&\quad (p_1 \asymp p_2) \asymp n_3) && \text{by lemma 11 (associativity of } \asymp) \\
&&& \square
\end{aligned}$$

10.C.4 Lemmas on Sets of Interaction Obligations

Lemma 20 *Commutativity of inner union*

$$P : O_1 \uplus O_2 = O_2 \uplus O_1$$

P :

$$\begin{aligned}
&O_1 \uplus O_2 \\
&= \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2\} && \text{by definition (10.30)} \\
&= \{(p_2 \cup p_1, n_2 \cup n_1) \mid (p_2, n_2) \in O_1 \wedge (p_1, n_1) \in O_2\} && \text{by renaming} \\
&= \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O_2 \wedge (p_2, n_2) \in O_1\} && \text{by commutativity of } \cup \text{ and } \wedge \\
&= O_2 \uplus O_1 && \text{by definition (10.30)}
\end{aligned}$$

Lemma 21 *Associativity of inner union*

$$P : (O_1 \uplus O_2) \uplus O_3 = O_1 \uplus (O_2 \uplus O_3)$$

P :

The two sides of the equation reduce to the same formula.

Left side:

$$\begin{aligned}
&(O_1 \uplus O_2) \uplus O_3 \\
&= \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O_1 \uplus O_2 \wedge (p_2, n_2) \in O_3\} && \text{by definition (10.30)} \\
&= \{(p_{12} \cup p_3, n_{12} \cup n_3) \mid (p_{12}, n_{12}) \in O_1 \uplus O_2 \wedge (p_3, n_3) \in O_3\} && \text{by renaming} \\
&= \{(p_{12} \cup p_3, n_{12} \cup n_3) \mid \\
&\quad (p_{12}, n_{12}) \in \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2\} \wedge \\
&\quad (p_3, n_3) \in O_3\} && \text{by definition (10.30)} \\
&= \{((p_1 \cup p_2) \cup p_3, (n_1 \cup n_2) \cup n_3) \mid \\
&\quad (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2 \wedge (p_3, n_3) \in O_3\} \\
&= \{(p_1 \cup p_2 \cup p_3, n_1 \cup n_2 \cup n_3) \mid \\
&\quad (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2 \wedge (p_3, n_3) \in O_3\} && \text{by commutativity} \\
&&& \text{of } \cup
\end{aligned}$$

Right side:

$$\begin{aligned}
&O_1 \uplus (O_2 \uplus O_3) \\
&= \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2 \uplus O_3\} && \text{by definition (10.30)} \\
&= \{(p_1 \cup p_{23}, n_1 \cup n_{23}) \mid (p_1, n_1) \in O_1 \wedge (p_{23}, n_{23}) \in O_2 \uplus O_3\} && \text{by renaming} \\
&= \{(p_1 \cup p_{23}, n_1 \cup n_{23}) \mid (p_1, n_1) \in O_1 \wedge \\
&\quad (p_{23}, n_{23}) \in \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O_2 \wedge (p_2, n_2) \in O_3\}\} && \text{by definition (10.30)} \\
&= \{(p_1 \cup p_{23}, n_1 \cup n_{23}) \mid (p_1, n_1) \in O_1 \wedge \\
&\quad (p_{23}, n_{23}) \in \{(p_2 \cup p_3, n_2 \cup n_3) \mid (p_2, n_2) \in O_2 \wedge (p_3, n_3) \in O_3\}\} && \text{by renaming}
\end{aligned}$$

$$\begin{aligned}
&= \{(p_1 \cup (p_2 \cup p_3), n_1 \cup (n_2 \cup n_3)) \mid \\
&\quad (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2 \wedge (p_3, n_3) \in O_3\} \\
&= \{(p_1 \cup p_2 \cup p_3, n_1 \cup n_2 \cup n_3) \mid \\
&\quad (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2 \wedge (p_3, n_3) \in O_3\} \text{ by commutativity} \\
&\quad \text{of } \cup
\end{aligned}$$

□

Lemma 22 *Commutativity of parallel execution*

$$P : O_1 \parallel O_2 = O_2 \parallel O_1$$

P :

$$\begin{aligned}
O_1 \parallel O_2 &= \{o_1 \parallel o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\} \text{ by definition (10.18)} \\
&= \{o_2 \parallel o_1 \mid o_2 \in O_1 \wedge o_1 \in O_2\} \text{ by renaming} \\
&= \{o_1 \parallel o_2 \mid o_1 \in O_2 \wedge o_2 \in O_1\} \text{ by commutativity of } \parallel \text{ (lemma 17) and } \wedge \\
&= O_2 \parallel O_1 \text{ by definition (10.18)}
\end{aligned}$$

Lemma 23 *Associativity of parallel execution*

Let d_1 , d_2 and d_3 be syntactically well-formed sequence diagrams such that also $d_1 \text{ par } d_2$, $d_2 \text{ par } d_3$, $(d_1 \text{ par } d_2) \text{ par } d_3$ and $d_1 \text{ par } (d_2 \text{ par } d_3)$ are syntactically well-formed.

$$\begin{aligned}
A : & 1. \llbracket d_1 \rrbracket = O_1 \\
& 2. \llbracket d_2 \rrbracket = O_2 \\
& 3. \llbracket d_3 \rrbracket = O_3
\end{aligned}$$

$$P : (O_1 \parallel O_2) \parallel O_3 = O_1 \parallel (O_2 \parallel O_3)$$

P :

The two sides of the equation reduce to the same formula.

Left side:

$$\begin{aligned}
(O_1 \parallel O_2) \parallel O_3 &= \{o_1 \parallel o_2 \mid o_1 \in O_1 \parallel O_2 \wedge o_2 \in O_3\} \text{ by definition (10.18)} \\
&= \{o_{12} \parallel o_3 \mid o_{12} \in O_1 \parallel O_2 \wedge o_3 \in O_3\} \text{ by renaming} \\
&= \{o_{12} \parallel o_3 \mid \\
&\quad o_{12} \in \{o_1 \parallel o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\} \wedge \\
&\quad o_3 \in O_3\} \text{ by definition (10.18)} \\
&= \{(o_1 \parallel o_2) \parallel o_3 \mid \\
&\quad o_1 \in O_1 \wedge o_2 \in O_2 \wedge o_3 \in O_3\}
\end{aligned}$$

Right side:

$$\begin{aligned}
O_1 \parallel (O_2 \parallel O_3) &= \{o_1 \parallel o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2 \parallel O_3\} \text{ by definition (10.18)} \\
&= \{o_1 \parallel o_{23} \mid o_1 \in O_1 \wedge o_{23} \in O_2 \parallel O_3\} \text{ by renaming} \\
&= \{o_1 \parallel o_{23} \mid o_1 \in O_1 \wedge \\
&\quad o_{23} \in \{o_1 \parallel o_2 \mid o_1 \in O_2 \wedge o_2 \in O_3\}\} \text{ by definition (10.18)} \\
&= \{o_1 \parallel o_{23} \mid o_1 \in O_1 \wedge \\
&\quad o_{23} \in \{o_2 \parallel o_3 \mid o_2 \in O_2 \wedge o_3 \in O_3\}\} \text{ by renaming}
\end{aligned}$$

$$\begin{aligned}
&= \{o_1 \parallel (o_2 \parallel o_3) \mid \\
&\quad o_1 \in O_1 \wedge o_2 \in O_2 \wedge o_3 \in O_3\} \\
&= \{(o_1 \parallel o_2) \parallel o_3 \mid \\
&\quad o_1 \in O_1 \wedge o_2 \in O_2 \wedge o_3 \in d_3\} \quad \text{by lemma 18 (associativity of } \parallel \text{)}
\end{aligned}$$

□

Lemma 24 *Associativity of weak sequencing*

Let d_1 , d_2 and d_3 be syntactically well-formed sequence diagrams such that also $\text{seq } [d_1, d_2]$, $\text{seq } [d_2, d_3]$, $\text{seq } [\text{seq } [d_1, d_2], d_3]$ and $\text{seq } [d_1, \text{seq } [d_2, d_3]]$ are syntactically well-formed.

$$\begin{aligned}
A &: \begin{aligned} 1. \quad &[\![d_1]\!] = O_1 \\ 2. \quad &[\![d_2]\!] = O_2 \\ 3. \quad &[\![d_3]\!] = O_3 \end{aligned} \\
P &: (O_1 \succsim O_2) \succsim O_3 = O_1 \succsim (O_2 \succsim O_3) \\
P &:
\end{aligned}$$

The two sides of the equation reduce to the same formula.

Left side:

$$\begin{aligned}
(O_1 \succsim O_2) \succsim O_3 &= \{o_1 \succsim o_2 \mid o_1 \in O_1 \succsim O_2 \wedge o_2 \in O_2\} \quad \text{by definition (10.19)} \\
&= \{o_{12} \succsim o_3 \mid o_{12} \in O_1 \succsim O_2 \wedge o_3 \in O_3\} \quad \text{by renaming} \\
&= \{o_{12} \succsim o_3 \mid \\
&\quad o_{12} \in \{o_1 \succsim o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\} \wedge \\
&\quad o_3 \in O_3\} \quad \text{by definition (10.19)} \\
&= \{(o_1 \succsim o_2) \succsim o_3 \mid \\
&\quad o_1 \in O_1 \wedge o_2 \in O_2 \wedge o_3 \in O_3\}
\end{aligned}$$

Right side:

$$\begin{aligned}
O_1 \succsim (O_2 \succsim O_3) &= \{o_1 \succsim o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2 \succsim O_3\} \quad \text{by definition (10.19)} \\
&= \{o_1 \succsim o_{23} \mid o_1 \in O_1 \wedge o_{23} \in O_2 \succsim O_3\} \quad \text{by renaming} \\
&= \{o_1 \succsim o_{23} \mid o_1 \in O_1 \wedge \\
&\quad o_{23} \in \{o_1 \succsim o_2 \mid o_1 \in O_2 \wedge o_2 \in O_3\}\} \quad \text{by definition (10.19)} \\
&= \{o_1 \succsim o_{23} \mid o_1 \in O_1 \wedge \\
&\quad o_{23} \in \{o_2 \succsim o_3 \mid o_2 \in O_2 \wedge o_3 \in O_3\}\} \quad \text{by renaming} \\
&= \{o_1 \succsim (o_2 \succsim o_3) \mid \\
&\quad o_1 \in O_1 \wedge o_2 \in O_2 \wedge o_3 \in O_3\} \\
&= \{(o_1 \succsim o_2) \succsim o_3 \mid \\
&\quad o_1 \in O_1 \wedge o_2 \in O_2 \wedge o_3 \in O_3\} \quad \text{by lemma 19 (associativity of } \succsim \text{)}
\end{aligned}$$

□

10.C.5 Theorems on Sequence Diagram Operators

Theorem 1 *Commutativity of potential alternative*

$$P : d_1 \text{ alt } d_2 = d_2 \text{ alt } d_1$$

P :

$$\begin{aligned} & \llbracket d_1 \text{ alt } d_2 \rrbracket \\ &= \llbracket d_1 \rrbracket \uplus \llbracket d_2 \rrbracket \quad \text{by definition (10.37)} \\ &= \llbracket d_2 \rrbracket \uplus \llbracket d_1 \rrbracket \quad \text{by commutativity of } \uplus \text{ (lemma 20)} \\ &= \llbracket d_2 \text{ alt } d_1 \rrbracket \quad \text{by definition (10.37)} \end{aligned}$$

Theorem 2 *Associativity of potential alternative*

$$P : (d_1 \text{ alt } d_2) \text{ alt } d_3 = d_1 \text{ alt } (d_2 \text{ alt } d_3)$$

P :

$$\begin{aligned} & \llbracket (d_1 \text{ alt } d_2) \text{ alt } d_3 \rrbracket \\ &= \llbracket d_1 \text{ alt } d_2 \rrbracket \uplus \llbracket d_3 \rrbracket \quad \text{by definition (10.37)} \\ &= (\llbracket d_1 \rrbracket \uplus \llbracket d_2 \rrbracket) \uplus \llbracket d_3 \rrbracket \quad \text{by definition (10.37)} \\ &= \llbracket d_1 \rrbracket \uplus (\llbracket d_2 \rrbracket \uplus \llbracket d_3 \rrbracket) \quad \text{by lemma 21 (associativity of } \uplus \text{)} \\ &= \llbracket d_1 \rrbracket \uplus \llbracket d_2 \text{ alt } d_3 \rrbracket \quad \text{by definition (10.37)} \\ &= \llbracket d_1 \text{ alt } (d_2 \text{ alt } d_3) \rrbracket \quad \text{by definition (10.37)} \end{aligned}$$

Theorem 3 *Commutativity of mandatory alternative*

$$P : d_1 \text{ xalt } d_2 = d_2 \text{ xalt } d_1$$

P :

$$\begin{aligned} & \llbracket d_1 \text{ xalt } d_2 \rrbracket \\ &= \llbracket d_1 \rrbracket \cup \llbracket d_2 \rrbracket \quad \text{by definition (10.38)} \\ &= \llbracket d_2 \rrbracket \cup \llbracket d_1 \rrbracket \quad \text{by commutativity of } \cup \\ &= \llbracket d_2 \text{ xalt } d_1 \rrbracket \quad \text{by definition (10.38)} \end{aligned}$$

Theorem 4 *Associativity of mandatory alternative*

$$P : (d_1 \text{ xalt } d_2) \text{ xalt } d_3 = d_1 \text{ xalt } (d_2 \text{ xalt } d_3)$$

P :

$$\begin{aligned} & \llbracket (d_1 \text{ xalt } d_2) \text{ xalt } d_3 \rrbracket \\ &= \llbracket d_1 \text{ xalt } d_2 \rrbracket \cup \llbracket d_3 \rrbracket \quad \text{by definition (10.38)} \\ &= (\llbracket d_1 \rrbracket \cup \llbracket d_2 \rrbracket) \cup \llbracket d_3 \rrbracket \quad \text{by definition (10.38)} \\ &= \llbracket d_1 \rrbracket \cup (\llbracket d_2 \rrbracket \cup \llbracket d_3 \rrbracket) \quad \text{by associativity of } \cup \\ &= \llbracket d_1 \rrbracket \cup \llbracket d_2 \text{ xalt } d_3 \rrbracket \quad \text{by definition (10.38)} \\ &= \llbracket d_1 \text{ xalt } (d_2 \text{ xalt } d_3) \rrbracket \quad \text{by definition (10.38)} \end{aligned}$$

Theorem 5 *Commutativity of parallel execution*

$$P : d_1 \text{ par } d_2 = d_2 \text{ par } d_1$$

P :

$$\begin{aligned} & \llbracket d_1 \text{ par } d_2 \rrbracket \\ &= \llbracket d_1 \rrbracket \parallel \llbracket d_2 \rrbracket \quad \text{by definition (10.39)} \\ &= \llbracket d_2 \rrbracket \parallel \llbracket d_1 \rrbracket \quad \text{by commutativity of } \parallel \text{ (lemma 22)} \\ &= \llbracket d_2 \text{ par } d_1 \rrbracket \quad \text{by definition (10.39)} \end{aligned}$$

Theorem 6 *Associativity of parallel execution*

$$P : (d_1 \text{ par } d_2) \text{ par } d_3 = d_1 \text{ par } (d_2 \text{ par } d_3)$$

P :

$$\begin{aligned} & \llbracket (d_1 \text{ par } d_2) \text{ par } d_3 \rrbracket \\ &= \llbracket d_1 \text{ par } d_2 \rrbracket \parallel \llbracket d_3 \rrbracket \quad \text{by definition (10.39)} \\ &= (\llbracket d_1 \rrbracket \parallel \llbracket d_2 \rrbracket) \parallel \llbracket d_3 \rrbracket \quad \text{by definition (10.39)} \\ &= \llbracket d_1 \rrbracket \parallel (\llbracket d_2 \rrbracket \parallel \llbracket d_3 \rrbracket) \quad \text{by lemma 23 (associativity of } \parallel \text{)} \\ &= \llbracket d_1 \rrbracket \parallel \llbracket d_2 \text{ par } d_3 \rrbracket \quad \text{by definition (10.39)} \\ &= \llbracket d_1 \text{ par } (d_2 \text{ par } d_3) \rrbracket \quad \text{by definition (10.39)} \end{aligned}$$

Theorem 7 *Associativity of weak sequencing*

$$P : \text{seq} [\text{seq} [d_1, d_2], d_3] = \text{seq} [d_1, \text{seq} [d_2, d_3]]$$

P :

$$\begin{aligned} & \llbracket \text{seq} [\text{seq} [d_1, d_2], d_3] \rrbracket \\ &= \llbracket \text{seq} [\text{seq} [d_1, d_2]] \rrbracket \succsim \llbracket d_3 \rrbracket \quad \text{by definition (10.36)} \\ &= \llbracket \text{seq} [d_1, d_2] \rrbracket \succsim \llbracket d_3 \rrbracket \quad \text{by definition (10.36)} \\ &= (\llbracket \text{seq} [d_1] \rrbracket \succsim \llbracket d_2 \rrbracket) \succsim \llbracket d_3 \rrbracket \quad \text{by definition (10.36)} \\ &= (\llbracket d_1 \rrbracket \succsim \llbracket d_2 \rrbracket) \succsim \llbracket d_3 \rrbracket \quad \text{by definition (10.36)} \\ &= \llbracket d_1 \rrbracket \succsim (\llbracket d_2 \rrbracket \succsim \llbracket d_3 \rrbracket) \quad \text{by lemma 24 (associativity of } \succsim \text{)} \\ &= \llbracket d_1 \rrbracket \succsim (\llbracket \text{seq} [d_2] \rrbracket \succsim \llbracket d_3 \rrbracket) \quad \text{by definition (10.36)} \\ &= \llbracket d_1 \rrbracket \succsim \llbracket \text{seq} [d_2, d_3] \rrbracket \quad \text{by definition (10.36)} \\ &= \llbracket \text{seq} [d_1] \rrbracket \succsim \llbracket \text{seq} [d_2, d_3] \rrbracket \quad \text{by definition (10.36)} \\ &= \llbracket \text{seq} [d_1, \text{seq} [d_2, d_3]] \rrbracket \quad \text{by definition (10.36)} \end{aligned}$$

10.D Reflexivity and Transitivity

In this section we prove that refinement as defined in this paper is reflexive and transitive.

10.D.1 Reflexivity

Lemma 25 *Reflexivity of \rightsquigarrow_r*

$$A : o = (p, n)$$

$$P : o \rightsquigarrow_r o$$

$\langle 1 \rangle$ 1. Requirement 1: $n \subseteq n$

P : Trivial.

$\langle 1 \rangle$ 2. Requirement 2: $p \subseteq p \cup n$

P : Trivial.

$\langle 1 \rangle$ 3. Q.E.D.

P : Definition (10.45) of \rightsquigarrow_r .

□

Theorem 8 *Reflexivity of the refinement operator \rightsquigarrow_g*

P : $d \rightsquigarrow_g d$,
 i.e. $\llbracket d \rrbracket \rightsquigarrow_g \llbracket d \rrbracket$ by definition (10.47),
 i.e. $\forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d \rrbracket : o \rightsquigarrow_r o'$ by definition (10.46).

$\langle 1 \rangle 1.$ $\exists o' \in \llbracket d \rrbracket : o \rightsquigarrow_r o'$ for arbitrary $o \in \llbracket d \rrbracket$

$\langle 2 \rangle 1.$ Choose $o' = o$
 P : $o \in \llbracket d \rrbracket$ by $\langle 1 \rangle 1.$

$\langle 2 \rangle 2.$ $o \rightsquigarrow_r o$
 P : Lemma 25 (reflexivity of \rightsquigarrow_r).

$\langle 2 \rangle 3.$ Q.E.D.

$\langle 1 \rangle 2.$ Q.E.D.

P : \forall -rule.

□

10.D.2 Transitivity

Lemma 26 *Transitivity of \rightsquigarrow_r*

A : 1. $(p, n) \rightsquigarrow_r (p', n')$

2. $(p', n') \rightsquigarrow_r (p'', n'')$

P : $(p, n) \rightsquigarrow_r (p'', n'')$

$\langle 1 \rangle 1.$ $n \subseteq n''$

$\langle 2 \rangle 1.$ $n \subseteq n'$

P : Assumption 1 and definition (10.45) of \rightsquigarrow_r .

$\langle 2 \rangle 2.$ $n' \subseteq n''$

P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .

$\langle 2 \rangle 3.$ Q.E.D.

P : $\langle 2 \rangle 1, \langle 2 \rangle 2$, and transitivity of \subseteq .

$\langle 1 \rangle 2.$ $p \subseteq p'' \cup n''$

$\langle 2 \rangle 1.$ $p \subseteq p' \cup n'$

P : Assumption 1 and definition (10.45) of \rightsquigarrow_r .

$\langle 2 \rangle 2.$ $p' \subseteq p'' \cup n''$

P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .

$\langle 2 \rangle 3.$ $n' \subseteq n''$

P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .

$\langle 2 \rangle 4.$ Q.E.D.

P : $\langle 2 \rangle 1, \langle 2 \rangle 2, \langle 2 \rangle 3$ and

$p \subseteq p' \cup n' \wedge p' \subseteq p'' \cup n'' \wedge n' \subseteq n''$

↓

$p \subseteq p'' \cup n' \cup n''$

↓

$p \subseteq p'' \cup n'' \cup n''$

↓

$p \subseteq p'' \cup n''$

$\langle 1 \rangle 3.$ Q.E.D.

P : By definition (10.45) of \rightsquigarrow_r .

□

Theorem 9 *Transitivity of the refinement operator \rightsquigarrow_g*

A : 1. $d \rightsquigarrow_g d'$

2. $d' \rightsquigarrow_g d''$

P : $d \rightsquigarrow_g d''$

$\langle 1 \rangle 1.$ $\llbracket d \rrbracket \rightsquigarrow_g \llbracket d'' \rrbracket$

$\langle 2 \rangle 1.$ $\forall o \in \llbracket d \rrbracket : \exists o'' \in \llbracket d'' \rrbracket : o \rightsquigarrow_r o''$

$\langle 3 \rangle 1.$ $\forall (p, n) \in \llbracket d \rrbracket : \exists (p'', n'') \in \llbracket d'' \rrbracket : (p, n) \rightsquigarrow_r (p'', n'')$

$\langle 4 \rangle 1.$ $\exists (p'', n'') \in \llbracket d'' \rrbracket : (p, n) \rightsquigarrow_r (p'', n'')$ for arbitrary $(p, n) \in \llbracket d \rrbracket$

$\langle 5 \rangle 1.$ Choose $(p', n') \in \llbracket d' \rrbracket$ and $(p'', n'') \in \llbracket d'' \rrbracket$ such that

1. $(p, n) \rightsquigarrow_r (p', n')$

2. $(p', n') \rightsquigarrow_r (p'', n'')$

P : Assumptions 1 and 2, and definitions (10.47) and (10.46) of \rightsquigarrow_g .

$\langle 5 \rangle 2.$ $(p, n) \rightsquigarrow_r (p'', n'')$

P : $\langle 5 \rangle 1$ and lemma 26 (transitivity of \rightsquigarrow_r).

$\langle 5 \rangle 3.$ Q.E.D.

$\langle 4 \rangle 2.$ Q.E.D.

P : \forall -rule.

$\langle 3 \rangle 2.$ Q.E.D.

$\langle 2 \rangle 2.$ Q.E.D.

P : By definition (10.46) of \rightsquigarrow_g .

$\langle 1 \rangle 2.$ Q.E.D.

P : By definition (10.47) of \rightsquigarrow_g .

□

10.E Monotonicity

In this section we prove that the refinement operator \rightsquigarrow_g is monotonic with respect to the composition operators neg, alt, xalt, seq, loop, par and tc, meaning that refining one operand will give a refinement of the whole composition.

In general, we do not have monotonicity with respect to assert. However, we prove that in the special case of narrowing, written $\rightsquigarrow_{g,n}$, we do also have monotonicity with respect to assert.

First, in Section 10.E.1 we prove monotonicity of \rightsquigarrow_r with respect to weak sequencing, parallel execution and time constraint on trace sets. Then, in Section 10.E.2 we prove monotonicity of \rightsquigarrow_g with respect to weak sequencing, parallel execution, time constraint, inner union and looping on sets of interaction obligations. Finally, in Section 10.E.3 we prove the monotonicity theorems for \rightsquigarrow_g with respect to the sequence diagram operators.

10.E.1 Monotonicity of \rightsquigarrow_r

Lemma 27 (*To be used when proving monotonicity with respect to \rightsquigarrow*)

A : 1. $s_1 \subseteq s'_1$

2. $s_2 \subseteq s'_2$

P : $s_1 \rightsquigarrow s_2 \subseteq s'_1 \rightsquigarrow s'_2$

$\langle 1 \rangle 1.$ C : $s_1 \rightsquigarrow s_2 = \emptyset$

P : Trivial, as $\emptyset \subseteq A$ for all sets A .

$\langle 1 \rangle 2.$ C : $s_1 \rightsquigarrow s_2 \neq \emptyset$

$\langle 2 \rangle 1.$ Choose arbitrary $h \in s_1 \rightsquigarrow s_2$

P : Non-empty by case assumption.

$\langle 2 \rangle 2.$ $h \in s'_1 \rightsquigarrow s'_2$

$\langle 3 \rangle 1.$ Choose $h_1 \in s_1$ and $h_2 \in s_2$ such that $\forall l \in \mathcal{L} : e.l @ h = e.l @ h_1 \cap e.l @ h_2$

P : $\langle 2 \rangle 1$ and definition (10.13) of \rightsquigarrow .

$\langle 3 \rangle 2.$ $h_1 \in s'_1$

P : $\langle 3 \rangle 1$ and assumption 1.

$\langle 3 \rangle 3.$ $h_2 \in s'_2$

P : $\langle 3 \rangle 1$ and assumption 2.

$\langle 3 \rangle 4.$ $h \in s'_1 \rightsquigarrow s'_2$

P : $\langle 3 \rangle 1, \langle 3 \rangle 2, \langle 3 \rangle 3$ and definition (10.13) of \rightsquigarrow .

$\langle 3 \rangle 5.$ Q.E.D.

$\langle 2 \rangle 3.$ Q.E.D.

P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition of \subseteq .

$\langle 1 \rangle 3.$ Q.E.D.

P : The cases are exhaustive.

□

Lemma 28 (*To be used when proving monotonicity with respect to \parallel*)

A : 1. $s_1 \subseteq s'_1$

2. $s_2 \subseteq s'_2$

P : $s_1 \parallel s_2 \subseteq s'_1 \parallel s'_2$

$\langle 1 \rangle 1.$ C : $s_1 \rightsquigarrow s_2 = \emptyset$

P : Trivial, as $\emptyset \subseteq A$ for all sets A .

$\langle 1 \rangle 2.$ C : $s_1 \parallel s_2 \neq \emptyset$

$\langle 2 \rangle 1.$ Choose arbitrary $h \in s_1 \parallel s_2$

P : Non-empty by case assumption.

$\langle 2 \rangle 2.$ $h \in s'_1 \parallel s'_2$

$\langle 3 \rangle 1.$ Choose $p \in \{1, 2\}^\infty$ such that

$\pi_2((\{1\} \times [\mathcal{E}]) @ (p, h)) \in s_1$ and

$\pi_2((\{2\} \times [\mathcal{E}]) @ (p, h)) \in s_2$

P : $\langle 2 \rangle 1$ and definition (10.12) of \parallel .

$\langle 3 \rangle 2.$ $\pi_2((\{1\} \times [\mathcal{E}]) @ (p, h)) \in s'_1$

P : $\langle 3 \rangle 1$ and assumption 1.

$\langle 3 \rangle 3.$ $\pi_2((\{2\} \times [\mathcal{E}]) @ (p, h)) \in s'_2$

P : $\langle 3 \rangle 1$ and assumption 2.
 $\langle 3 \rangle 4.$ $h \in s'_1 \| s'_2$
P : $\langle 3 \rangle 1, \langle 3 \rangle 2, \langle 3 \rangle 3$ and definition (10.12) of $\|$.
 $\langle 3 \rangle 5.$ Q.E.D.
 $\langle 2 \rangle 3.$ Q.E.D.
P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition of \subseteq .
 $\langle 1 \rangle 3.$ Q.E.D.
P : The cases are exhaustive.

□

Lemma 29 (*To be used when proving monotonicity with respect to \setminus*)

A : $s \subseteq s'$
P : $s \setminus C \subseteq s' \setminus C$
 $\langle 1 \rangle 1.$ C : $s \setminus C = \emptyset$
P : Trivial, as $\emptyset \subseteq A$ for all sets A .
 $\langle 1 \rangle 2.$ C : $s \setminus C \neq \emptyset$
 $\langle 2 \rangle 1.$ Choose arbitrary $h \in s \setminus C$
P : Non-empty by case assumption.
 $\langle 2 \rangle 2.$ $h \in s' \setminus C$
 $\langle 3 \rangle 1.$ $h \in \{h \in s \mid h \models C\}$
P : $\langle 2 \rangle 1$ and definition (10.14) of \models .
 $\langle 3 \rangle 2.$ $h \in \{h \in s' \mid h \models C\}$
P : $\langle 3 \rangle 1$ and the assumption.
 $\langle 3 \rangle 3.$ Q.E.D.
P : $\langle 3 \rangle 2$ and definition (10.14) of \models .
 $\langle 3 \rangle 4.$ Q.E.D.
 $\langle 2 \rangle 3.$ Q.E.D.
P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition of \subseteq .
 $\langle 1 \rangle 3.$ Q.E.D.
P : The cases are exhaustive.

□

Lemma 30 *Monotonicity of \rightsquigarrow_r with respect to \succsim*

A : 1. $(p, n) = (p_1, n_1) \succsim (p_2, n_2)$,
i.e. $(p, n) = (p_1 \succsim p_2, (n_1 \succsim (n_2 \cup p_2)) \cup (p_1 \succsim n_2))$ by definition (10.16).
2. $(p_1, n_1) \rightsquigarrow_r (p'_1, n'_1)$
3. $(p_2, n_2) \rightsquigarrow_r (p'_2, n'_2)$
4. $(p', n') = (p'_1, n'_1) \succsim (p'_2, n'_2)$,
i.e. $(p', n') = (p'_1 \succsim p'_2, (n'_1 \succsim (n'_2 \cup p'_2)) \cup (p'_1 \succsim n'_2))$ by definition (10.16).
P : $(p, n) \rightsquigarrow_r (p', n')$
 $\langle 1 \rangle 1.$ Requirement 1: $n \subseteq n'$,
i.e. $(n_1 \succsim (n_2 \cup p_2)) \cup (p_1 \succsim n_2) \subseteq (n'_1 \succsim (n'_2 \cup p'_2)) \cup (p'_1 \succsim n'_2)$
 $\langle 2 \rangle 1.$ $n_1 \succsim n_2 \subseteq n'_1 \succsim n'_2$

$\langle 3 \rangle 1. n_1 \subseteq n'_1$

P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .

$\langle 3 \rangle 2. n_2 \subseteq n'_2$

P : Assumption 3 and definition (10.45) of \rightsquigarrow_r .

$\langle 3 \rangle 3. Q.E.D.$

P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and lemma 27.

$\langle 2 \rangle 2. n_1 \succsim p_2 \subseteq n'_1 \succsim (n'_2 \cup p'_2)$

$\langle 3 \rangle 1. n_1 \subseteq n'_1$

P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .

$\langle 3 \rangle 2. p_2 \subseteq p'_2 \cup n'_2$

P : Assumption 3 and definition (10.45) of \rightsquigarrow_r .

$\langle 3 \rangle 3. n_1 \succsim p_2 \subseteq n'_1 \succsim (p'_2 \cup n'_2)$

P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and lemma 27.

$\langle 3 \rangle 4. Q.E.D.$

P : $\langle 3 \rangle 3$ and associativity of \cup .

$\langle 2 \rangle 3. p_1 \succsim n_2 \subseteq (p'_1 \succsim n'_2) \cup (n'_1 \succsim n'_2)$

$\langle 3 \rangle 1. p_1 \subseteq p'_1 \cup n'_1$

P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .

$\langle 3 \rangle 2. n_2 \subseteq n'_2$

P : Assumption 3 and definition (10.45) of \rightsquigarrow_r .

$\langle 3 \rangle 3. p_1 \succsim n_2 \subseteq (p'_1 \cup n'_1) \succsim n'_2$

P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and lemma 27.

$\langle 3 \rangle 4. Q.E.D.$

P : $\langle 3 \rangle 3$ and lemma 15 (distributivity of \succsim over \cup).

$\langle 2 \rangle 4. Q.E.D.$

P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and $\langle 2 \rangle 3$ and lemma 14 (distributivity of \succsim over \cup).

$\langle 1 \rangle 2. \text{Requirement 2: } p \subseteq p' \cup n'$,

i.e. $p_1 \succsim p_2 \subseteq (p'_1 \succsim p'_2) \cup (n'_1 \succsim (n'_2 \cup p'_2)) \cup (p'_1 \succsim n'_2)$

$\langle 2 \rangle 1. p_1 \subseteq p'_1 \cup n'_1$

P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .

$\langle 2 \rangle 2. p_2 \subseteq p'_2 \cup n'_2$

P : Assumption 3 and definition (10.45) of \rightsquigarrow_r .

$\langle 2 \rangle 3. p_1 \succsim p_2 \subseteq (p'_1 \cup n'_1) \succsim (p'_2 \cup n'_2)$

P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and lemma 27.

$\langle 2 \rangle 4. Q.E.D.$

P : $\langle 2 \rangle 3$ and lemmas 14 and 15 (distributivity of \succsim over \cup).

$\langle 1 \rangle 3. Q.E.D.$

P : Assumptions 1 and 4 and definition (10.45) of \rightsquigarrow_r .

□

Lemma 31 *Monotonicity of \rightsquigarrow_r with respect to \parallel*

A : 1. $(p, n) = (p_1, n_1) \parallel (p_2, n_2)$,

i.e. $(p, n) = (p_1 \parallel p_2, (n_1 \parallel (p_2 \cup n_2)) \cup (n_2 \parallel p_1))$ by definition (10.15).

2. $(p_1, n_1) \rightsquigarrow_r (p'_1, n'_1)$

3. $(p_2, n_2) \rightsquigarrow_r (p'_2, n'_2)$
4. $(p', n') = (p'_1, n'_1) \parallel (p'_2, n'_2)$,
i.e. $(p', n') = (p'_1 \parallel p'_2, (n'_1 \succsim (p'_2 \cup n'_2)) \cup (n'_2 \parallel p'_1))$ by definition (10.15).
- P : $(p, n) \rightsquigarrow_r (p', n')$
- $\langle 1 \rangle 1.$ Requirement 1: $n \subseteq n'$,
i.e. $(n_1 \parallel (p_2 \cup n_2)) \cup (n_2 \parallel p_1) \subseteq (n'_1 \parallel (p'_2 \cup n'_2)) \cup (n'_2 \parallel p'_1)$
- $\langle 2 \rangle 1.$ $n_1 \parallel n_2 \subseteq n'_1 \parallel n'_2$
- $\langle 3 \rangle 1.$ $n_1 \subseteq n'_1$
P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .
- $\langle 3 \rangle 2.$ $n_2 \subseteq n'_2$
P : Assumption 3 and definition (10.45) of \rightsquigarrow_r .
- $\langle 3 \rangle 3.$ Q.E.D.
P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and lemma 28.
- $\langle 2 \rangle 2.$ $n_1 \parallel p_2 \subseteq n'_1 \parallel (p'_2 \cup n'_2)$
- $\langle 3 \rangle 1.$ $n_1 \subseteq n'_1$
P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .
- $\langle 3 \rangle 2.$ $p_2 \subseteq p'_2 \cup n'_2$
P : Assumption 3 and definition (10.45) of \rightsquigarrow_r .
- $\langle 3 \rangle 3.$ Q.E.D.
P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and lemma 27.
- $\langle 2 \rangle 3.$ $n_2 \parallel p_1 \subseteq (n'_2 \parallel p'_1) \cup (n'_2 \parallel n'_1)$
- $\langle 3 \rangle 1.$ $n_2 \subseteq n'_2$
P : Assumption 3 and definition (10.45) of \rightsquigarrow_r .
- $\langle 3 \rangle 2.$ $p_1 \subseteq p'_1 \cup n'_1$
P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .
- $\langle 3 \rangle 3.$ $n_2 \parallel p_1 \subseteq n'_2 \parallel (p'_1 \cup n'_1)$
P : $\langle 3 \rangle 2, \langle 3 \rangle 1$ and lemma 28.
- $\langle 3 \rangle 4.$ Q.E.D.
P : $\langle 3 \rangle 3$ and lemma 15 (distributivity of \succsim over \cup).
- $\langle 2 \rangle 4.$ Q.E.D.
P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and $\langle 2 \rangle 3$, lemma 3 (commutativity of \parallel) and lemma 12 (distributivity of \parallel over \cup).
- $\langle 1 \rangle 2.$ Requirement 2: $p \subseteq p' \cup n'$,
i.e. $p_1 \parallel p_2 \subseteq (p'_1 \parallel p'_2) \cup (n'_1 \parallel (p'_2 \cup n'_2)) \cup (n'_2 \parallel p'_1)$
- $\langle 2 \rangle 1.$ $p_1 \subseteq p'_1 \cup n'_1$
P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .
- $\langle 2 \rangle 2.$ $p_2 \subseteq p'_2 \cup n'_2$
P : Assumption 3 and definition (10.45) of \rightsquigarrow_r .
- $\langle 2 \rangle 3.$ $p_1 \parallel p_2 \subseteq (p'_1 \cup n'_1) \succsim (p'_2 \parallel n'_2)$
P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and lemma 28.
- $\langle 2 \rangle 4.$ Q.E.D.
P : $\langle 2 \rangle 3$, lemma 3 (commutativity of \parallel) and lemmas 12 and 13 (distributivity of \parallel over \cup).
- $\langle 1 \rangle 3.$ Q.E.D.

P : Assumptions 1 and 4 and definition (10.45) of \rightsquigarrow_r .

□

Lemma 32 *Monotonicity of \rightsquigarrow_r with respect to \wr*

A : 1. $(p, n) = (p_1, n_1) \wr C$,
 i.e. $(p, n) = (p_1 \wr C, n_1 \cup (p_1 \wr \neg C))$ by definition (10.17).

2. $(p_1, n_1) \rightsquigarrow_r (p'_1, n'_1)$

3. $(p', n') = (p'_1, n'_1) \wr C$,

i.e. $(p', n') = (p'_1 \wr C, n'_1 \cup (p'_1 \wr \neg C))$ by definition (10.17).

P : $(p, n) \rightsquigarrow_r (p', n')$

$\langle 1 \rangle$ 1. Requirement 1: $n \subseteq n'$,

i.e. $n_1 \cup (p_1 \wr \neg C) \subseteq n'_1 \cup (p'_1 \wr \neg C)$

$\langle 2 \rangle$ 1. $n_1 \subseteq n'_1$

P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .

$\langle 2 \rangle$ 2. $p_1 \wr \neg C \subseteq n'_1 \cup (p'_1 \wr \neg C)$

$\langle 3 \rangle$ 1. $p_1 \subseteq p'_1 \cup n'_1$

P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .

$\langle 3 \rangle$ 2. $p_1 \wr \neg C \subseteq (p'_1 \cup n'_1) \wr \neg C$

P : $\langle 3 \rangle$ 1 and lemma 29.

$\langle 3 \rangle$ 3. $p_1 \wr \neg C \subseteq (p'_1 \wr \neg C) \cup (n'_1 \wr \neg C)$

P : $\langle 3 \rangle$ 2 and lemma 16 (distributivity of \wr over \cup).

$\langle 3 \rangle$ 4. $n'_1 \wr \neg C \subseteq n'_1$

P : Definition (10.14) of \wr .

$\langle 3 \rangle$ 5. Q.E.D.

P : $\langle 3 \rangle$ 3, $\langle 3 \rangle$ 4 and associativity of \cup .

$\langle 2 \rangle$ 3. Q.E.D.

P : $\langle 2 \rangle$ 1 and $\langle 2 \rangle$ 2.

$\langle 1 \rangle$ 2. Requirement 2: $p \subseteq p' \cup n'$,

i.e. $p_1 \wr C \subseteq (p'_1 \wr C) \cup (n'_1 \cup (p'_1 \wr \neg C))$

$\langle 2 \rangle$ 1. $p_1 \subseteq p'_1 \cup n'_1$

P : Assumption 2 and definition (10.45) of \rightsquigarrow_r .

$\langle 2 \rangle$ 2. $p_1 \wr C \subseteq (p'_1 \cup n'_1) \wr C$

P : $\langle 2 \rangle$ 1 and lemma 29.

$\langle 2 \rangle$ 3. $p_1 \wr C \subseteq (p'_1 \wr C) \cup (n'_1 \wr C)$

P : $\langle 2 \rangle$ 3 and lemma 16 (distributivity of \wr over \cup).

$\langle 2 \rangle$ 4. $n'_1 \wr C \subseteq n'_1$

P : Definition (10.14) of \wr .

$\langle 2 \rangle$ 5. Q.E.D.

P : $\langle 2 \rangle$ 3 and $\langle 2 \rangle$ 4.

$\langle 1 \rangle$ 3. Q.E.D.

P : Assumptions 1 and 3 and definition (10.45) of \rightsquigarrow_r .

□

10.E.2 Monotonicity of \rightsquigarrow_g with Respect to Operators on Sets of Interaction Obligations

Lemma 33 *Monotonicity of \rightsquigarrow_g with respect to \succsim on sets of interaction obligations*

A : 1. $O = O_1 \succsim O_2$,
i.e. $O = \{o_1 \succsim o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\}$ by definition (10.19).

2. $O_1 \rightsquigarrow_g O'_1$,
i.e. $\forall o_1 \in O_1 : \exists o'_1 \in O'_1 : o_1 \rightsquigarrow_r o'_1$ by definition (10.46).

3. $O_2 \rightsquigarrow_g O'_2$,
i.e. $\forall o_2 \in O_2 : \exists o'_2 \in O'_2 : o_2 \rightsquigarrow_r o'_2$ by definition (10.46).

4. $O' = O'_1 \succsim O'_2$,
i.e. $O' = \{o_1 \succsim o_2 \mid o_1 \in O'_1 \wedge o_2 \in O'_2\}$ by definition (10.19).

P : $O \rightsquigarrow_g O'$,
i.e. $O_1 \succsim O_2 \rightsquigarrow_g O'_1 \succsim O'_2$,
i.e. $\forall o \in O_1 \succsim O_2 : \exists o' \in O'_1 \succsim O'_2 : o \rightsquigarrow_r o'$ by definition (10.46).

$\langle 1 \rangle 1.$ $\exists o' \in O'_1 \succsim O'_2 : o \rightsquigarrow_r o'$ for arbitrary $o \in O_1 \succsim O_2$

$\langle 2 \rangle 1.$ Choose $o_1 \in O_1$ and $o_2 \in O_2$ such that $o = o_1 \succsim o_2$

P : Assumption 1.

$\langle 2 \rangle 2.$ Choose $o'_1 \in O'_1$ such that $o_1 \rightsquigarrow_r o'_1$

P : Assumption 2.

$\langle 2 \rangle 3.$ Choose $o'_2 \in O'_2$ such that $o_2 \rightsquigarrow_r o'_2$

P : Assumption 3.

$\langle 2 \rangle 4.$ $o' = o'_1 \succsim o'_2 \in O'_1 \succsim O'_2$

P : Assumption 4.

$\langle 2 \rangle 5.$ $o \rightsquigarrow_r o'$, i.e. $o_1 \succsim o_2 \rightsquigarrow_r o'_1 \succsim o'_2$

P : Lemma 30 with $(p, n) = o$, $(p', n') = o'$, $(p_1, n_1) = o_1$,
 $(p_2, n_2) = o_2$, $(p'_1, n'_1) = o'_1$ and $(p'_2, n'_2) = o'_2$.

$\langle 2 \rangle 6.$ Q.E.D.

$\langle 1 \rangle 2.$ Q.E.D.

P : \forall -rule.

□

Lemma 34 *Monotonicity of \rightsquigarrow_g with respect to \parallel on sets of interaction obligations*

A : 1. $O = O_1 \parallel O_2$,
i.e. $O = \{o_1 \parallel o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\}$ by definition (10.18).

2. $O_1 \rightsquigarrow_g O'_1$,
i.e. $\forall o_1 \in O_1 : \exists o'_1 \in O'_1 : o_1 \rightsquigarrow_r o'_1$ by definition (10.46).

3. $O_2 \rightsquigarrow_g O'_2$,
i.e. $\forall o_2 \in O_2 : \exists o'_2 \in O'_2 : o_2 \rightsquigarrow_r o'_2$ by definition (10.46).

4. $O' = O'_1 \parallel O'_2$,
i.e. $O' = \{o_1 \parallel o_2 \mid o_1 \in O'_1 \wedge o_2 \in O'_2\}$ by definition (10.18).

P : $O \rightsquigarrow_g O'$,
 i.e. $O_1 \parallel O_2 \rightsquigarrow_g O'_1 \parallel O'_2$,
 i.e. $\forall o \in O_1 \parallel O_2 : \exists o' \in O'_1 \parallel O'_2 : o \rightsquigarrow_r o'$ by definition (10.46).

- $\langle 1 \rangle 1.$ $\exists o' \in O'_1 \parallel O'_2 : o \rightsquigarrow_r o'$ for arbitrary $o \in O_1 \parallel O_2$
 $\langle 2 \rangle 1.$ Choose $o_1 \in O_1$ and $o_2 \in O_2$ such that $o = o_1 \parallel o_2$

P : Assumption 1.

- $\langle 2 \rangle 2.$ Choose $o'_1 \in O'_1$ such that $o_1 \rightsquigarrow_r o'_1$

P : Assumption 2.

- $\langle 2 \rangle 3.$ Choose $o'_2 \in O'_2$ such that $o_2 \rightsquigarrow_r o'_2$

P : Assumption 3.

- $\langle 2 \rangle 4.$ $o' = o'_1 \parallel o'_2 \in O'_1 \parallel O'_2$

P : Assumption 4.

- $\langle 2 \rangle 5.$ $o \rightsquigarrow_r o'$, i.e. $o_1 \parallel o_2 \rightsquigarrow_r o'_1 \parallel o'_2$

P : Lemma 31 with $(p, n) = o$, $(p', n') = o'$, $(p_1, n_1) = o_1$,
 $(p_2, n_2) = o_2$, $(p'_1, n'_1) = o'_1$ and $(p'_2, n'_2) = o'_2$.

- $\langle 2 \rangle 6.$ Q.E.D.

- $\langle 1 \rangle 2.$ Q.E.D.

P : \forall -rule.

□

Lemma 35 *Monotonicity of \rightsquigarrow_g with respect to \wr on sets of interaction obligations*

A : 1. $O = O_1 \wr C$,
 i.e. $O = \{o_1 \wr C \mid o_1 \in O_1\}$ by definition (10.20).
 2. $O_1 \rightsquigarrow_g O'_1$,
 i.e. $\forall o_1 \in O_1 : \exists o'_1 \in O'_1 : o_1 \rightsquigarrow_r o'_1$ by definition (10.46).
 3. $O' = O'_1 \wr C$,
 i.e. $O' = \{o_1 \wr C \mid o_1 \in O'_1\}$ by definition (10.20).

P : $O \rightsquigarrow_g O'$,
 i.e. $O_1 \wr C \rightsquigarrow_g O'_1 \wr C$,
 i.e. $\forall o \in O_1 \wr C : \exists o' \in O'_1 \wr C : o \rightsquigarrow_r o'$ by definition (10.46).

- $\langle 1 \rangle 1.$ $\exists o' \in O'_1 \wr C : o \rightsquigarrow_r o'$ for arbitrary $o \in O_1 \wr C$

- $\langle 2 \rangle 1.$ Choose $o_1 \in O_1$ such that $o = o_1 \wr C$

P : Assumption 1.

- $\langle 2 \rangle 2.$ Choose $o'_1 \in O'_1$ such that $o_1 \rightsquigarrow_r o'_1$

P : Assumption 2.

- $\langle 2 \rangle 3.$ $o' = o'_1 \wr C \in O'_1 \wr C$

P : Assumption 3.

- $\langle 2 \rangle 4.$ $o \rightsquigarrow_r o'$, i.e. $o_1 \wr C \rightsquigarrow_r o'_1 \wr C$

P : Lemma 32 with $(p, n) = o$, $(p', n') = o'$, $(p_1, n_1) = o_1$, and $(p'_1, n'_1) = o'_1$.

- $\langle 2 \rangle 5.$ Q.E.D.

- $\langle 1 \rangle 2.$ Q.E.D.

P : \forall -rule.

□

Lemma 36 *Monotonicity of \rightsquigarrow_g with respect to \uplus on sets of interaction obligations*

- A : 1. $O = O_1 \uplus O_2$,
 i.e. $O = \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2\}$ by definition (10.30).
2. $O_1 \rightsquigarrow_g O'_1$,
 i.e. $\forall o_1 \in O_1 : \exists o'_1 \in O'_1 : o_1 \rightsquigarrow_r o'_1$ by definition (10.46).
3. $O_2 \rightsquigarrow_g O'_2$,
 i.e. $\forall o_2 \in O_2 : \exists o'_2 \in O'_2 : o_2 \rightsquigarrow_r o'_2$ by definition (10.46).
4. $O' = O'_1 \uplus O'_2$,
 i.e. $O' = \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O'_1 \wedge (p_2, n_2) \in O'_2\}$ by definition (10.30).
- P : $O \rightsquigarrow_g O'$,
 i.e. $\forall o \in O : \exists o' \in O' : o \rightsquigarrow_r o'$ by definition (10.46).
- $\langle 1 \rangle$ 1. $\exists o' \in O' : o \rightsquigarrow_r o'$ for arbitrary $o = (p, n) \in O$
- $\langle 2 \rangle$ 1. Choose $(p_1, n_1) \in O_1$ and $(p_2, n_2) \in O_2$ such that $p = p_1 \cup p_2$ and $n = n_1 \cup n_2$
 P : Assumption 1.
- $\langle 2 \rangle$ 2. Choose $(p'_1, n'_1) \in O'_1$ such that $(p_1, n_1) \rightsquigarrow_r (p'_1, n'_1)$
 P : Assumption 2.
- $\langle 2 \rangle$ 3. Choose $(p'_2, n'_2) \in O'_2$ such that $(p_2, n_2) \rightsquigarrow_r (p'_2, n'_2)$
 P : Assumption 3.
- $\langle 2 \rangle$ 4. $o' = (p', n') = (p'_1 \cup p'_2, n'_1 \cup n'_2) \in O'$
 P : Assumption 4.
- $\langle 2 \rangle$ 5. $(p, n) \rightsquigarrow_r (p', n')$
- $\langle 3 \rangle$ 1. Requirement 1: $n \subseteq n'$, i.e. $n_1 \cup n_2 \subseteq n'_1 \cup n'_2$
 P : $n_1 \subseteq n'_1$ (by $\langle 2 \rangle$ 2 and definition (10.45) of \rightsquigarrow_r) and $n_2 \subseteq n'_2$ (by $\langle 2 \rangle$ 3 and definition (10.45) of \rightsquigarrow_r).
- $\langle 3 \rangle$ 2. Requirement 2: $p \subseteq p' \cup n'$, i.e. $p_1 \cup p_2 \subseteq (p'_1 \cup p'_2) \cup (n'_1 \cup n'_2)$
 P : $p_1 \subseteq p'_1 \cup n'_1$ (by $\langle 2 \rangle$ 2 and definition (10.45) of \rightsquigarrow_r) and $p_2 \subseteq p'_2 \cup n'_2$ (by $\langle 2 \rangle$ 3 and definition (10.45) of \rightsquigarrow_r).
- $\langle 3 \rangle$ 3. Q.E.D.
 P : Definition (10.45) of \rightsquigarrow_r .
- $\langle 2 \rangle$ 6. Q.E.D.
- $\langle 1 \rangle$ 2. Q.E.D.
 P : \forall -rule.

□

Lemma 37 *Monotonicity of \rightsquigarrow_g with respect to \uplus on sets of interaction obligations*

- A : 1. $O = \biguplus_{i \in I} O_i$,
 i.e. $O = \{\biguplus_{i \in I} o_i \mid o_i \in O_i\}$ by definition (10.31),
 i.e. $O = \{(\bigcup_{i \in I} p_i, \bigcup_{i \in I} n_i) \mid (p_i, n_i) \in O_i\}$ by definition (10.32).
2. $\forall i \in I : O_i \rightsquigarrow_g O'_i$,
 i.e. $\forall i \in I : \forall o \in O_i : \exists o' \in O'_i : o \rightsquigarrow_r o'$ by definition (10.46).

3. $O' = \biguplus_{i \in I} O'_i$,
 i.e. $O' = \{\biguplus_{i \in I} o_i \mid o_i \in O'_i\}$ by definition (10.31),
 i.e. $O' = \{(\bigcup_{i \in I} p_i, \bigcup_{i \in I} n_i) \mid (p_i, n_i) \in O'_i\}$ by definition (10.32).
- P : $O \rightsquigarrow_g O'$,
 i.e. $\forall o \in O : \exists o' \in O' : o \rightsquigarrow_r o'$ by definition (10.46).
- $\langle 1 \rangle 1.$ $\exists o' \in O' : o \rightsquigarrow_r o'$ for arbitrary $o = (p, n) \in O$
- $\langle 2 \rangle 1.$ For all $i \in I$, choose $(p_i, n_i) \in O_i$ such that $p = \bigcup_{i \in I} p_i$ and
 $n = \bigcup_{i \in I} n_i$
- P : Assumption 1.
- $\langle 2 \rangle 2.$ For all $i \in I$, choose $(p'_i, n'_i) \in O'_i$ such that $(p_i, n_i) \rightsquigarrow_r (p'_i, n'_i)$
- P : Assumption 2.
- $\langle 2 \rangle 3.$ $o' = (p', n') = (\bigcup_{i \in I} p'_i, \bigcup_{i \in I} n'_i) \in O'$
- P : Assumption 3.
- $\langle 2 \rangle 4.$ $(p, n) \rightsquigarrow_r (p', n')$
- $\langle 3 \rangle 1.$ Requirement 1: $n \subseteq n'$, i.e. $\bigcup_{i \in I} n_i \subseteq \bigcup_{i \in I} n'_i$
 P : $\forall i \in I : n_i \subseteq n'_i$ by $\langle 2 \rangle 2$ and definition (10.45) of \rightsquigarrow_r .
- $\langle 3 \rangle 2.$ Requirement 2: $p \subseteq p' \cup n'$,
 i.e. $\bigcup_{i \in I} p_i \subseteq \bigcup_{i \in I} p'_i \cup \bigcup_{i \in I} n'_i$
 P : $\forall i \in I : p_i \subseteq p'_i \cup n'_i$ by $\langle 2 \rangle 2$ and definition (10.45) of \rightsquigarrow_r .
- $\langle 3 \rangle 3.$ Q.E.D.
- P : Definition (10.45) of \rightsquigarrow_r .
- $\langle 1 \rangle 2.$ Q.E.D.
- P : \forall -rule.

□

Lemma 38 *Monotonicity of \rightsquigarrow_g with respect to the inductive definition of μ_n*

- A : $O \rightsquigarrow_g O'$,
 i.e. $\forall o \in O : \exists o' \in O' : o \rightsquigarrow_r o'$ by definition (10.46).
- P : $\mu_n O \rightsquigarrow_g \mu_n O'$,
 i.e. $\forall o \in \mu_n O : \exists o' \in \mu_n O' : o \rightsquigarrow_r o'$ by definition (10.46).
- $\langle 1 \rangle 1.$ $\exists o' \in \mu_n O' : o \rightsquigarrow_r o'$ for arbitrary $o \in \mu_n O$
- $\langle 2 \rangle 1.$ C : $n = 0$
- $\langle 3 \rangle 1.$ $\mu_0 O = \{(\{\langle\rangle\}, \emptyset)\}$, i.e. $o = (\{\langle\rangle\}, \emptyset)$
 P : Definition (10.26) of μ_0 .
- $\langle 3 \rangle 2.$ $o = (\{\langle\rangle\}, \emptyset) \in \mu_0 O'$
 P : Definition (10.26) of μ_0 .
- $\langle 3 \rangle 3.$ $o \rightsquigarrow_r o$
 P : Lemma 25 (reflexivity of \rightsquigarrow_r).
- $\langle 3 \rangle 4.$ Q.E.D.
- $\langle 2 \rangle 2.$ C : $n = 1$
- $\langle 3 \rangle 1.$ $o \in O$
 P : Definition (10.27) of μ_1 .
- $\langle 3 \rangle 2.$ Choose $o' \in O'$ such that $o \rightsquigarrow_r o'$
 P : $\langle 3 \rangle 1$ and the assumption.

$\langle 3 \rangle 3. o' \in \mu_1 O'$

P : $\langle 3 \rangle 2$ and definition (10.27) of μ_1 .

$\langle 3 \rangle 4. Q.E.D.$

$\langle 2 \rangle 3. C : 1 < n < \infty$

$\langle 3 \rangle 1. A : \mu_k O \rightsquigarrow_g \mu_k O'$ (induction hypothesis).

P : $\mu_{k+1} O \rightsquigarrow_g \mu_{k+1} O'$,

i.e. $O \succsim \mu_k O \rightsquigarrow_g O' \succsim \mu_k O'$ by definition (10.28) of μ_n .

$\langle 4 \rangle 1. O \rightsquigarrow_g O'$

P : The main assumption.

$\langle 4 \rangle 2. \mu_k O \rightsquigarrow_g \mu_k O'$

P : The induction hypothesis.

$\langle 4 \rangle 3. Q.E.D.$

P : Lemma 33 (monotonicity of \rightsquigarrow_g with respect to \succsim) with

$O_1 = O, O_2 = \mu_k O, O'_1 = O' \text{ and } O'_2 = \mu_k O'$.

$\langle 3 \rangle 2. Q.E.D.$

P : Standard induction on n , base cases proved by $\langle 2 \rangle 1$ and $\langle 2 \rangle 2$.

$\langle 2 \rangle 4. C : n = \infty$

P : For each chain $\bar{o} \in \text{chains}(O)$, we may construct a chain \bar{o}' such that $\forall j \in \mathbb{N} : \bar{o}[j] \rightsquigarrow_r \bar{o}'[j]$ due to the lemma assumption. Each trace in the chains $\text{pos}(\bar{o})$ and $\text{negs}(\bar{o})$ is selected from a corresponding obligation in the chain \bar{o} . By definition (10.45) of \rightsquigarrow_r , these traces are also contained (as positive or negative) in the corresponding obligations in \bar{o}' .

$\langle 3 \rangle 1. \exists o' \in \{\sqcup \bar{o}' \mid \bar{o}' \in \text{chains}(O')\} : o \rightsquigarrow_r o'$ for arbitrary
 $o \in \{\sqcup \bar{o} \mid \bar{o} \in \text{chains}(O)\}$

$\langle 4 \rangle 1. \exists \bar{o}' \in \text{chains}(O') : \sqcup \bar{o} \rightsquigarrow_r \sqcup \bar{o}'$ for arbitrary $\bar{o} \in \text{chains}(O)$

$\langle 5 \rangle 1. \text{Choose } \bar{o}' \in \text{chains}(O') \text{ such that } \forall j \in \mathbb{N} : \bar{o}[j] \rightsquigarrow_r \bar{o}'[j]$

$\langle 6 \rangle 1. \bar{o}[1] \in O$

P : Definition (10.21) of chains .

$\langle 6 \rangle 2. \text{Choose } \bar{o}'[1] \in O'$ such that $\bar{o}[1] \rightsquigarrow_r \bar{o}'[1]$

P : $\langle 6 \rangle 1$ and the assumption.

$\langle 6 \rangle 3. \forall j \in \mathbb{N} :$

choose $o \in O$ such that $\bar{o}[j+1] = \bar{o}[j] \succsim o$,

choose $o' \in O'$ such that $o \rightsquigarrow_r o'$,

and let $\bar{o}'[j+1] = \bar{o}'[j] \succsim o'$

P : Definition (10.21) of chains and the assumption.

$\langle 6 \rangle 4. \forall j > 1 : \bar{o}[j] \rightsquigarrow_r \bar{o}'[j]$

$\langle 7 \rangle 1. A : \bar{o}[j] \rightsquigarrow_r \bar{o}'[j]$ (induction hypothesis).

P : $\bar{o}[j+1] \rightsquigarrow_r \bar{o}'[j+1]$,

i.e. $\bar{o}[j] \succsim o \rightsquigarrow_r \bar{o}'[j] \succsim o'$ by $\langle 6 \rangle 3$.

$\langle 8 \rangle 1. \bar{o}[j] \rightsquigarrow_r \bar{o}'[j]$

P : The induction hypothesis.

$\langle 8 \rangle 2. o \rightsquigarrow_r o'$

P : $\langle 6 \rangle 3$.

$\langle 8 \rangle 3. Q.E.D.$

P : Lemma 30 (monotonicity of \rightsquigarrow_r with respect to \gtrsim) with $(p, n) = \bar{o}[j+1]$, $(p', n') = \bar{o}'[j+1]$, $(p_1, n_1) = \bar{o}[j]$, $(p_2, n_2) = o$, $(p'_1, n'_1) = \bar{o}'[j]$ and $(p'_2, n'_2) = o'$.

$\langle 7 \rangle 2.$ Q.E.D.

P : Standard induction on j , base case proved by $\langle 6 \rangle 2$.

$\langle 6 \rangle 5.$ $\bar{o}' \in \text{chains}(O')$

P : $\langle 6 \rangle 2$, $\langle 6 \rangle 3$ and definition (10.21) of *chains*.

$\langle 6 \rangle 6.$ Q.E.D.

P : $\langle 6 \rangle 2$, $\langle 6 \rangle 4$ and $\langle 6 \rangle 5$.

$\langle 5 \rangle 2.$ $\sqcup \bar{o} \rightsquigarrow_r \sqcup \bar{o}'$

$\langle 6 \rangle 1.$ Requirement 1: $\bigcup_{\bar{t} \in \text{negs}(\bar{o})} \sqcup \bar{t} \subseteq \bigcup_{\bar{t} \in \text{negs}(\bar{o}')} \sqcup \bar{t}$

$\langle 7 \rangle 1.$ $\forall \bar{t} \in \text{negs}(\bar{o}) : \bar{t} \in \text{negs}(\bar{o}')$

$\langle 8 \rangle 1.$ $\bar{t} \in \text{negs}(\bar{o}')$ for arbitrary $\bar{t} \in \text{negs}(\bar{o})$

$\langle 9 \rangle 1.$ Choose i such that $\forall j \in \mathbb{N} : \bar{t}[j] \in \pi_2(\bar{o}[j+i-1])$

P : Definition (10.23) of *negs*.

$\langle 9 \rangle 2.$ $\forall j \in \mathbb{N} : \bar{t}[j] \in \pi_2(\bar{o}'[j+i-1])$

P : $\forall j \in \mathbb{N} : \pi_2(\bar{o}[j]) \subseteq \pi_2(\bar{o}'[j])$ by $\langle 5 \rangle 1$ and definition (10.45) of \rightsquigarrow_r .

$\langle 9 \rangle 3.$ $\forall j \in \mathbb{N} : \exists t \in \mathcal{H} : \bar{t}[j+1] \in \{\bar{t}[j]\} \gtrsim \{t\}$

P : $\langle 8 \rangle 1$ and definition (10.23) of *negs*.

$\langle 9 \rangle 4.$ Q.E.D.

P : $\langle 9 \rangle 1$, $\langle 9 \rangle 2$, $\langle 9 \rangle 3$ and definition (10.23) of *negs*.

$\langle 8 \rangle 2.$ Q.E.D.

P : \forall -rule.

$\langle 7 \rangle 2.$ Q.E.D.

P : Definition of \cup and \subseteq .

$\langle 6 \rangle 2.$ Requirement 2: $\bigcup_{\bar{t} \in \text{pos}(\bar{o})} \sqcup \bar{t} \subseteq (\bigcup_{\bar{t} \in \text{pos}(\bar{o}')} \sqcup \bar{t}) \cup (\bigcup_{\bar{t} \in \text{negs}(\bar{o}')} \sqcup \bar{t})$

$\langle 7 \rangle 1.$ $\forall \bar{t} \in \text{pos}(\bar{o}) : \sqcup \bar{t} \subseteq (\bigcup_{\bar{t} \in \text{pos}(\bar{o}')} \sqcup \bar{t}) \cup (\bigcup_{\bar{t} \in \text{negs}(\bar{o}')} \sqcup \bar{t})$

$\langle 8 \rangle 1.$ $\sqcup \bar{t} \subseteq (\bigcup_{\bar{t} \in \text{pos}(\bar{o}')} \sqcup \bar{t}) \cup (\bigcup_{\bar{t} \in \text{negs}(\bar{o}')} \sqcup \bar{t})$ for arbitrary $\bar{t} \in \text{pos}(\bar{o})$

$\langle 9 \rangle 1.$ $\forall j \in \mathbb{N} : \bar{t}[j] \in \pi_1(\bar{o}[j])$

P : Definition (10.22) of *pos*.

$\langle 9 \rangle 2.$ $\forall j \in \mathbb{N} : \bar{t}[j] \in \pi_1(\bar{o}'[j]) \cup \pi_2(\bar{o}'[j])$

P : $\langle 5 \rangle 1$ and definition (10.45) of \rightsquigarrow_r .

$\langle 9 \rangle 3.$ $\forall j \in \mathbb{N} : \exists t \in \mathcal{H} : \bar{t}[j+1] \in \{\bar{t}[j]\} \gtrsim \{t\}$

P : $\langle 8 \rangle 1$ and definition (10.22) of *pos*.

$\langle 9 \rangle 4.$ C : A : $\forall j \in \mathbb{N} : \bar{t}[j] \in \pi_1(\bar{o}'[j])$

P : $\sqcup \bar{t} \subseteq \bigcup_{\bar{t} \in \text{pos}(\bar{o}')} \sqcup \bar{t}$

P : $\bar{t} \in \text{pos}(\bar{o}')$ by $\langle 9 \rangle 3$, the case assumption and definition (10.22) of *pos*.

$\langle 9 \rangle 5.$ C : A : $\exists i \in \mathbb{N} : (\bar{t}[i] \in \pi_2(\bar{o}'[i]) \wedge$

$\forall j < i : \bar{t}[j] \in \pi_1(\bar{o}'[j]))$

P : $\sqcup \bar{t} \subseteq \bigcup_{\bar{t} \in \text{negs}(\bar{o}')} \sqcup \bar{t}$

$\langle 10 \rangle 1.$ $\forall j > i : \bar{t}[j] \in \pi_2(\bar{o}'[j])$

- $\langle 11 \rangle 1.$ A : $\bar{t}[j] \in \pi_2(\bar{o}'[j])$ (induction hypothesis)
 P : $\bar{t}[j+1] \in \pi_2(\bar{o}'[j+1])$
- $\langle 12 \rangle 1.$ $\bar{t}[j] \in \pi_2(\bar{o}'[j])$
 P : The induction hypothesis.
- $\langle 12 \rangle 2.$ $\bar{t}[j+1] \in \pi_1(\bar{o}'[j+1]) \cup \pi_2(\bar{o}'[j+1])$
 P : $\langle 9 \rangle 2.$
- $\langle 12 \rangle 3.$ Choose $t \in \mathcal{H}$ such that $\bar{t}[j+1] \in \{\bar{t}[j]\} \succsim \{t\}$
 P : $\langle 9 \rangle 3.$
- $\langle 12 \rangle 4.$ Choose $o' \in O'$ such that $\bar{o}'[j+1] = \bar{o}'[j] \succsim o'$
 P : Definition (10.21) of chains.
- $\langle 12 \rangle 5.$ $t \in \pi_1(o') \cup \pi_2(o')$
 P : $\langle 12 \rangle 1, \langle 12 \rangle 2, \langle 12 \rangle 3$ and $\langle 12 \rangle 4.$
- $\langle 12 \rangle 6.$ $\pi_2(\bar{o}'[j]) \succsim (\pi_1(o') \cup \pi_2(o')) \subseteq \pi_2(\bar{o}'[j+1])$
 P : $\langle 12 \rangle 4$ and definition (10.13) of $\succsim.$
- $\langle 12 \rangle 7.$ $\bar{t}[j+1] \in \pi_2(\bar{o}'[j+1])$
 P : $\langle 12 \rangle 1, \langle 12 \rangle 3, \langle 12 \rangle 5$ and $\langle 12 \rangle 6.$
- $\langle 12 \rangle 8.$ Q.E.D.
- $\langle 11 \rangle 2.$ Q.E.D.
 P : Standard induction on j , base case proved by the assumption in $\langle 9 \rangle 5.$
- $\langle 10 \rangle 2.$ $\bar{t}[i \dots \infty) \in negs(\bar{o}')$ ⁴
- $\langle 11 \rangle 1.$ $\forall j \in \mathbb{N} : \bar{t}[i \dots \infty)[j] \in \pi_2(\bar{o}'[j+i-1])$
 P : $\forall j \in [i \dots \infty) : \bar{t}[j] \in \pi_2(\bar{o}'[j])$ by $\langle 9 \rangle 5$ and $\langle 10 \rangle 1.$
- $\langle 11 \rangle 2.$ $\forall j \in \mathbb{N} : \exists t \in \mathcal{H} : \bar{t}[i \dots \infty)[j+1] \in \{\bar{t}[i \dots \infty)[j]\} \succsim \{t\}$
 P : $\langle 9 \rangle 3.$
- $\langle 11 \rangle 3.$ Q.E.D.
 P : $\langle 11 \rangle 1, \langle 11 \rangle 2$, and definition (10.23) of $negs.$
- $\langle 10 \rangle 3.$ $\sqcup \bar{t} = \sqcup \bar{t}[i \dots \infty)$
- $\langle 11 \rangle 1.$ $\forall l \in \mathcal{L} : \sqcup_l \bar{t} = \sqcup_l \bar{t}[i \dots \infty)$
 P : $\sqcup_l \bar{t}$ is the least upper bound.
- $\langle 11 \rangle 2.$ Q.E.D.
 P : Definition (10.24) of $\sqcup \bar{t}.$
- $\langle 10 \rangle 4.$ $\sqcup \bar{t} \subseteq \bigcup_{\bar{t} \in negs(\bar{o}')} \sqcup \bar{t}$
 P : $\langle 10 \rangle 2$ and $\langle 10 \rangle 3.$
- $\langle 10 \rangle 5.$ Q.E.D.
- $\langle 9 \rangle 6.$ Q.E.D.
 P : By $\langle 9 \rangle 2$, the cases in $\langle 9 \rangle 4$ and $\langle 9 \rangle 5$ are exhaustive.
- $\langle 8 \rangle 2.$ Q.E.D.
 P : \forall -rule.
- $\langle 7 \rangle 2.$ Q.E.D.
 P : Definition of $\subseteq.$

⁴For a sequence q , $q[i \dots \infty)$ denotes the subsequence starting at index i , i.e. the sequence q with the first $i-1$ elements removed.

- $\langle 6 \rangle 3.$ Q.E.D.
 P : Definition (10.45) of \rightsquigarrow_r .
 $\langle 5 \rangle 3.$ Q.E.D.
 $\langle 4 \rangle 2.$ Q.E.D.
 $\langle 3 \rangle 2.$ Q.E.D.
 P : By definition (10.29) of μ_∞ .
 $\langle 2 \rangle 5.$ Q.E.D.
 P : The cases are exhaustive, as n is required to be a non-negative number or ∞ .
 $\langle 1 \rangle 2.$ Q.E.D.
 P : \forall -rule.

□

10.E.3 Monotonicity of \rightsquigarrow_g with Respect to the Sequence Diagram Operators

Theorem 10 *Monotonicity of \rightsquigarrow_g with respect to the neg operator*

- A : 1. $d = \text{neg } d_1$,
 i.e. $\llbracket d \rrbracket = \{(\{\langle \rangle\}, p \cup n) | (p, n) \in \llbracket d_1 \rrbracket\}$ by definition (10.34).
 2. $d_1 \rightsquigarrow_g d'_1$,
 i.e. $\llbracket d_1 \rrbracket \rightsquigarrow_g \llbracket d'_1 \rrbracket$ by definition (10.47),
 i.e. $\forall o_1 \in \llbracket d_1 \rrbracket : \exists o'_1 \in \llbracket d'_1 \rrbracket : o_1 \rightsquigarrow_r o'_1$ by definition (10.46).
 3. $d' = \text{neg } d'_1$,
 i.e. $\llbracket d' \rrbracket = \{(\{\langle \rangle\}, p \cup n) | (p, n) \in \llbracket d'_1 \rrbracket\}$ by definition (10.34).

- P : $d \rightsquigarrow_g d'$,
 i.e. $\llbracket d \rrbracket \rightsquigarrow_g \llbracket d' \rrbracket$ by definition (10.47),
 i.e. $\forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow_r o'$ by definition (10.46).

$\langle 1 \rangle 1.$ $\exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow_r o'$ for arbitrary $o = (p, n) \in \llbracket d \rrbracket$

$\langle 2 \rangle 1.$ $p = \{\langle \rangle\}$

P : Assumption 1.

$\langle 2 \rangle 2.$ Choose $(p_1, n_1) \in \llbracket d_1 \rrbracket$ such that $n = p_1 \cup n_1$

P : Assumption 1.

$\langle 2 \rangle 3.$ Choose $(p'_1, n'_1) \in \llbracket d'_1 \rrbracket$ such that $(p_1, n_1) \rightsquigarrow_r (p'_1, n'_1)$

P : Assumption 2.

$\langle 2 \rangle 4.$ $o' = (p', n') = (\{\langle \rangle\}, p'_1 \cup n'_1) \in \llbracket d' \rrbracket$

P : Assumption 3.

$\langle 2 \rangle 5.$ $(p, n) \rightsquigarrow_r (p', n')$

$\langle 3 \rangle 1.$ Requirement 1: $n \subseteq n'$, i.e. $p_1 \cup n_1 \subseteq p'_1 \cup n'_1$

P : $p_1 \subseteq p'_1 \cup n'_1$ and $n_1 \subseteq n'_1$ by $\langle 2 \rangle 3$ and definition (10.45) of \rightsquigarrow_r .

$\langle 3 \rangle 2.$ Requirement 2: $p \subseteq p' \cup n'$

P : Trivial, since $p = p' = \{\langle \rangle\}$ by $\langle 2 \rangle 1$ and $\langle 2 \rangle 4$.

$\langle 3 \rangle 3.$ Q.E.D.

P : Definition (10.45) of \rightsquigarrow_r .

$\langle 2 \rangle 6.$ Q.E.D.

$\langle 1 \rangle 2.$ Q.E.D.

P : \forall -rule.

□

Theorem 11 *Monotonicity of \rightsquigarrow_g with respect to the alt operator*

- A : 1. $d = d_1 \text{ alt } d_2,$
 i.e. $\llbracket d \rrbracket = \llbracket d_1 \rrbracket \uplus \llbracket d_2 \rrbracket$ by definition (10.37).
2. $d_1 \rightsquigarrow_g d'_1,$
 i.e. $\llbracket d_1 \rrbracket \rightsquigarrow_g \llbracket d'_1 \rrbracket$ by definition (10.47).
3. $d_2 \rightsquigarrow_g d'_2,$
 i.e. $\llbracket d_2 \rrbracket \rightsquigarrow_g \llbracket d'_2 \rrbracket$ by definition (10.47).
4. $d' = d'_1 \text{ alt } d'_2,$
 i.e. $\llbracket d' \rrbracket = \llbracket d'_1 \rrbracket \uplus \llbracket d'_2 \rrbracket$ by definition (10.37).

- P : $d \rightsquigarrow_g d',$
 i.e. $\llbracket d \rrbracket \rightsquigarrow_g \llbracket d' \rrbracket$ by definition (10.47).

P : Lemma 36 with $O = \llbracket d \rrbracket$, $O_1 = \llbracket d_1 \rrbracket$, $O_2 = \llbracket d_2 \rrbracket$, $O' = \llbracket d' \rrbracket$, $O'_1 = \llbracket d'_1 \rrbracket$ and $O'_2 = \llbracket d'_2 \rrbracket$.

□

Theorem 12 *Monotonicity of \rightsquigarrow_g with respect to the xalt operator*

- A : 1. $d = d_1 \text{ xalt } d_2,$
 i.e. $\llbracket d \rrbracket = \llbracket d_1 \rrbracket \cup \llbracket d_2 \rrbracket$ by definition (10.38).
2. $d_1 \rightsquigarrow_g d'_1,$
 i.e. $\llbracket d_1 \rrbracket \rightsquigarrow_g \llbracket d'_1 \rrbracket$ by definition (10.47),
 i.e. $\forall o_1 \in \llbracket d_1 \rrbracket : \exists o'_1 \in \llbracket d'_1 \rrbracket : o_1 \rightsquigarrow_r o'_1$ by definition (10.46).
3. $d_2 \rightsquigarrow_g d'_2,$
 i.e. $\llbracket d_2 \rrbracket \rightsquigarrow_g \llbracket d'_2 \rrbracket$ by definition (10.47),
 i.e. $\forall o_2 \in \llbracket d_2 \rrbracket : \exists o'_2 \in \llbracket d'_2 \rrbracket : o_2 \rightsquigarrow_r o'_2$ by definition (10.46).
4. $d' = d'_1 \text{ xalt } d'_2,$
 i.e. $\llbracket d' \rrbracket = \llbracket d'_1 \rrbracket \cup \llbracket d'_2 \rrbracket$ by definition (10.38).

- P : $d \rightsquigarrow_g d',$
 i.e. $\llbracket d \rrbracket \rightsquigarrow_g \llbracket d' \rrbracket$ by definition (10.47),
 i.e. $\forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow_r o'$ by definition (10.46).

$\langle 1 \rangle 1.$ $\exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow_r o'$ for arbitrary $o \in \llbracket d \rrbracket$

$\langle 2 \rangle 1.$ C : $o \in \llbracket d_1 \rrbracket$

$\langle 3 \rangle 1.$ Choose $o' \in \llbracket d'_1 \rrbracket$ such that $o \rightsquigarrow_r o'$

P : Assumption 2.

$\langle 3 \rangle 2.$ $o' \in \llbracket d' \rrbracket$

P : Assumption 4.

$\langle 3 \rangle 3.$ Q.E.D.

$\langle 2 \rangle 2.$ C : $o \in \llbracket d_2 \rrbracket$

$\langle 3 \rangle 1.$ Choose $o' \in \llbracket d'_2 \rrbracket$ such that $o \rightsquigarrow_r o'$

P : Assumption 3.
 ⟨3⟩2. $d' \in \llbracket d' \rrbracket$
 P : Assumption 4.
 ⟨3⟩3. Q.E.D.
 ⟨2⟩3. Q.E.D.
 P : By ⟨2⟩1, ⟨2⟩2, assumption 1 and definition of \cup .
 ⟨1⟩2. Q.E.D.
 P : \forall -rule.

□

Theorem 13 *Monotonicity of \rightsquigarrow_g with respect to the seq operator*

A : 1. $d = \text{seq } [d_1, \dots, d_n]$
 2. $\forall i \in [1 \dots n] : d_i \rightsquigarrow_g d'_i$,
 i.e. $\forall i \in [1 \dots n] : \llbracket d_i \rrbracket \rightsquigarrow_g \llbracket d'_i \rrbracket$ by definition (10.47).
 3. $d' = \text{seq } [d'_1, \dots, d'_n]$

P : $d \rightsquigarrow_g d'$,
 i.e. $\llbracket d \rrbracket \rightsquigarrow_g \llbracket d' \rrbracket$ by definition (10.47).

⟨1⟩1. C : $n = 1$
 ⟨2⟩1. $\llbracket d \rrbracket = \llbracket d_1 \rrbracket$
 P : Assumption 1, ⟨1⟩1 and definition (10.36) of seq.

⟨2⟩2. $\llbracket d' \rrbracket = \llbracket d'_1 \rrbracket$
 P : Assumption 3, ⟨1⟩1 and definition (10.36) of seq.

⟨2⟩3. $\llbracket d_1 \rrbracket \rightsquigarrow_g \llbracket d'_1 \rrbracket$
 P : Assumption 2.

⟨2⟩4. Q.E.D.
 P : ⟨2⟩1, ⟨2⟩2 and ⟨2⟩3.

⟨1⟩2. C : $n > 1$
 ⟨2⟩1. A : $\llbracket \text{seq } [d_1, \dots, d_{n-1}] \rrbracket \rightsquigarrow_g \llbracket \text{seq } [d'_1, \dots, d'_{n-1}] \rrbracket$
 (induction hypothesis)

P : $\llbracket \text{seq } [d_1, \dots, d_n] \rrbracket \rightsquigarrow_g \llbracket \text{seq } [d'_1, \dots, d'_n] \rrbracket$
 ⟨3⟩1. $\llbracket \text{seq } [d_1, \dots, d_n] \rrbracket = \llbracket \text{seq } [d_1, \dots, d_{n-1}] \rrbracket \gtrsim \llbracket d_n \rrbracket$
 P : Definition (10.36) of seq.

⟨3⟩2. $\llbracket \text{seq } [d'_1, \dots, d'_n] \rrbracket = \llbracket \text{seq } [d'_1, \dots, d'_{n-1}] \rrbracket \gtrsim \llbracket d'_n \rrbracket$
 P : Definition (10.36) of seq.

⟨3⟩3. $\llbracket \text{seq } [d_1, \dots, d_{n-1}] \rrbracket \rightsquigarrow_g \llbracket \text{seq } [d'_1, \dots, d'_{n-1}] \rrbracket$
 P : The induction hypothesis.

⟨3⟩4. $\llbracket d_n \rrbracket \rightsquigarrow_g \llbracket d'_n \rrbracket$
 P : Assumption 2.

⟨3⟩5. Q.E.D.

P : ⟨3⟩1, ⟨3⟩2, ⟨3⟩3, ⟨3⟩4 and lemma 33 with $O = \llbracket d \rrbracket$, $O_1 = \llbracket \text{seq } [d_1, \dots, d_{n-1}] \rrbracket$,
 $O_2 = \llbracket d_n \rrbracket$, $O' = \llbracket d' \rrbracket$, $O'_1 = \llbracket \text{seq } [d'_1, \dots, d'_{n-1}] \rrbracket$ and $O'_2 = \llbracket d'_n \rrbracket$.

⟨2⟩2. Q.E.D.

P : Assumptions 1 and 3 and standard induction on n , base case proved by ⟨1⟩1.

$\langle 1 \rangle 3.$ Q.E.D.

P : The cases are exhaustive.

□

Theorem 14 *Monotonicity of \rightsquigarrow_g with respect to the par operator*

- A : 1. $d = d_1 \text{par} d_2,$
 i.e. $\llbracket d \rrbracket = \llbracket d_1 \rrbracket \parallel \llbracket d_2 \rrbracket$ by definition (10.39).
2. $d_1 \rightsquigarrow_g d'_1,$
 i.e. $\llbracket d_1 \rrbracket \rightsquigarrow_g \llbracket d'_1 \rrbracket$ by definition (10.47).
3. $d_2 \rightsquigarrow_g d'_2,$
 i.e. $\llbracket d_2 \rrbracket \rightsquigarrow_g \llbracket d'_2 \rrbracket$ by definition (10.47).
4. $d' = d'_1 \text{par} d'_2,$
 i.e. $\llbracket d' \rrbracket = \llbracket d'_1 \rrbracket \parallel \llbracket d'_2 \rrbracket$ by definition (10.39).

P : $d \rightsquigarrow_g d',$
 i.e. $\llbracket d \rrbracket \rightsquigarrow_g \llbracket d' \rrbracket$ by definition (10.47).

P : Lemma 34 with $O = \llbracket d \rrbracket$, $O_1 = \llbracket d_1 \rrbracket$, $O_2 = \llbracket d_2 \rrbracket$, $O' = \llbracket d' \rrbracket$, $O'_1 = \llbracket d'_1 \rrbracket$ and $O'_2 = \llbracket d'_2 \rrbracket$.

□

Theorem 15 *Monotonicity of \rightsquigarrow_g with respect to the tc operator*

- A : 1. $d = d_1 \text{tc} C,$
 i.e. $\llbracket d \rrbracket = \llbracket d_1 \rrbracket \wr C$ by definition (10.40).
2. $d_1 \rightsquigarrow_g d'_1,$
 i.e. $\llbracket d_1 \rrbracket \rightsquigarrow_g \llbracket d'_1 \rrbracket$ by definition (10.47).
3. $d' = d'_1 \text{tc} C,$
 i.e. $\llbracket d' \rrbracket = \llbracket d'_1 \rrbracket \wr C$ by definition (10.40).

P : $d \rightsquigarrow_g d',$
 i.e. $\llbracket d \rrbracket \rightsquigarrow_g \llbracket d' \rrbracket$ by definition (10.47).

P : Lemma 35 with $O = \llbracket d \rrbracket$, $O_1 = \llbracket d_1 \rrbracket$, $O' = \llbracket d' \rrbracket$ and $O'_1 = \llbracket d'_1 \rrbracket$.

□

Theorem 16 *Monotonicity of \rightsquigarrow_g with respect to the loop operator*

- A : 1. $d = \text{loop } I d_1,$
 i.e. $\llbracket d \rrbracket = \biguplus_{i \in I} \mu_i \llbracket d_1 \rrbracket$ by definition (10.41).
2. $d_1 \rightsquigarrow_g d'_1,$
 i.e. $\llbracket d_1 \rrbracket \rightsquigarrow_g \llbracket d'_1 \rrbracket$ by definition (10.47).
3. $d' = \text{loop } I d'_1,$
 i.e. $\llbracket d' \rrbracket = \biguplus_{i \in I} \mu_i \llbracket d'_1 \rrbracket$ by definition (10.41).

P : $d \rightsquigarrow_g d',$
 i.e. $\llbracket d \rrbracket \rightsquigarrow_g \llbracket d' \rrbracket$ by definition (10.47).

$\langle 1 \rangle 1. \forall i \in I : \mu_i \llbracket d_1 \rrbracket \rightsquigarrow_g \mu_i \llbracket d'_1 \rrbracket$

P : Assumption 2 and lemma 38 (monotonicity of \rightsquigarrow_g with respect to μ_n) with $O = \llbracket d_1 \rrbracket$ and $O' = \llbracket d'_1 \rrbracket$.

$\langle 1 \rangle 2.$ Q.E.D.

P : $\langle 1 \rangle 1$, lemma 37 (monotonicity of \rightsquigarrow_g with respect to \sqcup) with $O_i = \mu_i \llbracket d_1 \rrbracket$ and $O'_i = \mu_i \llbracket d'_1 \rrbracket$.

□

Theorem 17 *Monotonicity of \rightsquigarrow_g , restricted to narrowing, with respect to the assert operator*

A : 1. $d = \text{assert } d_1$,

i.e. $\llbracket d \rrbracket = \{(p_1, n_1 \cup (\mathcal{H} \setminus p_1)) | (p_1, n_1) \in \llbracket d_1 \rrbracket\}$ by definition (10.35).

2. $d_1 \rightsquigarrow_{g,n} d'_1$,

i.e. $\llbracket d_1 \rrbracket \rightsquigarrow_{g,n} \llbracket d'_1 \rrbracket$ by definition (10.47),

i.e. $\forall o_1 \in \llbracket d_1 \rrbracket : \exists o'_1 \in \llbracket d'_1 \rrbracket : o_1 \rightsquigarrow_n o'_1$ by definition (10.46) restricted to narrowing.

3. $d' = \text{assert } d'_1$, i.e. $\llbracket d' \rrbracket = \{(p'_1, n'_1 \cup (\mathcal{H} \setminus p'_1)) | (p'_1, n'_1) \in \llbracket d'_1 \rrbracket\}$ by definition (10.35).

P : $d \rightsquigarrow_{g,n} d'$,

i.e. $\llbracket d \rrbracket \rightsquigarrow_{g,n} \llbracket d' \rrbracket$ by definition (10.47),

i.e. $\forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow_n o'$ by definition (10.46) restricted to narrowing.

$\langle 1 \rangle 1.$ $\exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow_n o'$ for arbitrary $o = (p, n) \in \llbracket d \rrbracket$

$\langle 2 \rangle 1.$ Choose $(p_1, n_1) \in \llbracket d_1 \rrbracket$ such that $p = p_1$ and $n = n_1 \cup (\mathcal{H} \setminus p_1)$

P : Assumption 1.

$\langle 2 \rangle 2.$ Choose $(p'_1, n'_1) \in \llbracket d'_1 \rrbracket$ such that $(p_1, n_1) \rightsquigarrow_n (p'_1, n'_1)$

P : Assumption 2.

$\langle 2 \rangle 3.$ $o' = (p', n') = (p'_1, n'_1 \cup (\mathcal{H} \setminus p'_1)) \in \llbracket d' \rrbracket$

P : Assumption 3.

$\langle 2 \rangle 4.$ $(p, n) \rightsquigarrow_n (p', n')$

$\langle 3 \rangle 1.$ Requirement 1: $p' \subset p$, i.e. $p'_1 \subset p_1$

P : $p'_1 \subset p_1$ by $\langle 2 \rangle 2$ and definition (10.49) of \rightsquigarrow_n .

$\langle 3 \rangle 2.$ Requirement 2: $n' = n \cup (p \setminus p')$,

i.e. $n'_1 \cup (\mathcal{H} \setminus p'_1) = (n_1 \cup (\mathcal{H} \setminus p_1)) \cup (p_1 \setminus p'_1)$

$\langle 4 \rangle 1.$ $n'_1 = n_1 \cup (p_1 \setminus p'_1)$

P : $\langle 2 \rangle 2$ and definition (10.49) of \rightsquigarrow_n .

$\langle 4 \rangle 2.$ $(\mathcal{H} \setminus p'_1) = (\mathcal{H} \setminus p_1) \cup (p_1 \setminus p'_1)$

$\langle 5 \rangle 1.$ $\mathcal{H} \setminus p_1 \subset \mathcal{H} \setminus p'_1$

P : $p'_1 \subset p_1$ by $\langle 2 \rangle 2$ and definition (10.49) of \rightsquigarrow_n .

$\langle 5 \rangle 2.$ $(\mathcal{H} \setminus p'_1) = (\mathcal{H} \setminus p_1) \cup ((\mathcal{H} \setminus p'_1) \setminus (\mathcal{H} \setminus p_1))$

P : $\langle 5 \rangle 1$ and the general lemma $A \subset B \implies B = A \cup (B \setminus A)$.

$\langle 5 \rangle 3.$ $(\mathcal{H} \setminus p'_1) \setminus (\mathcal{H} \setminus p_1) = p_1 \setminus p'_1$

P : $(\mathcal{H} \setminus a) \setminus (\mathcal{H} \setminus b) = b \setminus a$ by the general lemma illustrated in figure 10.E.3.

$\langle 5 \rangle 4.$ Q.E.D.

P : $\langle 5 \rangle 2$ and $(\mathcal{H} \setminus p_1) \cup ((\mathcal{H} \setminus p'_1) \setminus (\mathcal{H} \setminus p_1)) = (\mathcal{H} \setminus p_1) \cup (p_1 \setminus p'_1)$ by $\langle 5 \rangle 3$.

$\langle 4 \rangle 3.$ Q.E.D.

P : $\langle 4 \rangle 1, \langle 4 \rangle 2$ and associativity of \cup .

$\langle 3 \rangle 3.$ Q.E.D.

P : Definition (10.49) of \rightsquigarrow_n .

$\langle 2 \rangle 5.$ Q.E.D.

$\langle 1 \rangle 2.$ Q.E.D.

P : \forall -rule.

□

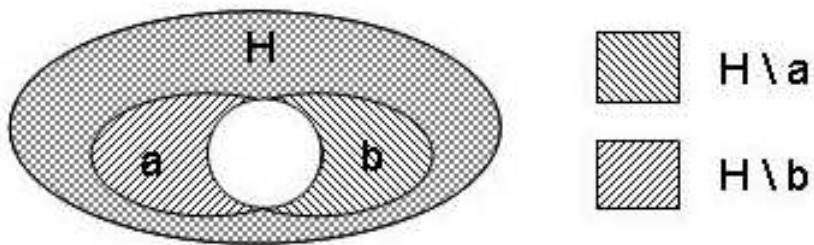


Figure 10.15: Lemma: $(H \setminus a) \setminus (H \setminus b) = b \setminus a$

Chapter 11

How to Transform UML neg into a Useful Construct

Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen

Publication.

Published in *Norsk Informatikkonferanse NIK'2005*, pages 55–66. Tapir, 2005.

Abstract.

In UML, the operator neg is used to specify negative, or unwanted, system behaviour. We agree that being able to specify negative behaviour is important. However, the UML neg is currently not well-suited for this purpose, the main problem being that a single operator is used with several different meanings depending on the context. In this paper we investigate some alternative definitions of neg. We also propose a solution in which neg is replaced by two new operators for specifying negative behaviour.

Note.

Minor error corrections have been made with respect to the original paper.

11.1 Introduction

In the interactions of UML 2.0 [OMG04], the unary operator neg is used to describe negative or invalid behaviour, i.e. scenarios that must not occur. By using structuring mechanisms like neg and the other interaction operators of UML 2.0, it is possible to “describe a number of traces in a compact and concise manner” [OMG04].

Interactions are typically used only to describe possible executions of the system. This means that nothing can be deduced about the validity of system behaviour *not* described by the interaction. In such a setting, it is particularly important to be able to specify negative as well as positive behaviours. Hence, we say that an interaction describes a set of positive behaviours, and a set of negative behaviours. Behaviours that are neither categorized as positive nor as negative are called inconclusive [HS03].

The problem with the UML neg operator is that people tend to interpret it differently depending on the context in which the operator appears. This makes it difficult to define a precise semantics for neg. The advantages of having a precise semantics include the following:

- it gives a clear answer in cases where intuition is weak.
- it facilitates formal reasoning.
- it forms a basis for tool-vendors and methodology builders.

11.2 Background

To set the stage for detailed discussion, in this section we give a brief introduction to UML 2.0 interactions and their semantics as defined in STAIRS [HHR05a, HHR05b].

11.2.1 Interactions and Trace Semantics

We define the semantics of interactions by sets of traces. A trace is a sequence of events, used to represent some system run. We distinguish between two kinds of events, the sending and reception of a message m , denoted $!m$ and $?m$, respectively.

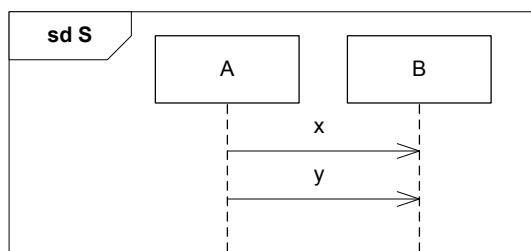


Figure 11.1: A simple interaction

A simple example of an interaction in the form of a sequence diagram is given in Figure 11.1. This diagram specifies that A sends the messages x and y to B . A and B are called lifelines, each

representing an object or a component in the system or its environment. The traces described by a diagram like this, are all possible sequences of events in the diagram such that both the send event is ordered before the corresponding receive event, and events on the same lifeline are ordered from the top and downwards. For the example in Figure 11.1, it follows that the first event to happen must be the sending of x . After that, either B may receive x or A may send y . The diagram thus specifies the two traces $\langle !x, ?x, !y, ?y \rangle$ and $\langle !x, ?y, ?x, ?y \rangle$.

Formally, we define the semantics of interaction diagrams by a function $\llbracket \quad \rrbracket$ that for any diagram d yields a pair (p, n) of trace-sets. The traces in p are called the positive traces of d , representing traces that may be the result of running the final system. The traces in n are called the negative traces of d , and must not appear in the final implementation. Traces that are neither defined as positive nor as negative are called *inconclusive*. We let \mathcal{H} denote the set of all traces where for each message, the send event is ordered before the corresponding receive event.

To illustrate the semantics of an interaction, we use a circle that is divided into three regions as shown in Figure 11.2. The circle as a whole corresponds to \mathcal{H} , the set of all possible traces. The topmost region represents the positive trace-set p , the lowest region represents the negative trace-set n , while the middle region contains the inconclusive traces.

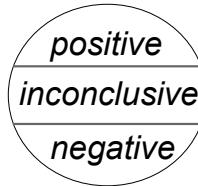


Figure 11.2: Illustrating the traces of an interaction

Definition 1 *The semantics of a diagram consisting of a single event e , is given by:*

$$\llbracket e \rrbracket \stackrel{\text{def}}{=} \{(\{\langle e \rangle\}, \emptyset)\}$$

Weak sequencing (seq) is the implicit composition mechanism construct used to create diagrams like the one in Figure 11.1, one of its possible textual representations being $\langle !x \text{ seq } ?x \rangle \text{ seq } \langle !y \text{ seq } ?y \rangle$.

Definition 2 *Assume $\llbracket d_1 \rrbracket = (p_1, n_1)$ and $\llbracket d_2 \rrbracket = (p_2, n_2)$. We then define the semantics of seq by:*

$$\llbracket d_1 \text{ seq } d_2 \rrbracket \stackrel{\text{def}}{=} (p_1 \succsim p_2, (n_1 \succsim p_2) \cup (n_1 \succsim n_2) \cup (p_1 \succsim n_2))$$

where weak sequencing of trace-sets, \succsim , is defined as:

$$s_1 \succsim s_2 \stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \forall l \in \mathcal{L} : h \upharpoonright l = h_1 \upharpoonright l \circ h_2 \upharpoonright l\}$$

where \mathcal{L} is the set of all lifelines, \circ is the concatenation operator on sequences, and $h \upharpoonright l$ is the trace h with all events not taking place on the lifeline l removed.

More complex interactions are constructed through the application of various operators. In this paper we only need one more operator, namely `alt` for specifying alternative behaviour. Definitions of other central operators may be found in e.g. [HHR05a].

Definition 3 Assume $\llbracket d_1 \rrbracket = (p_1, n_1)$ and $\llbracket d_2 \rrbracket = (p_2, n_2)$. We then define the semantics of `alt` by:

$$\llbracket d_1 \text{ alt } d_2 \rrbracket \stackrel{\text{def}}{=} (p_1 \cup p_2, n_1 \cup n_2)$$

11.2.2 Refinement

Refinement means to add information to a specification such that the specification becomes more complete. This may be achieved by categorizing inconclusive traces as either positive or negative, or by reducing the set of positive traces. Negative traces always remain negative.

Definition 4 Assume $\llbracket d \rrbracket = (p, n)$ and $\llbracket d' \rrbracket = (p', n')$. Refinement, \rightsquigarrow , is defined by:

$$d \rightsquigarrow d' \stackrel{\text{def}}{=} n \subseteq n' \wedge p \subseteq p' \cup n'$$

If $d \rightsquigarrow d'$, we say that d is refined by d' , or that d' is a refinement of d .

More details with motivation and examples using this definition may be found in [HHR05a] and [HHR05b]. In [HHR05c] we have proved that refinement as defined above is transitive, meaning that the result of several successive refinement steps will be a valid refinement of the original specification.

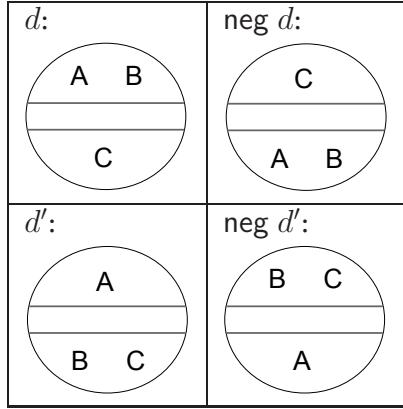
11.3 Alternative Definitions of neg

Obviously, the intuition behind `neg d` is that the positive traces described by d should be taken as negative. But as a definition, it is not complete, because it fails to answer questions such as:

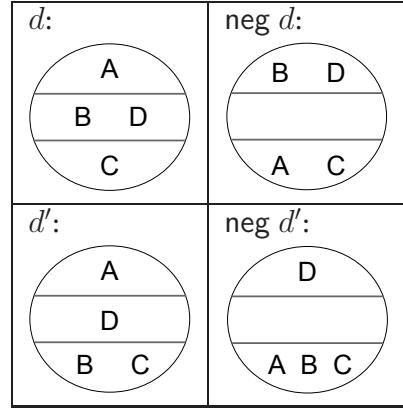
1. Does `neg neg d` mean the same as d ?
2. What happens to the negative traces of d ?
3. What should be the positive traces of `neg d`?

In this section we focus on the last question, by investigating the following possible answers:

- a. The negative traces of d .
- b. All traces except the negative traces of `neg d`.
- c. No trace at all.
- d. The empty trace.



(a) Using definition 5



(b) Using definition 6

Figure 11.3: Example specifications

All of these are feasible answers, and to some extent they are already used in practice. We will give formal definitions for each of the given alternatives, and discuss them from both a practical and a formal point of view. During our treatment of alternative a, we will also answer questions 1 and 2 above.

A basic premise in the following discussion is that we want compositionality (in formal theory often called monotonicity), meaning that by refining the different interaction operands separately, we obtain a refinement of the complete interaction. In the context of neg this means that for a (sub-)specification $\text{neg } d$, if d' is a refinement of the operand d , then $\text{neg } d'$ should be a refinement of $\text{neg } d$.

11.3.1 Alternative a: The Positive Traces of $\text{neg } d$ Are the Negative Traces of d

Intuitively, given the interpretation of negation in classical logic, it is natural to view $\text{neg } d$ as the opposite of d , taking the negative traces of d as the positive traces of $\text{neg } d$.

Definition 5 Assume $\llbracket d \rrbracket = (p, n)$. Then the semantics of neg may be defined by:

$$\llbracket \text{neg } d \rrbracket \stackrel{\text{def}}{=} (n, p)$$

With this definition we get $\text{neg neg } d = d$, as is the case of negation in classical logic. However, it is not obvious that logical negation is a good parallel to negation in the context of interactions. In fact, the following example gives a formal argument for why this is not a good interpretation of neg.

Consider the four specifications in Figure 11.3(a), where we only have the three traces A , B and C . The specification d' is a valid refinement of d , $d \rightsquigarrow d'$, where the originally positive trace B has been redefined as negative in d' . The figure also illustrates the negated specifications $\text{neg } d$ and $\text{neg } d'$. Since we have $d \rightsquigarrow d'$, by compositionality we should also have $\text{neg } d \rightsquigarrow \text{neg } d'$. However, this is not the case as the trace B has been moved from negative to positive, violating the refinement requirement $n \subseteq n'$.

To sum up, if we want compositionality with respect to neg (and we do), then definition 5 is not a good definition for neg. Also, we may conclude that the answer to question 1 above is that neg neg d should *not* be the same as d .

Negative Traces Remain Negative

As indicated by the above example, the negative traces in d cannot be defined as positive for neg d . By a similar argument, it may be demonstrated that they cannot be defined as inconclusive either. The only remaining alternative is to include the negative traces of d in the negative trace-set of neg d . Hence, the negative traces of neg d must be the positive and negative traces of d , combined.

The answer to question 2 is then that the negative traces of d remain negative for neg d . Hence, for the rest of this paper we only consider definitions where this is indeed the case. We now return to the discussion of the different alternatives for the positive traces of neg d .

11.3.2 Alternative b: The Positive Traces of neg d Are All Traces Except the Negative Traces of neg d

Another possibility is to let the positive traces of neg d be all traces except the negative traces of neg d (which, as we have argued, should consist of the positive and negative traces of d). The intuition here is that by using neg, the specification focuses on what is not allowed, and therefore everything else should be considered legal (i.e. positive).

Definition 6 Assume $\llbracket d \rrbracket = (p, n)$. Then the semantics of neg may be defined by:

$$\llbracket \text{neg } d \rrbracket \stackrel{\text{def}}{=} (\mathcal{H} \setminus (p \cup n), p \cup n)$$

Figure 11.3(b) gives an example using this definition, with \mathcal{H} (the universe of possible traces) being the set $\{A, B, C, D\}$. In the example we have both $d \rightsquigarrow d'$ (by categorizing the trace B as negative) and $\text{neg } d \rightsquigarrow \text{neg } d'$ (by redefining the trace B as negative) as desired. Compositionality is not simply a feature of this particular example, in Appendix 11.A we prove that it is in fact a general property of definition 6.

As the example demonstrates, all inconclusive traces for d are by definition made positive for neg d . This makes it impossible to write a specification defining some positive and negative traces, while wanting the rest to be inconclusive (as they will be considered positive by definition). Hence, using neg with this definition results in a less powerful specification language.

11.3.3 Alternative c: neg d Has No Positive Traces

The two previous sections have discussed the two alternatives where the positive traces of neg d should be the negative or inconclusive traces of d , concluding that none of these is an ideal solution. A third alternative, then, is to say that neg d has no positive traces at all. The operator neg should be used to specify negative behaviour, and nothing else.

Definition 7 Assume $\llbracket d \rrbracket = (p, n)$. Then the semantics of neg may be defined by:

$$\llbracket \text{neg } d \rrbracket \stackrel{\text{def}}{=} (\emptyset, p \cup n)$$

To understand the effect of this definition, it is useful to look at example interactions using neg in combination with other operators such as seq and alt. For the examples, we will use a simple vending machine selling tea and coffee.

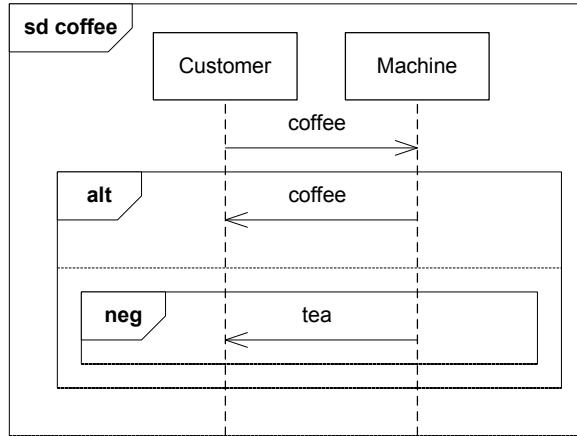


Figure 11.4: Ordering coffee

The interaction in Figure 11.4 specifies two traces, both starting with the customer ordering coffee. The first alt-operand specifies that the machine giving coffee to the customer is a positive trace. This operand defines no negative traces. The second alt-operand specifies no positive traces, but states that giving tea is negative. Using the definition of seq to combine the first message of ordering coffee with the alt-fragment, we get that ordering and getting coffee is positive, while ordering coffee and getting tea is negative.

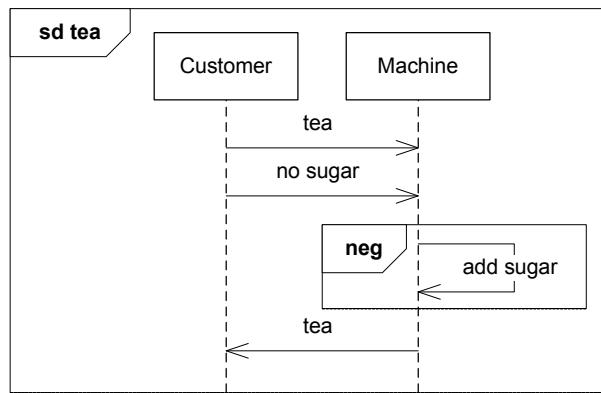


Figure 11.5: Ordering tea

The interaction in Figure 11.5 is meant to specify that if the customer orders tea with no sugar, then the machine should *not* add sugar before giving the tea to the customer. However, with the given definitions of neg and seq, the actual meaning of this interaction is not what intuition believes it to be. The neg-fragment in itself defines adding sugar as negative. Together with the remaining traces of the interaction, we then get that the trace of ordering tea, ordering

no sugar, the machine adding sugar and then giving tea is a negative trace. This is in accordance with the intuition.

But, using the definitions of seq and neg, we get that the interaction in Figure 11.5 has *no* positive traces! This results from fact that the neg-fragment has no positive traces, and that a set of traces sequenced with the empty set gives the empty set. This is not in accordance with the intuition that the interaction should have the same positive traces as the same interaction only with the neg-fragment simply removed.

A possible solution to this problem could be to change the definition of weak sequencing of trace-sets to include special cases for the empty set.

Definition 8 *Weak sequencing of trace-sets may instead be defined as:*

$$s_1 \succsim s_2 \stackrel{\text{def}}{=} \begin{cases} s_2 & \text{if } s_1 = \emptyset \\ s_1 & \text{if } s_2 = \emptyset \\ \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \forall l \in \mathcal{L} : h \upharpoonright l = h_1 \upharpoonright l \frown h_2 \upharpoonright l\} & \text{otherwise} \end{cases}$$

However, it turns out that this definition does not give compositionality for seq. With seq being the basic composition operator, it is essential that it is compositional in order to let the different parts of a basic interaction be refined separately. Hence, we have kept definition 2, for which we have proved compositionality in [HHRSS05c].

11.3.4 Alternative d: The Only Positive Trace for neg d Is the Empty Trace

Another alternative, motivated by the above discussion on Figure 11.5, is to let the empty trace be positive for neg d.

Definition 9 *Assume $\llbracket d \rrbracket = (p, n)$. Then the semantics of neg may be defined by:*

$$\llbracket \text{neg } d \rrbracket \stackrel{\text{def}}{=} (\{\langle \rangle\}, p \cup n)$$

As opposed to the empty trace-set \emptyset having no behaviours at all, the empty trace $\langle \rangle$ represents the possible behaviour of doing nothing. In the context of Figure 11.5, this is a highly relevant distinction.

Using definition 9, we get the same negative trace(s) as we did with definition 7. The difference is that Figure 11.5 with definition 9 also specifies the intended result that ordering tea, ordering no sugar and then receiving tea (without sugar) is positive. This is because a trace sequenced with the empty trace equals the original trace, meaning that we may simply omit the negative fragments from an interaction when calculating its positive traces.

From this example, it is tempting to believe that definition 9 is a better definition of neg than definition 7. However, returning to the example in Figure 11.4, we will demonstrate that this is not the case. With our new definition of neg, the second alt-operand in Figure 11.4 gives that the empty trace $\langle \rangle$ is positive. Combining this with the first message in the diagram, we get that ordering coffee but not getting anything is a positive trace. This result is however not intended when using neg in this context.

Hence, neither definition 7 nor definition 9 are perfect definitions for neg. In the next section we propose a solution that is based on multiple operators for negation.

11.3.5 Suggested Solution

As we have demonstrated, it is not possible to give one single context-free definition of neg capturing all its intuitive meanings. We do not regard introducing context-sensitive definitions to be a good solution, as they tend to be very complicated and difficult to use. Instead, we strongly believe that most of the problems come from having only one operator for specifying negative behaviour. Hence, our solution is to define two different operators for negation, capturing the most important uses of the neg operator.

First we introduce skip, the empty diagram corresponding to a program doing nothing.

Definition 10 *The semantics of skip, the empty diagram, is defined as*

$$\llbracket \text{skip} \rrbracket \stackrel{\text{def}}{=} (\{\langle\rangle\}, \emptyset)$$

Of the alternative definitions given for neg, we believe that definition 7 is the most fundamental one. We therefore define a new basic operator refuse using this definition.

Definition 11 *Assume $\llbracket d \rrbracket = (p, n)$. We then define the semantics of refuse by:*

$$\llbracket \text{refuse } d \rrbracket \stackrel{\text{def}}{=} (\emptyset, p \cup n)$$

The operator refuse may be used to define other operators for specifying negative behaviour, and is also meant to be used when specifying that one of the operands of an alt-fragment represents negative behaviour (as in Figure 11.4).

We may now define another operator veto as a high-level operator.

Definition 12 *Assume $\llbracket d \rrbracket = (p, n)$. We then define the semantics of veto by:*

$$\text{veto } d \stackrel{\text{def}}{=} \text{skip alt}(\text{refuse } d)$$

This definition is equal to the alternative definition 9 above, and is meant to be used when specifying additional messages that make otherwise positive behaviours negative (as in Figure 11.5).

If desirable, it is also possible to define another operator corresponding to definition 6, in order to specify that everything apart from the given traces should be positive. Additional negation operators may also be defined. However, this should be done with great care, because even though more operators may result in a more expressive language, it might lead to reduced readability.

A major challenge is to find unambiguous and intuitive names for the different negation operators proposed here. We do not claim to have succeeded in this task. Our main contribution is that it is necessary to have more than one operator for specifying negative behaviour, and that some of these might be high-level operators derived from a well-chosen set of basic operators.

11.4 Related Work

Störrle [Stö03] discusses three alternative definitions for neg. His first alternative, referred to as “not the traces of d ”, defines that $\text{neg } d$ has no positive traces, and that the negative traces of $\text{neg } d$ are the positive traces of d . Since the negative traces of d are lost with this definition,

this alternative is rejected. His second alternative, “anything but d ”, is similar to our alternative definition 6, except that the negative traces of d are considered positive for $\text{neg } d$. The third alternative he proposes, “flip valid and invalid”, equals our definition 5. Of the three alternatives, Störrle chooses this third interpretation, calling it “the only consistent approach”, as it gives $\text{neg neg } d = d$. But, as we have argued in this paper, logical negation is not the most obvious interpretation of negation in the context of interactions.

In [CK04], Cengarle and Knapp define the semantics of UML 2.0 interactions by the notions of positive and negative satisfaction. Based on a similar argument as the one we presented in favour of definition 9, they regard the empty trace as positive for $\text{neg } d$. However, the negative traces of d are inconclusive for $\text{neg } d$ (as in the first of Störrle’s definitions). For alternatives, specified by alt , they define that a trace is negative only if it is negative in both operands. This means that with their semantics, the interaction in Figure 11.4 has *no* negative traces even though it uses the operator neg .

Cengarle and Knapp pose an interesting question related to the use of negation in specifications: Should a trace be negative if a prefix of it is specified as negative? The answer in [CK04] is principally yes, proposing an even stronger approach where a trace is taken as negative as soon as it has completed a negative sub-diagram. An advantage of this is that it allows for earlier identification (or even prevention) of negative traces.

As an example of this approach, consider again the specification in Figure 11.4 (interpreting neg as refuse). The specification states that ordering co ee and then receiving tea is a negative trace. But what about the trace which then continues with the customer receiving co ee as well — should this be positive or negative? In STAIRS we regard such a trace as inconclusive, arguing that if a trace is not described in the diagram, then the specifier has either not thought about the situation or not wanted to classify it as either positive or negative.

The alternative, as proposed in [CK04], would be to say that the trace of ordering co ee , first receiving tea and then receiving co ee is negative. In general, once a negative (sub-)scenario has occurred, the total trace will be negative independently of what happens next. Formally, this may be achieved by the following definitions:

Definition 13 Assume $\llbracket d_1 \rrbracket = (p_1, n_1)$ and $\llbracket d_2 \rrbracket = (p_2, n_2)$. Then the semantics of refuse may be defined by:

$$\llbracket \text{refuse } d_1 \rrbracket \stackrel{\text{def}}{=} (\emptyset, (p_1 \cup n_1) \succsim \mathcal{H})$$

The semantics of seq may be defined by:

$$\llbracket d_1 \text{ seq } d_2 \rrbracket \stackrel{\text{def}}{=} (p_1 \succsim p_2, n_1 \cup (p_1 \succsim n_2))$$

In appendix 11.A, we prove compositionality with respect to these alternative definitions.

These definitions are somewhat different from those in [CK04]. Translated to our formalism, they append \mathcal{H} only to the negative traces of the first operand of seq , and not to the negative traces of the second operand or to the negative traces of neg . We believe that our definition corresponds more closely to the intuition behind this alternative approach. In contrast to the definitions in [CK04], definition 13 also ensures that skip is an identity element for seq , i.e. $\llbracket d \text{ seq } \text{skip} \rrbracket = \llbracket d \rrbracket$.

Even without considering the possible formal definitions, there is at least one large disadvantage of this approach. If a trace is negative as soon as it has traversed a negative region, it follows

that for a trace to be positive, all of its prefixes would have to be positive or at least inconclusive. In our vending machine example, this means that it would not be possible to make a consistent specification where ordering and getting coffee is positive, while ordering coffee and not getting anything (i.e. nothing more happens) is negative.

11.5 Conclusions

We have discussed alternative definitions of neg, considering both formal and practical issues. None of the proposed alternatives are able to capture all the different uses of the neg operator. As a result, we find it necessary to have more than one operator for specifying negative behaviour. In this paper we have proposed to replace neg with two new operators, for which the given definitions ensures compositionality.

Acknowledgements. The research on which this paper reports has partly been carried out within the context of the IKT-2010 project SARDAS (15295/431) funded by the Research Council of Norway. We thank the other SARDAS members for fruitful discussions, and in particular Mass Soldal Lund for coming up with the first vending machine example. We thank Iselin Engan for helpful comments on the final draft of this paper.

References

- [CK04] María Victoria Cengarle and Alexander Knapp. UML 2.0 interactions: Semantics and refinement. In *Proc. 3rd Int. Wsh. Critical Systems Development with UML (CSDUML'04)*, Technical report TUM-I0415, pages 85–99. Institut für Informatik, Technische Universität München, 2004.
- [HHR05a] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. STAIRS towards formal design with sequence diagrams. *Journal of Software and Systems Modeling*, Online First:1–13, 2005.
- [HHR05b] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. Why timed sequence diagrams require three-event semantics. In *Scenarios: Models, Transformations and Tools*, volume 3466 of *LNCS*, pages 1–25. Springer, 2005.
- [HHR05c] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309, Department of Informatics, University of Oslo, 2005.
- [HS03] Øystein Haugen and Ketil Stølen. STAIRS — Steps to analyze interactions with refinement semantics. In *Proc. International Conference on UML (UML'2003)*, volume 2863 of *LNCS*, pages 388–402. Springer, 2003.
- [OMG04] Object Management Group. *UML 2.0 Superstructure Specification*, document: ptc/04-10-02 edition, 2004.
- [Stö03] Harald Störrle. Assert, negate and refinement in UML-2 interactions. In *Proc. Wsh. Critical Systems Development with UML (CSDUML'03)*, Technical report TUM-I0317, pages 79–93. Institut für Informatik, Technische Universität München, 2003.

11.A Proofs

For all proofs in this section, we use $\llbracket d \rrbracket = (p, n)$, $\llbracket d' \rrbracket = (p', n')$, $\llbracket d_1 \rrbracket = (p_1, n_1)$, $\llbracket d'_1 \rrbracket = (p'_1, n'_1)$, $\llbracket d_2 \rrbracket = (p_2, n_2)$ and $\llbracket d'_2 \rrbracket = (p'_2, n'_2)$. We also use $(s_1 \cup s_2) \succsim s_3 = (s_1 \succsim s_3) \cup (s_2 \succsim s_3)$, as proved in [HHR05c]. Also, recall that a definition of neg is compositional if for any two specifications d and d' such that $d \rightsquigarrow d'$, we also have $\text{neg } d \rightsquigarrow \text{neg } d'$. Similarly for compositionality of other operators such as seq and refuse.

Theorem 1 *Definition 6 of neg is compositional with respect to refinement as defined in definition 4.*

Proof: Using definition 6, we get $\llbracket \text{neg } d \rrbracket = (\mathcal{H} \setminus (p \cup n), p \cup n)$ and $\llbracket \text{neg } d' \rrbracket = (\mathcal{H} \setminus (p' \cup n'), p' \cup n')$. By definition 4, we need to prove $p \cup n \subseteq p' \cup n'$ and $\mathcal{H} \setminus (p \cup n) \subseteq (\mathcal{H} \setminus (p' \cup n')) \cup (p' \cup n')$. The last condition is trivial, as the right side equals \mathcal{H} . As d is refined by d' (by assumption), we have $n \subseteq n'$ and $p \subseteq p' \cup n'$, giving $p \cup n \subseteq p' \cup n'$ as required.

Theorem 2 *Definition 7 of neg is compositional with respect to refinement as defined in definition 4.*

Proof: Using definition 7, we get $\llbracket \text{neg } d \rrbracket = (\emptyset, p \cup n)$ and $\llbracket \text{neg } d' \rrbracket = (\emptyset, p' \cup n')$. By definition 4, we need to prove $p \cup n \subseteq p' \cup n'$ and $\emptyset \subseteq \emptyset \cup (p \cup n)$. The last condition is trivial. As d is refined by d' , we have $n \subseteq n'$ and $p \subseteq p' \cup n'$, giving $p \cup n \subseteq p' \cup n'$ as required.

Theorem 3 *Definition 9 of neg is compositional with respect to refinement as defined in definition 4.*

Proof: Similar to the proof of Theorem 2, replacing every occurrence of \emptyset with $\{\langle \rangle\}$.

Theorem 4 *Definition 13 of refuse and seq is compositional with respect to refinement as defined in definition 4.*

Proof for refuse: Using definition 13, we get $\llbracket \text{refuse } d \rrbracket = (\emptyset, (p \cup n) \succsim \mathcal{H})$ and $\llbracket \text{refuse } d' \rrbracket = (\emptyset, (p' \cup n') \succsim \mathcal{H})$. By definition 4, we need to prove $(p \cup n) \succsim \mathcal{H} \subseteq (p' \cup n') \succsim \mathcal{H}$ and $\emptyset \subseteq \emptyset \cup ((p' \cup n') \succsim \mathcal{H})$. The last condition is trivial. As d is refined by d' , we have $n \subseteq n'$ and $p \subseteq p' \cup n'$, which together with definition 2 of \succsim gives $n \succsim \mathcal{H} \subseteq n' \succsim \mathcal{H}$ and $p \succsim \mathcal{H} \subseteq (p' \cup n') \succsim \mathcal{H}$, i.e. $(p \succsim \mathcal{H}) \cup (n \succsim \mathcal{H}) \subseteq (p' \succsim \mathcal{H}) \cup (n' \succsim \mathcal{H})$ as required.

Proof for seq: Using definition 13, we get $\llbracket d_1 \text{ seq } d_2 \rrbracket = (p_1 \succsim p_2, n_1 \cup (p_1 \succsim n_2))$ and $\llbracket d'_1 \text{ seq } d'_2 \rrbracket = (p'_1 \succsim p'_2, n'_1 \cup (p'_1 \succsim n'_2))$. By definition 4, we need to prove $n_1 \cup (p_1 \succsim n_2) \subseteq n'_1 \cup (p'_1 \succsim n'_2)$ and $p_1 \succsim p_2 \subseteq (p'_1 \succsim p'_2) \cup (n'_1 \cup (p'_1 \succsim n'_2))$. As d_1 is refined by d'_1 and d_2 is refined by d'_2 , we have $n_1 \subseteq n'_1$, $p_1 \subseteq p'_1 \cup n'_1$ and $n_2 \subseteq n'_2$, which together with definition 2 of \succsim gives $n_1 \cup (p_1 \succsim n_2) \subseteq n'_1 \cup ((p'_1 \cup n'_1) \succsim n'_2) = n'_1 \cup (p'_1 \succsim n'_2) \cup (n'_1 \succsim n'_2)$. By lemma 1, this is a subset of $n'_1 \cup (p'_1 \succsim n'_2)$ which was to be proved. Similarly, we get $p_1 \succsim p_2 \subseteq (p'_1 \cup n'_1) \succsim (p'_2 \cup n'_2) = (p'_1 \succsim p'_2) \cup (p'_1 \succsim n'_2) \cup (n'_1 \succsim (p'_2 \cup n'_2))$. By lemma 1, this is a subset of $(p'_1 \succsim p'_2) \cup (p'_1 \succsim n'_2) \cup n'_1$ which was to be proved.

Lemma 1 *Using definition 13 of refuse and seq, we get that for an arbitrary interaction d with negative trace-set n , i.e. $\llbracket d \rrbracket = (p, n)$, we have $n \succsim s \subseteq n$ for any trace-set $s \subseteq \mathcal{H}$.*

Proof: By induction over the structure of d .

Base case: d is a single event e , i.e. $n = \emptyset$ by definition 1. As $\emptyset \succsim s = \emptyset$ by definition 2, we get $\emptyset \succsim s \subseteq \emptyset$ as required.

Case 1: $d = \text{refuse } d_1$, i.e. $n = (p_1 \cup n_1) \succsim \mathcal{H}$ by definition 13. As $\mathcal{H} \succsim s \subseteq \mathcal{H}$ for arbitrary s , we get $(p_1 \cup n_1) \succsim \mathcal{H} \succsim s \subseteq (p_1 \cup n_1) \succsim \mathcal{H}$ as required.

Case 2: $d = d_1 \text{ alt } d_2$, i.e. $n = n_1 \cup n_2$ by definition 3. By induction hypothesis, we have $n_1 \succsim s \subseteq n_1$ and $n_2 \succsim s \subseteq n_2$, giving $(n_1 \cup n_2) \succsim s = (n_1 \succsim s) \cup (n_2 \succsim s) \subseteq n_1 \cup n_2$ as required.

Case 3: $d = d_1 \text{ seq } d_2$, i.e. $n = n_1 \cup (p_1 \succsim n_2)$ by definition 13. By induction hypothesis, we have $n_1 \succsim s \subseteq n_1$ and $n_2 \succsim s \subseteq n_2$, giving $((n_1 \cup (p_1 \succsim n_2)) \succsim s = (n_1 \succsim s) \cup (p_1 \succsim n_2 \succsim s) \subseteq n_1 \cup (p_1 \succsim n_2)$ as required.

Chapter 12

Refining UML Interactions with Underspecification and Nondeterminism

Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen

Publication.

Published as Technical Report 325, Department of Informatics, University of Oslo, 2007. Extended and revised version of: Refining UML interactions with underspecification and nondeterminism. In *Nordic Journal of Computing*, 12(2):157–188, 2005.

Abstract.

STAIRS is an approach to the compositional development of UML interactions, such as sequence diagrams and interaction overview diagrams. An important aspect of STAIRS is the ability to distinguish between underspecification and inherent nondeterminism through the use of potential and mandatory alternatives. This paper investigates this distinction in more detail. Refinement notions explain when (and how) both kinds of nondeterminism may be reduced during the development process. In particular, in this paper we extend STAIRS with guards, which may be used to specify the choice between alternatives. Finally, we introduce the notion of an implementation and define what it means for an implementation to be correct with respect to a specification.

ACM CCS Categories and Subject Descriptors.

D.2.1 [Software Engineering]: Requirements/Specifications; D.3.1 [Programming Languages]: Formal Definitions and Theory; F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs

Keywords.

UML interactions, formal semantics, nondeterminism, underspecification, refinement

12.1 Introduction

STAIRS [HS03, HHRS05b] is an approach to the compositional development of UML interactions, such as sequence diagrams and interaction overview diagrams. Interactions in UML 2.0 [OMG04] are behavioural definitions that describe some, but not necessarily all, of the behaviour that a given system performs. Most often the interactions will describe positive behaviours, i.e. behaviours that the system is allowed to perform. There may also be behaviours that the interactions define as negative, meaning that they are unacceptable, and there may even be behaviours of the system that are not at all covered by any of the interactions defined.

This partiality of interactions is motivated by several factors. First of all, the description of a real system requires far too many interaction diagrams to define all the behaviours. To manage such a volume of diagrams would be impractical. Also, the goal of interactions is to visualize important interaction patterns. Thus the emphasis is on importance, rather than completeness. This is in contrast to most other kinds of behavioural specifications, including UML state machines. The definition of a state machine is complete in the sense that it may be seen to define all the possible behaviours of that entity.

A methodology may initially use interactions to capture user requirements, and use these as stepping stones for the next development stages where emphasis is placed more on completeness and realizability. STAIRS supports this through the notion of refinement. In particular, the refinement definitions take into account that initial specifications in the form of interactions typically describe only a few example scenarios. A scenario not described by an initial specification is not necessarily unwanted, but it has not been thought of yet. Thus, a refinement step may be to include new scenarios in the specification, as well as to reduce the amount of underspecification and nondeterminism in the specification. Refinement may also be to describe some of the aspects of a scenario in more detail.

In this paper we focus on defining and refining specifications with nondeterminism. In the introductory chapter of the UNITY-book [CM88] Chandy and Misra observe:

Nondeterminism is useful in two ways. First, it is employed to derive simple programs, where simplicity is achieved by avoiding unnecessary determinism; such programs can be optimized by limiting the nondeterminism, i.e., by disallowing executions unsuitable for a given architecture. Second, some systems (e.g., operating systems and delay-insensitive circuits) are inherently nondeterministic; programs that represent such systems have to employ some nondeterministic constructs.

STAIRS is based on this overall observation. However, contrary to Chandy and Misra we take the position that the two useful ways of using nondeterminism should be described differently.

Avoiding unnecessary determinism may for instance be achieved through underspecification. By underspecification we mean that the specification gives several alternative behaviours that are equivalent in the sense that they all serve the same purpose. For an implementation to be correct, it is sufficient to fulfil only one of the alternative behaviours. Underspecification may also be used as an abstraction mechanism, for instance by giving several alternative behaviours but not stating how to select between them. This will typically later be refined into an if-then-else construct in the implementation.

On the other hand, inherent nondeterminism is used to capture alternative behaviours that must all be possible for the implementation. A typical example is the tossing of a coin, where

both heads and tails should be possible outcomes, and no legal refinement should remove one of these two alternatives. A system may also need to exhibit nondeterministic behaviour due to differences in its environment.

Inherent nondeterminism is very different from underspecification, and should be described differently. One important reason for this is that unless we do not distinguish these two, there will be no way to ensure that inherently nondeterministic behaviour is implemented as such. This may not seem like a major problem at first. If the development team knows that a given specification should be implemented as delay-insensitive circuits you will probably get the inherently nondeterministic implementation that you expect. However, in the domain of information security, the inherently nondeterministic behaviour is fundamental for the validity of the specification. As pointed out in e.g. [Jac89] and [JL00], security properties are in general not preserved by standard refinement. If nondeterminism is used as a means to hide the internal workings of a system, it is essential that it is not treated as underspecification, which allows elimination of all uncertainty (nondeterminism) in a refinement.

In [Ros95] Roscoe points out that using inherent nondeterminism ensures security as it prevents the making of any inference about the possible outcomes, while for nondeterminism based on underspecification there are three possible conclusions about the security of a system: secure, insecure, or don't know. Hence, it makes things a lot easier if the specification language provides a way to distinguish between these two ways of using nondeterminism.

In the setting of UML interactions, the operator `alt` is used to specify alternative behaviours. As the UML standard [OMG04] is rather vague on whether these alternatives represent underspecification or inherent nondeterminism, people interpret the same interaction differently, leading to confusion. This could be avoided by having two different operators for specifying alternative behaviours, as we have in STAIRS. This is particularly important as the partiality of interactions makes it important to know which of the described scenarios represent significantly different behaviours and which scenarios only serve as examples of how to achieve the same purpose.

The remainder of this paper is structured into six sections. Section 12.2 introduces the basic STAIRS formalism, while Section 12.3 uses this in an example specification illustrating nondeterminism. In Section 12.4 we extend the formalism with guards, and in Section 12.5 we discuss refinement in STAIRS with emphasis on nondeterminism. Section 12.6 defines what it means for a system to be a correct implementation of a STAIRS specification. Section 12.7 provides a brief summary and relates STAIRS to approaches known from the literature.

12.2 Background: UML Interactions with Denotational Trace Semantics

In this section, we present the basic STAIRS formalism. Section 12.2.1 gives the fundamental trace mechanisms. In Section 12.2.2 we present our textual syntax for interactions, while Section 12.2.3 formally defines denotational trace semantics for UML interactions.

12.2.1 Representing Executions by Traces

In STAIRS, we define the semantics of interactions by using sequences of events. By A^ω we denote the set of all finite and infinite sequences over the set A . We use $\langle \rangle$ to denote the empty sequence. Moreover, by $\langle e_1, e_2, \dots, e_m \rangle$ we denote the sequence of m elements, whose first element is e_1 , whose second element is e_2 , and so on. We define the functions

$$\#_a \in A^\omega \rightarrow \mathbb{N}_0 \cup \{\infty\}, \quad a[n] \in A^\omega \times \mathbb{N} \rightarrow A$$

to yield the length and the n th element of a sequence. Hence, $\#a$ yields the number of elements in a and $a[n]$ yields a 's n th element if $n \leq \#a$.

We also need functions for concatenation, truncation and filtering:

$$a \cdot b \in A^\omega \times A^\omega \rightarrow A^\omega, \quad a|n \in A^\omega \times \mathbb{N}_0 \rightarrow A^\omega, \quad a \circ B \in \mathbb{P}(A) \times A^\omega \rightarrow A^\omega$$

Concatenating two sequences implies gluing them together. Hence, $a_1 \cdot a_2$ denotes a sequence of length $\#a_1 + \#a_2$ that equals a_1 if a_1 is infinite, and is prefixed by a_1 and suffixed by a_2 , otherwise. For any $0 \leq i \leq \#a$, we define $a|_i$ to denote the prefix of a of length i .

The filtering function \circ is used to filter away elements. By $B \circ a$ we denote the sequence obtained from the sequence a by removing all elements in a that are not in the set of elements B . For example, we have that

$$\{1, 3\} \circ \langle 1, 1, 2, 1, 3, 2 \rangle = \langle 1, 1, 1, 3 \rangle$$

A trace h is a sequence of events, used to represent a system run. For any single message, transmission must happen before reception if both events are present. Thus we get the following well-formedness requirement on traces, stating that if at any point in the trace we have a transmission event, up to that point we must have had at least as many transmissions as receptions of that particular message:

$$\forall i \in [1, \#h] : k.h[i] = ! \Rightarrow \\ \#((\{!\} \times \{m.h[i]\}) \circ h|_i) > \#((\{?\} \times \{m.h[i]\}) \circ h|_i)$$

\mathcal{H} denotes the set of all well-formed traces.

12.2.2 Syntax of Interactions

The set of syntactically correct interactions, denoted by \mathcal{D} , is defined by the BNF-grammar in Fig. 12.1. Signal represents the actual content of a message, Lifeline is the name of a lifeline (representing a component) in the diagram and Set should be an expression that evaluates to a subset of \mathbb{N}_0 (the natural numbers including 0).

As can be seen from the definition, a message is a triple (s, tr, re) of a signal s , a transmitter tr , and a receiver re . As a shorthand, we will often use the name of the signal to stand for the whole message in cases where the transmitter and receiver are clear from the context. We let \mathcal{L} denote the set of all lifelines, and \mathcal{M} denote the set of all messages. We distinguish between two kinds of events; a transmission event tagged by an exclamation mark “!” represents the transmission of a message, while a reception event tagged by a question mark “?” represents the reception of a message. \mathcal{E} denotes the set of all events, while \mathcal{K} denotes $\{!, ?\}$.

$\langle \text{Interaction} \rangle$	$\rightarrow \langle \text{Empty} \rangle \mid \langle \text{Event} \rangle \mid \langle \text{Weak sequencing} \rangle \mid \langle \text{Refuse} \rangle \mid \langle \text{Assert} \rangle \mid \langle \text{Potential alternatives} \rangle \mid \langle \text{Mandatory alternatives} \rangle \mid \langle \text{Loop} \rangle$
$\langle \text{Empty} \rangle$	$\rightarrow \text{skip}$
$\langle \text{Event} \rangle$	$\rightarrow \langle \text{Kind} \rangle \langle \text{Message} \rangle$
$\langle \text{Kind} \rangle$	$\rightarrow \langle \text{Transmission} \rangle \mid \langle \text{Reception} \rangle$
$\langle \text{Transmission} \rangle$	$\rightarrow !$
$\langle \text{Reception} \rangle$	$\rightarrow ?$
$\langle \text{Message} \rangle$	$\rightarrow (\text{Signal}, \langle \text{Transmitter} \rangle, \langle \text{Receiver} \rangle)$
$\langle \text{Transmitter} \rangle$	$\rightarrow \text{Lifeline}$
$\langle \text{Receiver} \rangle$	$\rightarrow \text{Lifeline}$
$\langle \text{Refuse} \rangle$	$\rightarrow \text{refuse} [\langle \text{Interaction} \rangle]$
$\langle \text{Assert} \rangle$	$\rightarrow \text{assert} [\langle \text{Interaction} \rangle]$
$\langle \text{Potential alternatives} \rangle$	$\rightarrow \text{alt} [\langle \text{Interaction list} \rangle]$
$\langle \text{Mandatory alternatives} \rangle$	$\rightarrow \text{xalt} [\langle \text{Interaction list} \rangle]$
$\langle \text{Loop} \rangle$	$\rightarrow \text{loop Set} [\langle \text{Interaction} \rangle]$
$\langle \text{Weak sequencing} \rangle$	$\rightarrow \text{seq} [\langle \text{Interaction list} \rangle]$
$\langle \text{Interaction list} \rangle$	$\rightarrow \langle \text{Interaction} \rangle \mid \langle \text{Interaction list} \rangle, \langle \text{Interaction} \rangle$

Figure 12.1: Syntax of interactions

We define the functions

$$k._ \in \mathcal{E} \rightarrow \mathcal{K}, \quad m._ \in \mathcal{E} \rightarrow \mathcal{M}, \quad tr._, re._ \in \mathcal{E} \rightarrow \mathcal{L}$$

to yield the kind, message, transmitter and receiver of an event, respectively.

We also define the functions

$$ll._ \in \mathcal{D} \rightarrow \mathbb{P}(\mathcal{L}), \quad ev._ \in \mathcal{D} \rightarrow \mathbb{P}(\mathcal{E}), \quad msg._ \in \mathcal{D} \rightarrow \mathbb{P}(\mathcal{M})$$

to yield the set of lifelines, events and messages of an interaction, respectively.

Interactions are built from events through the application of various operators as defined by the grammar in Fig. 12.1. We do not cover the complete set of operators in UML 2.0 [OMG04], but rather focus on a few essential operators. These fundamental operators may be used to define other useful, high-level operators as demonstrated in Section 12.5.2. See [HHR06] for STAIRS definitions of additional operators like parallel execution and gates.

The operators assert, alt, seq and loop are UML 2.0 operators. The operator xalt is new, proposed in [HS03] to model mandatory alternatives, i.e. alternatives that must all be present in the final implementation. For negation, UML 2.0 uses the operator neg. However, this operator is used in several contexts, with slightly different meanings as we explain in [RHS05]. Therefore, we have in this paper chosen to introduce a new operator refuse that covers one of these traditional uses of neg.

We only consider interactions that are well-formed in the sense that if both the transmitter and the receiver lifelines of a message are present in the diagram, then both the transmission and the reception event of that message must be present as well. Formally:

$$\forall m \in msg.d : (\#ev.d > 1 \wedge tr.m \in ll.d \wedge re.m \in ll.d) \Rightarrow \\ \#\{ e \in ev.d \mid k.e = ! \wedge m.e = m \} = \#\{ e \in ev.d \mid k.e = ? \wedge m.e = m \}$$

where $\{\}$ denotes a multi-set and $\#$ is overloaded to yield the number of elements in such a set. A multi-set is needed here as the same message (consisting of a signal, a transmitter and a receiver) may occur more than once in the same diagram.

Also, we assume that for all operators except from seq, the operand(s) consist only of complete messages, i.e. messages with both the transmission and the reception event within the operand.

12.2.3 Semantics of Interactions

The semantics of interactions is defined by a function $\llbracket \cdot \rrbracket$ that for any interaction d yields a set $\llbracket d \rrbracket$ of interaction obligations. The term obligation is used to explicitly convey that any implementation of a specification is obliged to fulfil each specified alternative. (What it formally means to fulfil an obligation is discussed in Section 12.6.) An interaction obligation is a pair (p, n) of sets of traces. The first set p represents positive traces that may be the result of running the final system, while the second set n represents negative traces that must not appear in the implementation of the obligation. Traces not defined as positive or negative are called *inconclusive*. As will be formally defined in Section 12.5, a refinement may later redefine (some of) these inconclusive traces as positive or negative. An obligation pair (p, n) is contradictory if $p \cap n \neq \emptyset$.

The empty diagram, denoted by skip, is a specification without any events that corresponds to a program doing nothing. The empty diagram defines the empty trace as positive:

$$\llbracket \text{skip} \rrbracket \stackrel{\text{def}}{=} \{(\{\langle \rangle\}, \emptyset)\} \quad (12.1)$$

For an interaction consisting of a single event e , its semantics is given by:

$$\llbracket e \rrbracket \stackrel{\text{def}}{=} \{(\{\langle e \rangle\}, \emptyset)\} \quad (12.2)$$

The actual content of the messages is not significant for the purpose of this paper. Hence, we do not give any semantic interpretation of messages as such.

The rest of this section will define the semantics of the different composition operators described briefly in Section 12.2.2. Table 12.1 lists the notational conventions that will be used in the following definitions.

Weak Sequencing

Weak sequencing is the implicit composition mechanism combining constructs of an interaction. The operator seq is defined by the following invariants:

- The ordering of events within each of the operands is maintained in the result.

Symbol	Stands for
d	interaction
D	list of interactions, separated by comma
h	trace
s, p, n	trace set
o	interaction obligation
O	set of interaction obligations

Table 12.1: Notational conventions

- Events on different lifelines from different operands may come in any order.
- Events on the same lifeline from different operands are ordered such that an event of the first operand comes before that of the second operand, and so on.

First, we define weak sequencing of trace sets:

$$s_1 \succsim s_2 \stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc h_1 \cap e.l \circledcirc h_2\} \quad (12.3)$$

where $e.l$ denotes the set of events that may take place on the lifeline l . Formally:

$$e.l \stackrel{\text{def}}{=} \{e \in \mathcal{E} \mid (k.e = ! \wedge tr.e = l) \vee (k.e = ? \wedge re.e = l)\} \quad (12.4)$$

Weak sequencing of interaction obligations is defined as:

$$(p_1, n_1) \succsim (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \succsim p_2, (n_1 \succsim p_2) \cup (n_1 \succsim n_2) \cup (p_1 \succsim n_2)) \quad (12.5)$$

Notice that all traces obtained by combining a negative and a positive trace-set, will also be negative. Weak sequencing of sets of interaction obligations is defined as:

$$O_1 \succsim O_2 \stackrel{\text{def}}{=} \{o_1 \succsim o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\} \quad (12.6)$$

Finally, the seq construct is defined by:

$$\begin{aligned} \llbracket \text{seq } [d] \rrbracket &\stackrel{\text{def}}{=} \llbracket d \rrbracket \\ \llbracket \text{seq } [D, d] \rrbracket &\stackrel{\text{def}}{=} \llbracket \text{seq } [D] \rrbracket \succsim \llbracket d \rrbracket \end{aligned} \quad (12.7)$$

As an example, the interaction in Fig. 12.2 shows two messages both originating from L1 and targeting L2. Its semantics is calculated as:

$$\begin{aligned} \llbracket W \rrbracket &= \llbracket \text{seq } [!x, ?x, !y, ?y] \rrbracket \\ &= ((\llbracket !x \rrbracket \succsim \llbracket ?x \rrbracket) \succsim \llbracket !y \rrbracket) \succsim \llbracket ?y \rrbracket && (\text{Def. (12.7)}) \\ &= ((\{(\{(!x)\}, \emptyset)\} \succsim \{(\{?x\}, \emptyset)\}) \succsim \{(\{(!y)\}, \emptyset)\}) \\ &\quad \succsim \{(\{?y\}, \emptyset)\} && (\text{Def. (12.2)}) \\ &= (\{(\{(!x, ?x)\}, \emptyset)\} \succsim \{(\{(!y)\}, \emptyset)\}) \succsim \{(\{?y\}, \emptyset)\} && (\text{Defs. (12.3)} - (12.6)) \\ &= \{(\{(!x, ?x, !y), (!x, !y, ?x)\}, \emptyset)\} \succsim \{(\{?y\}, \emptyset)\} && (\text{Defs. (12.3)} - (12.6)) \\ &= \{(\{(!x, ?x, !y, ?y), (!x, !y, ?x, ?y)\}, \emptyset)\} && (\text{Defs. (12.3)} - (12.6)) \end{aligned}$$

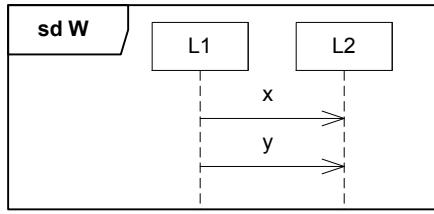


Figure 12.2: Weak sequencing

Hence, this interaction specifies one interaction obligation with two positive traces and no negative ones. The positive traces state that the transmission of x must be the first event to happen, but after that either y may be transmitted (by L1) or x may be received (by L2).

Negative Behaviour

The refuse construct defines negative traces:

$$\llbracket \text{refuse } [d] \rrbracket \stackrel{\text{def}}{=} \{(\emptyset, p \cup n) \mid (p, n) \in \llbracket d \rrbracket\} \quad (12.8)$$

Notice that a negative trace cannot be made positive by reapplying refuse. Negative traces remain negative, since negation should be seen as an operation that characterizes traces absolutely and not relatively.

Assertion

The assert construct makes all inconclusive traces negative. Except for that the sets of positive and negative traces are left unchanged:

$$\llbracket \text{assert } [d] \rrbracket \stackrel{\text{def}}{=} \{(p, n \cup (\mathcal{H} \setminus p)) \mid (p, n) \in \llbracket d \rrbracket\} \quad (12.9)$$

Notice that contradictory obligation pairs remain contradictory.

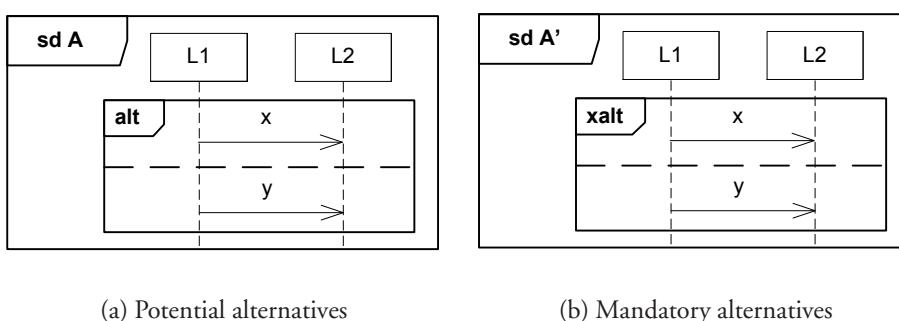


Figure 12.3: Specifying alternatives

Potential Alternatives

The `alt` construct is used when specifying underspecification, i.e. to define potential traces that are equivalent in the sense that it is sufficient for an implementation to include only one of them. The semantics of `alt` is the inner union of each point-wise selection of interaction obligations from its operands:

$$\llbracket \text{alt } [d_1, \dots, d_m] \rrbracket \stackrel{\text{def}}{=} \{ \biguplus \{o_1, \dots, o_m\} \mid \forall i \in [1, m] : o_i \in \llbracket d_i \rrbracket \} \quad (12.10)$$

The inner union of interaction obligations is defined as:

$$\biguplus_{i \in [1, m]} (p_i, n_i) \stackrel{\text{def}}{=} (\bigcup_{i \in [1, m]} p_i, \bigcup_{i \in [1, m]} n_i) \quad (12.11)$$

Fig. 12.3(a) gives a simple example using the `alt` construct. The dashed horizontal line separates the operands. We get:

$$\begin{aligned} \llbracket A \rrbracket &= \llbracket \text{alt } [\text{seq } \langle !x, ?x \rangle, \text{seq } \langle !y, ?y \rangle] \rrbracket \\ &= \{ \biguplus \{ (\{\langle !x, ?x \rangle\}, \emptyset), (\{\langle !y, ?y \rangle\}, \emptyset) \} \} \quad (\text{Defs. (12.3) -- (12.7), (12.10)}) \\ &= \{ (\{\langle !x, ?x \rangle, \langle !y, ?y \rangle\}, \emptyset) \} \quad (\text{Def. (12.11)}) \end{aligned}$$

Mandatory Alternatives

The `xalt` construct is used to specify inherent nondeterminism, i.e. mandatory alternatives that must all be present in an implementation:

$$\llbracket \text{xalt } [d_1, \dots, d_m] \rrbracket \stackrel{\text{def}}{=} \bigcup_{i \in [1, m]} \llbracket d_i \rrbracket \quad (12.12)$$

Fig. 12.3(b) has the same messages as Fig. 12.3(a), but separated by `xalt` instead of `alt`. In this case, we get two interaction obligations:

$$\begin{aligned} \llbracket A' \rrbracket &= \llbracket \text{xalt } [\text{seq } \langle !x, ?x \rangle, \text{seq } \langle !y, ?y \rangle] \rrbracket \\ &= \bigcup \{ \llbracket \text{seq } \langle !x, ?x \rangle \rrbracket, \llbracket \text{seq } \langle !y, ?y \rangle \rrbracket \} \quad (\text{Def. (12.12)}) \\ &= \bigcup \{ \{(\{\langle !x, ?x \rangle\}, \emptyset)\}, \{(\{\langle !y, ?y \rangle\}, \emptyset)\} \} \quad (\text{Defs. (12.3) -- (12.7)}) \\ &= \{ (\{\langle !x, ?x \rangle\}, \emptyset), (\{\langle !y, ?y \rangle\}, \emptyset) \} \quad (\text{Def. of } \bigcup) \end{aligned}$$

Loop

For a set of interaction obligations we define a finite loop construct $\mu_n O$, where $n \in \mathbb{N}_0$ denotes the number of times the body of the loop is iterated. $\mu_n O$ is defined inductively as follows:

$$\mu_n O \stackrel{\text{def}}{=} \begin{cases} \{(\{\langle \rangle \}, \emptyset)\} & \text{if } n = 0 \\ O & \text{if } n = 1 \\ \mu_{n-1} O \gtrsim O & \text{otherwise} \end{cases} \quad (12.13)$$

For a definition of infinite loop, see [HHR06].

In the UML 2.0 standard [OMG04], loop is used together with limits stating the minimum and maximum number of times the content of the loop should be executed. In our definition, the set I is a generalization of this, such that the numbers in I specify the possible alternatives for how many times the loop content should be executed. Not all of these need to be actual alternatives in an implementation, meaning that the definition of loop uses the point-wise inner union between these alternatives, similar to the definition of alt:

$$\llbracket \text{loop } I [d] \rrbracket \stackrel{\text{def}}{=} \left\{ \biguplus_{i \in I} o_i \mid \forall i \in I : o_i \in \mu_i \llbracket d \rrbracket \right\} \quad (12.14)$$

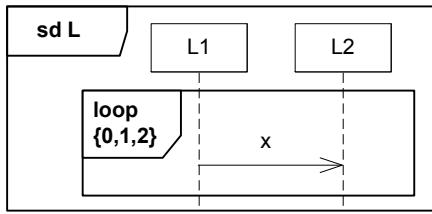


Figure 12.4: Looping

As an example, the interaction in Fig. 12.4 has the following semantics:

$$\begin{aligned}
\llbracket L \rrbracket &= \llbracket \text{loop } \{0, 1, 2\} [\text{seq } (!x, ?x)] \rrbracket \\
&= \left\{ \biguplus_{i \in \{0, 1, 2\}} o_i \mid \forall i \in \{0, 1, 2\} : \right. \\
&\quad \left. o_i \in \mu_i \llbracket \text{seq } (!x, ?x) \rrbracket \right\} \quad (\text{Def. (12.14)}) \\
&= \left\{ \biguplus_{i \in \{0, 1, 2\}} o_i \mid \forall i \in \{0, 1, 2\} : \right. \\
&\quad \left. o_i \in \mu_i \{(\{\langle !x, ?x \rangle\}, \emptyset)\} \right\} \quad (\text{Defs. (12.3) - (12.7)}) \\
&= \left\{ \biguplus_{i \in \{0, 1, 2\}} o_i \mid o_0 \in \mu_0 \{(\{\langle !x, ?x \rangle\}, \emptyset)\} \wedge \right. \\
&\quad o_1 \in \mu_1 \{(\{\langle !x, ?x \rangle\}, \emptyset)\} \wedge \\
&\quad o_2 \in \mu_2 \{(\{\langle !x, ?x \rangle\}, \emptyset)\} \} \\
\\
&= \left\{ \biguplus_{i \in \{0, 1, 2\}} o_i \mid o_0 \in \{(\{\langle \rangle\}, \emptyset)\} \wedge \right. \\
&\quad o_1 \in \{(\{\langle !x, ?x \rangle\}, \emptyset)\} \wedge \\
&\quad o_2 \in \{(\{\langle !x, ?x \rangle\}, \emptyset)\} \\
&\quad \quad \sim \{(\{\langle !x, ?x \rangle\}, \emptyset)\} \} \quad (\text{Def. (12.13)}) \\
&= \left\{ \biguplus_{i \in \{0, 1, 2\}} o_i \mid o_0 \in \{(\{\langle \rangle\}, \emptyset)\} \wedge \right. \\
&\quad o_1 \in \{(\{\langle !x, ?x \rangle\}, \emptyset)\} \wedge \\
&\quad o_2 \in \{(\{\langle !x, ?x, !x, ?x \rangle, \langle !x, !x, ?x, ?x \rangle\}, \emptyset)\} \} \quad (\text{Defs. (12.3) - (12.6)}) \\
&= \left\{ \biguplus \{ (\{\langle \rangle\}, \emptyset), \right. \\
&\quad (\{\langle !x, ?x \rangle\}, \emptyset), \\
&\quad (\{\langle !x, ?x, !x, ?x \rangle, \langle !x, !x, ?x, ?x \rangle\}, \emptyset) \} \} \\
&= \{ (\{\langle \rangle, \langle !x, ?x \rangle, \langle !x, ?x, !x, ?x \rangle, \langle !x, !x, ?x, ?x \rangle\}, \emptyset) \} \quad (\text{Def. (12.11)})
\end{aligned}$$

12.3 STAIRS and Nondeterminism

As seen in the previous section, weak sequencing may result in several different traces with the same events in a somewhat different order. These traces are alternative means to achieve the same goal, and they are therefore grouped into the same interaction obligation as it is sufficient to keep only one of them in an implementation.

In UML 2.0, the other means to specify alternative behaviours is by using the operator alt. This is used both for specifying potential alternatives where keeping only one is sufficient, and for mandatory alternatives that must all be present in a correct implementation. In STAIRS, we have distinguished these two uses by separating between our two operators alt and xalt. Each use of UML 2.0 alt corresponds in STAIRS to either alt or xalt. In this section we present an example illustrating the use of these two operators.

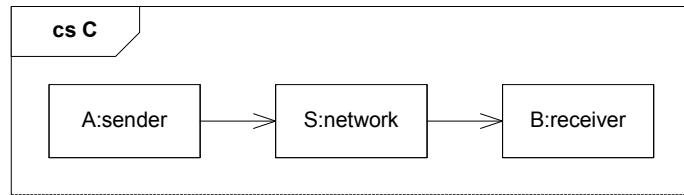


Figure 12.5: Composite structure of context C

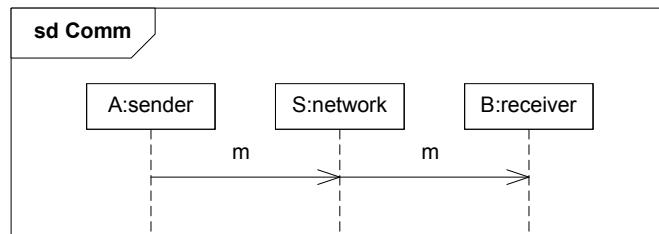


Figure 12.6: Very simple communication

Consider a situation where a sender communicates with a receiver through a network of type S as shown in the UML composite structure diagram in Fig. 12.5 (notice that this is not an interaction). A very simple communication is shown by the interaction in Fig. 12.6, its semantics being:

$$\{ (\{ \langle ! (m, A, S), ? (m, A, S), ! (m, S, B), ? (m, S, B) \rangle \}, \emptyset) \}$$

Next, we would like to specify that there is a need for redundant communication through the network S. That is, the network S needs to support more than one way of bringing the message m from one end of the network to the other. There may be several reasons for requiring this redundancy:

- Several paths through the network will make it easier to exploit the full capacity of the network.

- Multiple paths will ensure increased internal robustness of the network and as such improve the availability of the full communication.
- Multiple paths will make it more difficult to attack the network to jeopardize the communication, and as such the communication security is improved.

We indicate in Fig. 12.7 a simple network architecture for S where there are alternative branches. A real communication network may of course have far more paths, but giving a few is sufficient for the purpose of this paper. We want to make an interaction where we require two (different) communication possibilities, and we may do this by introducing an `xalt` construct as shown in Fig. 12.8, where S is expanded according to the structure in Fig. 12.7.

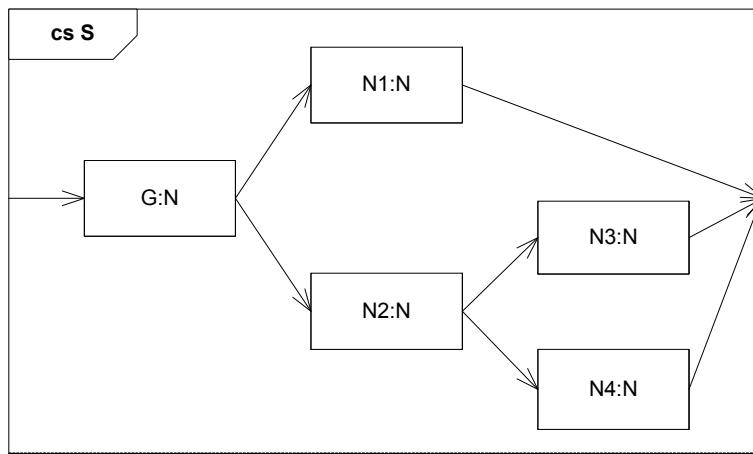


Figure 12.7: Internal structure of the network S showing three communication paths

We have used `xalt` here in order to express that the network must support at least two communication paths. Of course, for each concrete communication only one of them will be applied. After node N2, the network S has yet another branch giving two alternative paths. For the sake of the discussion we assume that it is not important to have both of these available, and so we specify the alternatives using `alt` and not `xalt` in Fig. 12.8.

The semantics of S_Comm is:

```

{ ( { ⟨!(m, A, G), ?(m, A, G), !(m, G, N1), ?(m, G, N1),
      !(m, N1, B), ?(m, N1, B)⟩ }, ∅ ),
  ( { ⟨!(m, A, G), ?(m, A, G), !(m, G, N2), ?(m, G, N2),
      !(m, N2, N3), ?(m, N2, N3), !(m, N3, B), ?(m, N3, B)⟩,
      ⟨!(m, A, G), ?(m, A, G), !(m, G, N2), ?(m, G, N2),
      !(m, N2, N4), ?(m, N2, N4), !(m, N4, B), ?(m, N4, B)⟩ }, ∅ ) }
  
```

Fig. 12.9 illustrates this semantics using a specialized Venn-diagram with one ellipse for each interaction obligation. Traces not shown as positive or negative in an obligation are inconclusive for this obligation.

Formally, S_Comm is a refinement of Comm. Refinement will be formally defined in Section 12.5. In Section 12.5 we will also develop this example further, by giving some possible

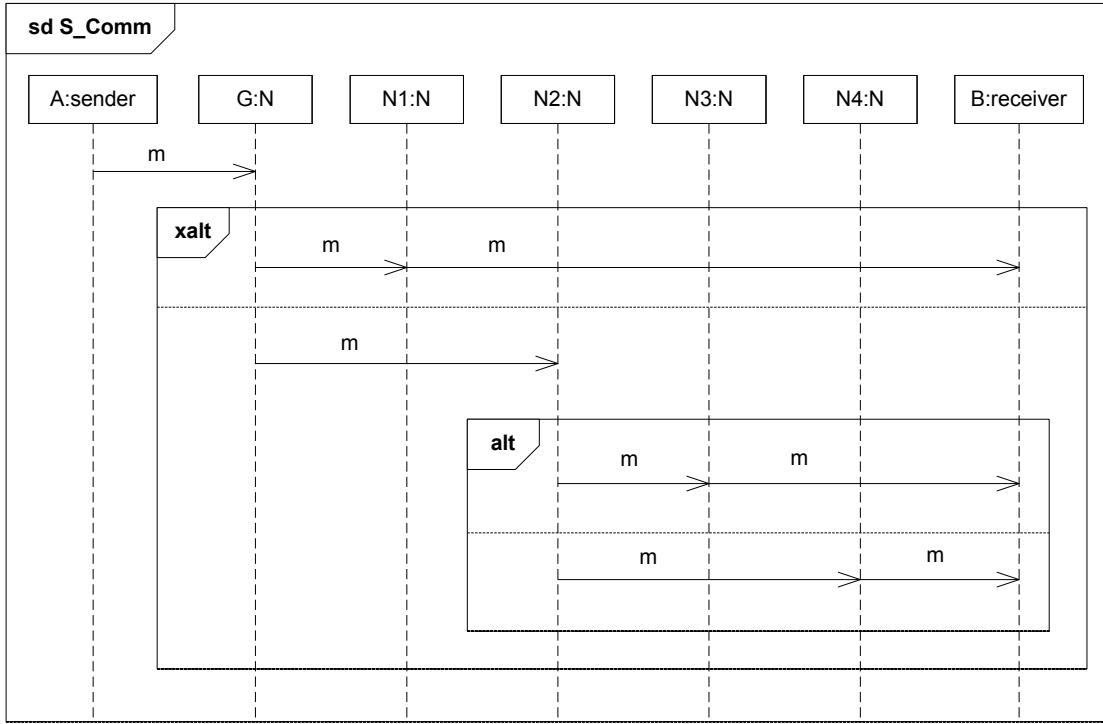


Figure 12.8: Communication behaviour requiring two communication paths

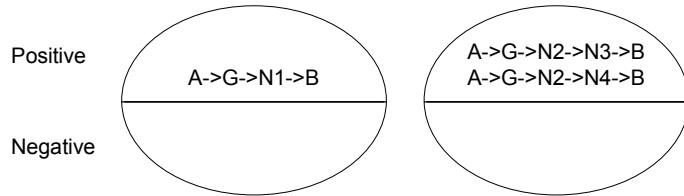


Figure 12.9: Venn-diagram of the specification in Fig. 12.8

refinements to illustrate the similarities and differences between the two operators *alt* and *xalt*. But first, in the next section, we formally extend STAIRS with guards, which may be used to specify the choice between alternatives.

12.4 Extending STAIRS with Data and Guards

Although the focus of interactions is on the messages, the diagrams may also be decorated with data. The most common use of data in interactions is in guards, which is a mechanism for choosing between alternatives. Data is also used in assignments and general constraints. In this section we extend our basic formalism with definitions of these concepts. The extension ensures that (sub-)interactions not including data have the same semantics as before.

12.4.1 Data

Since interactions mainly specify events and not data, the exact data values will most of the time be underspecified (or unspecified). Changes in the data may in general happen at any time, also when there is nothing in the diagram indicating such a change. As a consequence, in the semantic model we do not include data as such. Instead, data is represented indirectly through events representing its use in assignments, constraints, and guards.

Formally, we extend the syntax of interactions as defined by the BNF-grammar in Fig. 12.10. Nonterminals that are unchanged from the original syntax in Fig. 12.1 are not repeated. Variable should be either a global variable or a variable local to the lifeline on which the assignment is placed (not shown in our textual syntax), while Expression is a mathematical expression and Constraint an expression that evaluates to *true* or *false*. If an operand of guarded alt or guarded xalt does not contain an explicit guard, we interpret this as being the guard *true*.

$\langle \text{Interaction} \rangle$	$\rightarrow \langle \text{Empty} \rangle \mid \langle \text{Event} \rangle \mid \langle \text{Weak sequencing} \rangle \mid \langle \text{Refuse} \rangle \mid \langle \text{Assert} \rangle \mid \langle \text{Guarded alt} \rangle \mid \langle \text{Guarded xalt} \rangle \mid \langle \text{Loop} \rangle \mid \langle \text{Assignment} \rangle \mid \langle \text{Constraint} \rangle$
$\langle \text{Assignment} \rangle$	$\rightarrow \text{assign} (\text{ Variable }, \text{ Expression })$
$\langle \text{Constraint} \rangle$	$\rightarrow \text{constr} (\text{ Constraint })$
$\langle \text{Guarded alt} \rangle$	$\rightarrow \text{alt} [\langle \text{Guarded list} \rangle]$
$\langle \text{Guarded xalt} \rangle$	$\rightarrow \text{xalt} [\langle \text{Guarded list} \rangle]$
$\langle \text{Guarded list} \rangle$	$\rightarrow \langle \text{Guarded interaction} \rangle \mid \langle \text{Guarded list} \rangle, \langle \text{Guarded interaction} \rangle$
$\langle \text{Guarded interaction} \rangle$	$\rightarrow \langle \text{Guard} \rangle \rightarrow \langle \text{Interaction} \rangle$
$\langle \text{Guard} \rangle$	$\rightarrow \text{Constraint}$

Figure 12.10: Syntax of interactions with data

In the semantics, we extend the set of trace events with the two special events *write* (for assignments) and *check* (for constraints). We also need the notion of a state. Let *Var* be the set of all variables and *Val* be the set of all variable values. A state σ is then a total function assigning a value to each variable. Formally:

$$\sigma \in \text{Var} \rightarrow \text{Val}$$

For any expression *expr*, we use $\text{expr}(\sigma)$ to denote its value in σ .

12.4.2 Assignment

Explicit specification of variable values may be done by using assignments. In UML 2.0, assignments are written inside a rounded box on the appropriate lifeline, as illustrated in Fig. 12.11.

Semantically, we represent an assignment $\text{var} = \text{expr}$ by the special event $\text{write}(\sigma, \sigma')$ where

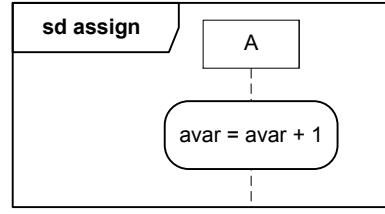


Figure 12.11: Assignment

σ is the state immediately before the assignment and σ' the state immediately after:

$$\begin{aligned} \llbracket \text{assign}(var, expr) \rrbracket &\stackrel{\text{def}}{=} \\ \{ (\langle \langle \text{write}(\sigma, \sigma') \rangle \rangle \mid \sigma'(var) = expr(\sigma) \wedge \\ &\quad \forall v \in \text{Var} : (v = var \vee \sigma'(v) = \sigma(v))\}, \emptyset \} \end{aligned} \quad (12.15)$$

12.4.3 Constraints (State Invariants)

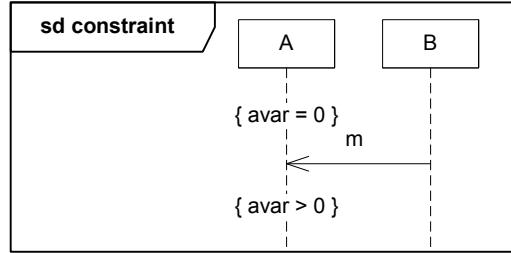


Figure 12.12: Constraint

In UML 2.0, constraints are written within curly brackets, as illustrated in Fig. 12.12. A constraint is a restriction that must be fulfilled by the system, meaning that we have a negative trace if the constraint is broken.

Semantically, a constraint is represented by the special event $\text{check}(\sigma)$, where σ is the state in which the constraint is evaluated:

$$\begin{aligned} \llbracket \text{constr}(c) \rrbracket &\stackrel{\text{def}}{=} \\ \{ (\langle \langle \text{check}(\sigma) \rangle \rangle \mid c(\sigma)\}, \langle \langle \text{check}(\sigma) \rangle \rangle \mid \neg c(\sigma)\} \} \end{aligned} \quad (12.16)$$

This definition ensures that if the constraint is a tautology, then the semantics of $\text{constr}(c)$ has no negative traces, and that a contradiction gives no positive traces:

$$\begin{aligned} \llbracket \text{constr}(true) \rrbracket &= \{ (\langle \langle \text{check}(\sigma) \rangle \rangle \mid true(\sigma)\}, \langle \langle \text{check}(\sigma) \rangle \rangle \mid false(\sigma)\} \} \\ &= \{ (\langle \langle \text{check}(\sigma) \rangle \rangle \mid \sigma \in \text{Var} \rightarrow \text{Val}\}, \emptyset \} \} \end{aligned}$$

$$\begin{aligned}
& \llbracket \text{constr}(false) \rrbracket \\
&= \{(\{\langle \text{check}(\sigma) \rangle \mid \text{false}(\sigma)\}, \{\langle \text{check}(\sigma) \rangle \mid \text{true}(\sigma)\})\} \\
&= \{(\emptyset, \{\langle \text{check}(\sigma) \rangle \mid \sigma \in \text{Var} \rightarrow \text{Val}\})\}
\end{aligned}$$

Notice that one constraint in itself gives potentially an infinite number of system traces, varying with respect to the state component only.

As an example of the use of constraints in an interaction, the complete semantics of the interaction in Fig. 12.12 may be calculated as:

$$\begin{aligned}
& \llbracket \text{constraint} \rrbracket = \\
& \llbracket \text{seq} [\text{constr}(avar = 0), !m, ?m, \text{constr}(avar > 0)] \rrbracket \\
&= (((\llbracket \text{constr}(avar = 0) \rrbracket \gtrsim \llbracket !m \rrbracket) \gtrsim \llbracket ?m \rrbracket) \gtrsim \llbracket \text{constr}(avar > 0) \rrbracket \\
&\quad (\text{Def. (12.7)}) \\
&= (((\llbracket \text{constr}(avar = 0) \rrbracket \gtrsim \{(\{\langle !m \rangle\}, \emptyset)\}) \gtrsim \{(\{\langle ?m \rangle\}, \emptyset)\}) \\
&\quad \gtrsim \llbracket \text{constr}(avar > 0) \rrbracket \\
&\quad (\text{Def. (12.2)}) \\
&= (((\{(\{\langle \text{check}(\sigma) \rangle \mid \sigma(avar) = 0\}, \{\langle \text{check}(\sigma) \rangle \mid \sigma(avar) \neq 0\}) \\
&\quad \gtrsim \{(\{\langle !m \rangle\}, \emptyset)\}) \gtrsim \{(\{\langle ?m \rangle\}, \emptyset)\}) \\
&\quad \gtrsim \{(\{\langle \text{check}(\sigma') \rangle \mid \sigma'(avar) > 0\}, \{\langle \text{check}(\sigma') \rangle \mid \sigma'(avar) \leq 0\})\} \\
&\quad (\text{Def. (12.16)}) \\
&= (\{(\{\langle \text{check}(\sigma) \rangle \mid \sigma(avar) = 0\} \gtrsim \{\langle !m \rangle\}, \\
&\quad \{\langle \text{check}(\sigma) \rangle \mid \sigma(avar) \neq 0\} \gtrsim \{\langle !m \rangle\} \\
&\quad \cup \{\langle \text{check}(\sigma) \rangle \mid \sigma(avar) \neq 0\} \gtrsim \emptyset \\
&\quad \cup \{\langle \text{check}(\sigma) \rangle \mid \sigma(avar) = 0\} \gtrsim \emptyset\}) \\
&\quad \gtrsim \{(\{\langle ?m \rangle\}, \emptyset)\}) \\
&\quad \gtrsim \{(\{\langle \text{check}(\sigma') \rangle \mid \sigma'(avar) > 0\}, \{\langle \text{check}(\sigma') \rangle \mid \sigma'(avar) \leq 0\})\} \\
&\quad (\text{Defs. (12.5) – (12.6)}) \\
&= (\{(\{\langle \text{check}(\sigma), !m \rangle \mid \sigma(avar) = 0\} \\
&\quad \cup \{\langle !m, \text{check}(\sigma) \rangle \mid \sigma(avar) = 0\}, \\
&\quad \{\langle \text{check}(\sigma), !m \rangle \mid \sigma(avar) \neq 0\} \\
&\quad \cup \{\langle !m, \text{check}(\sigma) \rangle \mid \sigma(avar) \neq 0\}) \\
&\quad \gtrsim \{(\{\langle ?m \rangle\}, \emptyset)\}) \\
&\quad \gtrsim \{(\{\langle \text{check}(\sigma') \rangle \mid \sigma'(avar) > 0\}, \{\langle \text{check}(\sigma') \rangle \mid \sigma'(avar) \leq 0\})\} \\
&\quad (\text{Def. (12.3)}) \\
&= \{(\{\langle \text{check}(\sigma), !m, ?m \rangle \mid \sigma(avar) = 0\} \\
&\quad \cup \{\langle !m, \text{check}(\sigma), ?m \rangle \mid \sigma(avar) = 0\}, \\
&\quad \{\langle \text{check}(\sigma), !m, ?m \rangle \mid \sigma(avar) \neq 0\} \\
&\quad \cup \{\langle !m, \text{check}(\sigma), ?m \rangle \mid \sigma(avar) \neq 0\}) \\
&\quad \gtrsim \{(\{\langle \text{check}(\sigma') \rangle \mid \sigma'(avar) > 0\}, \{\langle \text{check}(\sigma') \rangle \mid \sigma'(avar) \leq 0\})\} \\
&\quad (\text{Defs. (12.3) – (12.6)}) \\
&= \{(\{\langle \text{check}(\sigma), !m, ?m, \text{check}(\sigma') \rangle \mid \sigma(avar) = 0 \wedge \sigma'(avar) > 0\} \\
&\quad \cup \{\langle !m, \text{check}(\sigma), ?m, \text{check}(\sigma') \rangle \mid \sigma(avar) = 0 \wedge \sigma'(avar) > 0\}, \\
&\quad \{\langle \text{check}(\sigma), !m, ?m, \text{check}(\sigma') \rangle \mid \sigma(avar) \neq 0 \vee \sigma'(avar) \leq 0\} \\
&\quad \cup \{\langle !m, \text{check}(\sigma), ?m, \text{check}(\sigma') \rangle \mid \sigma(avar) \neq 0 \vee \sigma'(avar) \leq 0\}) \\
&\quad (\text{Defs. (12.3) – (12.6), and formula manipulation})
\end{aligned}$$

12.4.4 Guards (Interaction Constraints)

According to UML 2.0, alternatives (and other combined fragments) in an interaction may be guarded by an interaction constraint (also called a guard). A guard is a special kind of constraint that may only occur at the beginning of the interaction operand in question. As opposed to general constraints, guards are written inside square brackets, as illustrated in Fig. 12.13. As the example illustrates, the guards used in an alt (or xalt) may be overlapping and need not be exhaustive.

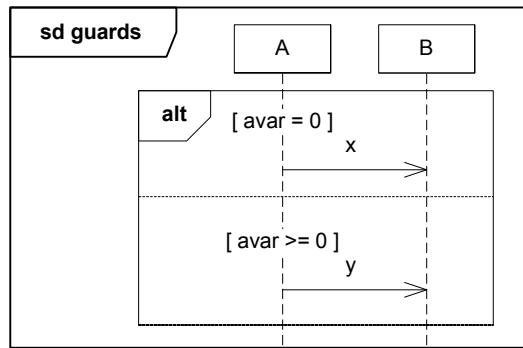


Figure 12.13: Guards

If the guard is true, the interaction operand describes positive traces of the system. The semantics in the case of a false guard is not stated explicitly in the UML 2.0 standard [OMG04]. However, with guards being a specialization of general constraints, it is natural to interpret traces with a false guard as negative. As will be demonstrated in Section 12.5, this is advantageous as it means that adding guards to an alt/xalt-construct constitutes a valid refinement step. A side effect of this is that we will be able to model guards by using the more general notion of constraints as defined in the previous section.

Guarded alt

UML 2.0 [OMG04] states that if none of the operands of an alt construct has a guard that evaluates to true, none of the operands are executed and the remainder of the enclosing interaction is executed. This gives the following semantics for guarded alt:

$$\begin{aligned}
 & [\text{alt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]] \stackrel{\text{def}}{=} \\
 & \{ \biguplus \{o_1, \dots, o_m, (\{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c_j)(\sigma)\}, \emptyset)\} \mid \\
 & \quad \forall i \in [1, m] : o_i \in [\text{seq } [\text{constr}(c_i), d_i]] \}
 \end{aligned} \tag{12.17}$$

The semantics of Fig. 12.13 is informally illustrated in Fig. 12.14. Formally, its complete

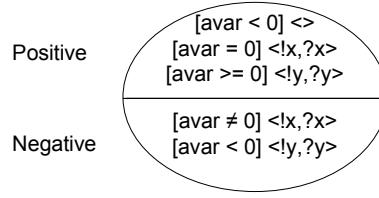


Figure 12.14: Semantics of guarded alt in Fig. 12.13

semantics may be calculated as:

$$\begin{aligned}
 & [\text{guards}] \\
 &= [\text{alt } [avar = 0 \rightarrow \text{seq } [!x, ?x], \\
 &\quad avar \geq 0 \rightarrow \text{seq } [!y, ?y]]] \\
 &= \{ \uplus \{ (\{\langle \text{check}(\sigma), !x, ?x \rangle \mid \sigma(avar) = 0\}, \\
 &\quad \{\langle \text{check}(\sigma), !x, ?x \rangle \mid \sigma(avar) \neq 0\}), \\
 &\quad (\{\langle \text{check}(\sigma), !y, ?y \rangle \mid \sigma(avar) \geq 0\}, \\
 &\quad \{\langle \text{check}(\sigma), !y, ?y \rangle \mid \sigma(avar) < 0\}) \} \mid \text{(Defs. (12.3) -- (12.7),} \\
 &\quad (\{\langle \text{check}(\sigma) \rangle \mid \sigma(avar) < 0\}, \emptyset)\} \} \mid \text{(12.16), (12.17))} \\
 &= \{ (\{\langle \text{check}(\sigma), !x, ?x \rangle \mid \sigma(avar) = 0\} \\
 &\quad \cup \{\langle \text{check}(\sigma), !y, ?y \rangle \mid \sigma(avar) \geq 0\} \\
 &\quad \cup \{\langle \text{check}(\sigma) \rangle \mid \sigma(avar) < 0\}, \\
 &\quad \{\langle \text{check}(\sigma), !x, ?x \rangle \mid \sigma(avar) \neq 0\} \\
 &\quad \cup \{\langle \text{check}(\sigma), !y, ?y \rangle \mid \sigma(avar) < 0\} \} \mid \text{(Def. (12.11))} \\
 \end{aligned}$$

Definition (12.17) giving the semantics of guarded alt is consistent with definition (12.10) of unguarded alt in Section 12.2.3. In our new setting, a specification $\text{alt } [D]$ without guards is interpreted as the specification $\text{alt } [D']$ where D' is the same list of interactions as D , each one guarded by *true*. Calculating this semantics using definition (12.17), gives us the same semantics as definition (12.10) when abstracting away all *check*-events. This is proved in Appendix 12.C.

Guarded xalt

We define the semantics of guarded xalt as:

$$[\text{xalt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]] \stackrel{\text{def}}{=} \bigcup_{i \in [1, m]} [\text{seq } [\text{constr}(c_i), d_i]] \quad (12.18)$$

Unlike guarded alt, the semantics of guarded xalt does not implicitly include the case where all guards are false, since xalt is used to specify explicit choices that must be present in the implementation.

As an example, the semantics of Fig. 12.13 with alt replaced by xalt, is informally illustrated

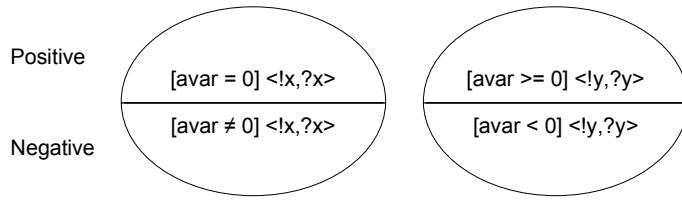


Figure 12.15: Semantics of guarded xalt in Fig. 12.13

in Fig. 12.15. Formally, its complete semantics may be calculated as:

$$\begin{aligned}
 & \llbracket \text{guards} \rrbracket \\
 &= \llbracket \text{xalt } [avar = 0 \rightarrow \text{seq } [!x, ?x], \\
 &\quad avar \geq 0 \rightarrow \text{seq } [!y, ?y]] \rrbracket \\
 &= \bigcup \{ \{(\langle check(\sigma), !x, ?x \rangle \mid \sigma(avar) = 0\}, \\
 &\quad \{\langle check(\sigma), !x, ?x \rangle \mid \sigma(avar) \neq 0\}), \\
 &\quad \{(\langle check(\sigma), !y, ?y \rangle \mid \sigma(avar) \geq 0\}, \quad (\text{Defs. (12.3) -- (12.7)}, \\
 &\quad \{\langle check(\sigma), !y, ?y \rangle \mid \sigma(avar) < 0\})\} \} \quad (12.16), (12.18) \\
 &= \{ (\langle check(\sigma), !x, ?x \rangle \mid \sigma(avar) = 0\}, \\
 &\quad \{\langle check(\sigma), !x, ?x \rangle \mid \sigma(avar) \neq 0\}), \\
 &\quad (\langle check(\sigma), !y, ?y \rangle \mid \sigma(avar) \geq 0\}, \\
 &\quad \{\langle check(\sigma), !y, ?y \rangle \mid \sigma(avar) < 0\}) \} \quad (\text{Def. } \bigcup)
 \end{aligned}$$

As for alt, removing the guards in definition (12.18) gives the original xalt-semantics in definition (12.12) as proved in Appendix 12.C.

12.5 Refinement

In this section we discuss some important aspects of refinement in the setting of STAIRS. Section 12.5.1 gives the necessary background for this discussion, presenting the main refinement definitions. In the rest of Section 12.5 we focus on underspecification and inherent nondeterminism, and how guards may be introduced as a refinement step in both cases.

12.5.1 Background: Formal Definitions

Refinement means to add information to a specification such that the specification becomes more complete. This may be achieved by categorizing inconclusive traces as either positive or negative, or by reducing the set of positive traces. Negative traces always remain negative. A specification may also become more complete by introducing more details.

Glass-Box Refinement

Formally, an interaction obligation (p', n') is a refinement of an interaction obligation (p, n) , written $(p, n) \rightsquigarrow_r (p', n')$, i

$$n \subseteq n' \quad \wedge \quad p \subseteq p' \cup n' \quad (12.19)$$

An interaction d' is a glass-box refinement of an interaction d , written $d \rightsquigarrow_g d'$, i

$$\forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow_r o' \quad (12.20)$$

Theorem *The refinement operator \rightsquigarrow_g is*

- *reflexive:* $d \rightsquigarrow_g d$
- *transitive:* $d \rightsquigarrow_g d' \wedge d' \rightsquigarrow_g d'' \Rightarrow d \rightsquigarrow_g d''$
- *monotonic with respect to refuse, loop, seq, (guarded) alt and (guarded) xalt:*

$$\begin{aligned} d \rightsquigarrow_g d' &\Rightarrow \text{refuse}[d] \rightsquigarrow_g \text{refuse}[d'] \\ d \rightsquigarrow_g d' &\Rightarrow \text{loop } I[d] \rightsquigarrow_g \text{loop } I[d'] \\ d_1 \rightsquigarrow_g d'_1, \dots, d_m \rightsquigarrow_g d'_m &\Rightarrow \text{seq}[d_1, \dots, d_m] \rightsquigarrow_g \text{seq}[d'_1, \dots, d'_m] \\ d_1 \rightsquigarrow_g d'_1, \dots, d_m \rightsquigarrow_g d'_m &\Rightarrow \text{alt}[d_1, \dots, d_m] \rightsquigarrow_g \text{alt}[d'_1, \dots, d'_m] \\ d_1 \rightsquigarrow_g d'_1, \dots, d_m \rightsquigarrow_g d'_m &\Rightarrow \text{xalt}[d_1, \dots, d_m] \rightsquigarrow_g \text{xalt}[d'_1, \dots, d'_m] \end{aligned}$$

Proof

Reflexivity, transitivity and monotonicity with respect to seq, loop and unguarded alt and xalt is proved in [HHR06]. Monotonicity with respect to refuse and guarded alt and xalt is proved in Appendix 12.E. \square

By definition (12.20), new interaction obligations may be freely added to the specification, thus increasing the mandatory nondeterminism required of an implementation. Adding new obligations is an important aspect of the STAIRS methodology. Sometimes, however, it is desirable to restrict this possibility.

A more restrictive notion of refinement is *limited glass-box refinement*, where each obligation in the new refined interaction must correspond to an obligation in the original interaction.

Formally, an interaction d' is a limited glass-box refinement of an interaction d , written $d \rightsquigarrow_l d'$, i

$$d \rightsquigarrow_g d' \wedge \forall o' \in \llbracket d' \rrbracket : \exists o \in \llbracket d \rrbracket : o \rightsquigarrow_r o' \quad (12.21)$$

The refinement theorem above is valid also when replacing \rightsquigarrow_g with \rightsquigarrow_l , as proved in Appendix 12.D (reflexivity and transitivity) and Appendix 12.E (monotonicity).

Notice that a step of refinement may still increase the total number of obligations, but only if two different obligations in $\llbracket d' \rrbracket$ refine the same obligation in $\llbracket d \rrbracket$.

Methodologically, a STAIRS specification would typically be developed by using \rightsquigarrow_g initially and switching to the more restrictive \rightsquigarrow_l after the desired level of nondeterminism in the specification has been reached.

Black-Box Refinement

Black-box refinement may be understood as refinement restricted to the externally visible behaviour. We define the function

$$\text{ext} \in \mathcal{H} \times \mathbb{P}(\mathcal{L}) \rightarrow \mathcal{H}$$

to yield the trace obtained from the trace given as first argument by filtering away those events that are internal with respect to the set of lifelines given as second argument:

$$\text{ext}(h, l) \stackrel{\text{def}}{=} \{e \in \mathcal{E} \mid \text{tr.e} \notin l \vee \text{re.e} \notin l\} \odot h \quad (12.22)$$

The *ext* operator is overloaded to sets of traces, to pairs of sets of traces, and sets of pairs of sets of traces in the standard pointwise manner:

$$\text{ext}(s, l) \stackrel{\text{def}}{=} \{\text{ext}(h, l) \mid h \in s\} \quad (12.23)$$

$$\text{ext}((p, n), l) \stackrel{\text{def}}{=} (\text{ext}(p, l), \text{ext}(n, l)) \quad (12.24)$$

$$\text{ext}(O, l) \stackrel{\text{def}}{=} \{\text{ext}((p, n), l) \mid (p, n) \in O\} \quad (12.25)$$

An interaction d' is a black-box refinement of an interaction d , written $d \rightsquigarrow_b d'$, if

$$\forall o \in \text{ext}(\llbracket d \rrbracket, ll.d) : \exists o' \in \text{ext}(\llbracket d' \rrbracket, ll.d') : o \rightsquigarrow_r o' \quad (12.26)$$

The refinement theorem above is valid also when replacing \rightsquigarrow_g with \rightsquigarrow_b , as the properties are independent of the content of the actual traces.

Black-box refinements will often include lifeline decompositions that are not externally visible. Some lifeline decompositions may also be externally visible due to a change in the sender or receiver of a message. We have already used this in Fig. 12.8, where the network S was decomposed into several nodes. Formally, an interaction d' is a lifeline decomposition of an interaction d with respect to a lifeline substitution ls , written $d \rightsquigarrow_l^{ls} d'$, if

$$d \rightsquigarrow_b \text{subst}(d', ls) \quad (12.27)$$

where $ls \in L \rightarrow L$ is a function defining the lifeline substitution and the function $\text{subst}(d, ls)$ yields the interaction d with every lifeline l in d substituted with the lifeline $ls(l)$.

12.5.2 Adding Positive Behaviour

We now return to our running example from Section 12.3. Even with two different communication paths, we have no guarantee that any of them will be available at a certain time. This is made explicit in Fig. 12.16, where the empty diagram (i.e. skip) is added as a third operand to the `xalt`-construct. When this operand is selected, we get a positive trace consisting of only two events, the transmission of m from A to G, and the reception at G. No further communication will take place, and B will never receive the message.

The semantics of N_Comm is illustrated in Fig. 12.17. Comparing this with Fig. 12.9, which illustrates the semantics of S_Comm (Fig. 12.8), we see that every interaction obligation given by S_Comm is also an interaction obligation by N_Comm. By definitions (12.19)–(12.20), this means that the modified specification is a valid refinement of the original one, S_Comm \rightsquigarrow_g N_Comm. The last obligation in Fig. 12.17 illustrates that new obligations may be added freely when using standard glass-box refinement, \rightsquigarrow_g .

Assume now that our communication network describes the emergency network used by the police, that a police officer needs to communicate, but that the communication for some reason

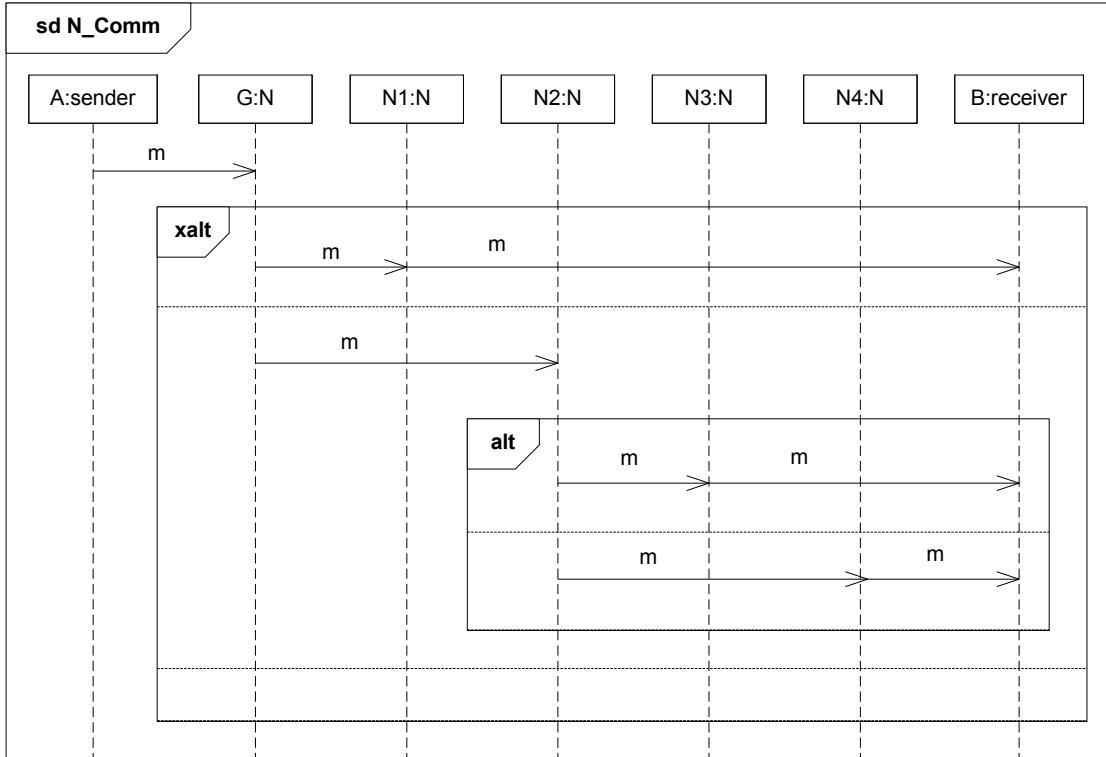


Figure 12.16: Refinement by adding behaviour

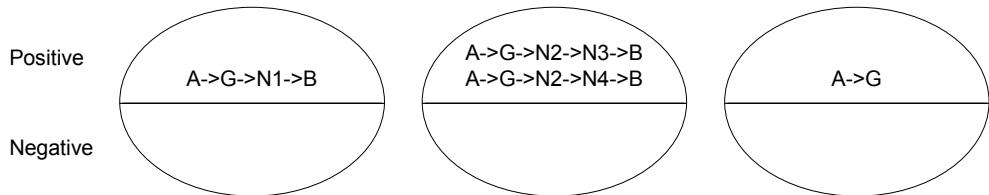


Figure 12.17: Semantics of N_Comm (Fig. 12.16)

fails. In practice, a police officer may grab his personal mobile phone and call his colleague. This is not a mandatory choice (the police are not set up with personal mobile phones), but may be used as an alternative. The resulting specification is shown in Fig. 12.18. The opt-construct is a high-level operator, which may be defined as

$$\text{opt } d \stackrel{\text{def}}{=} d \text{ alt skip} \quad (12.28)$$

The modified specification affects only the last of the interaction obligations in Fig. 12.17, where a positive behaviour is added as illustrated in Fig. 12.19. By definition (12.19), this is a valid refinement as the negative trace-sets in both interaction obligations are empty and the positive trace-set in the N_Comm one is a subset of the new positive trace set in the M_Comm

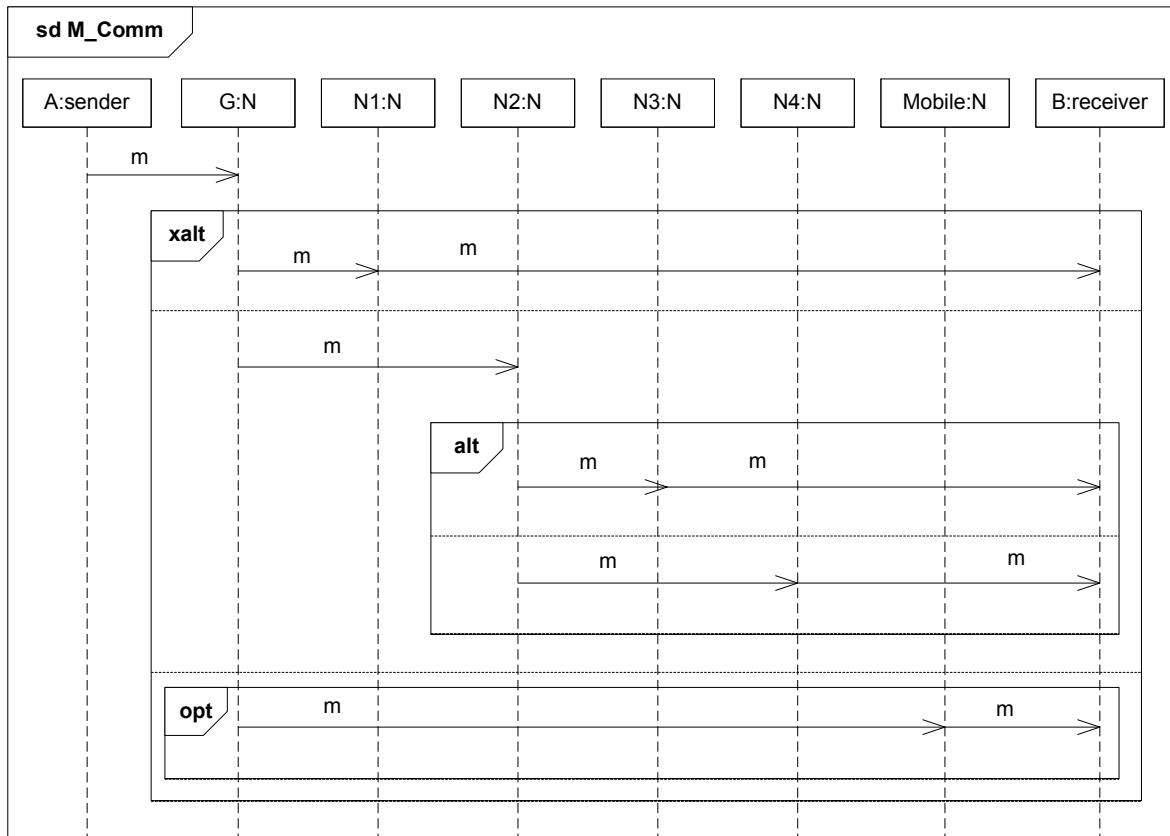


Figure 12.18: Refinement by adding behaviour

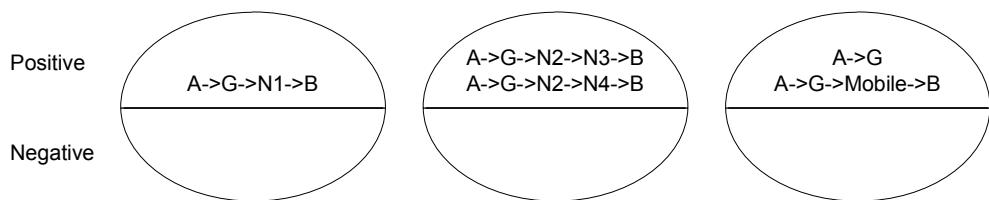


Figure 12.19: Semantics of M_Comm (Fig. 12.18)

one:

$$\begin{aligned}
 & \{ \langle !(m, A, G), ?(m, A, G) \rangle \} \subseteq \\
 & \{ \langle !(m, A, G), ?(m, A, G) \rangle, \\
 & \quad \langle !(m, A, G), ?(m, A, G), !(m, G, \text{Mobile}), ?(m, G, \text{Mobile}), \\
 & \quad \quad !(m, \text{Mobile}, B), ?(m, \text{Mobile}, B) \rangle \}
 \end{aligned}$$

Notice that adding an extra lifeline (the mobile phone) to the interaction is unproblematic, as all traces involving this new lifeline were considered inconclusive in the original interaction.

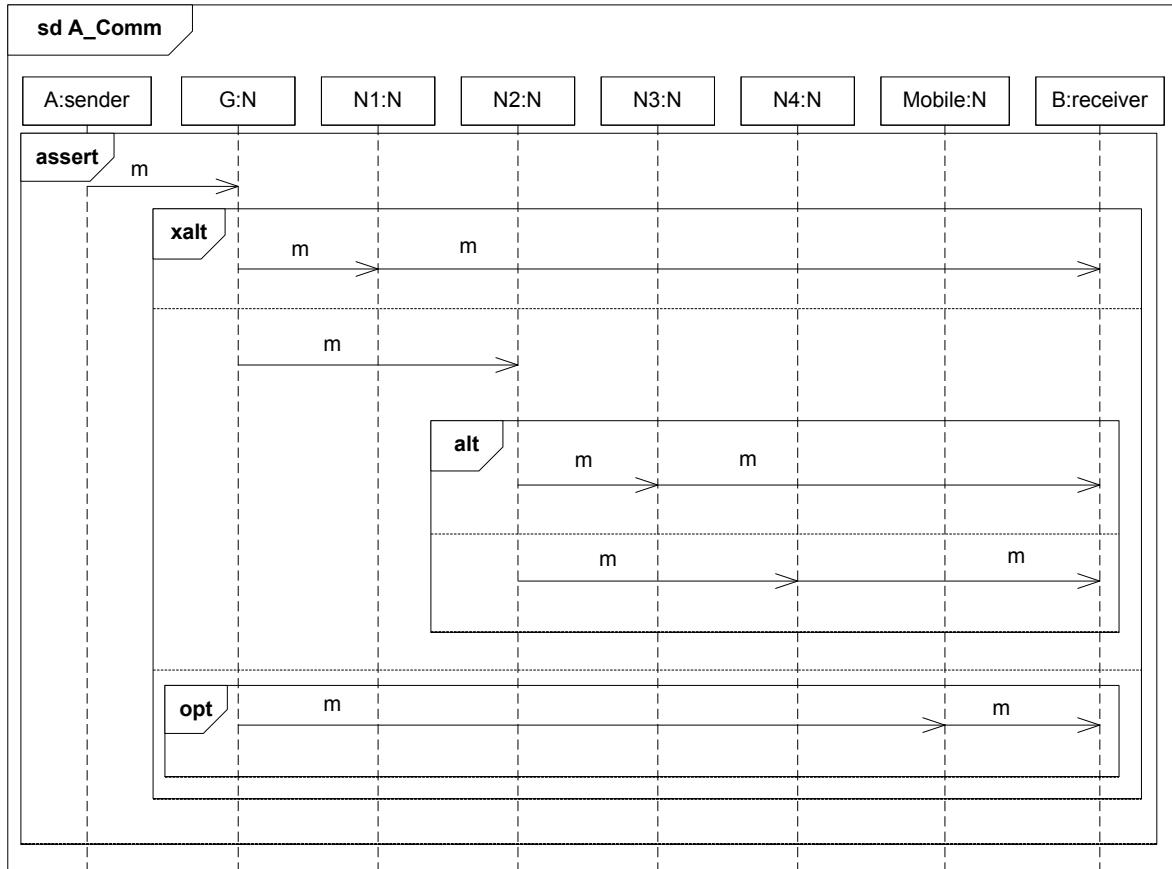


Figure 12.20: Adding negative behaviour

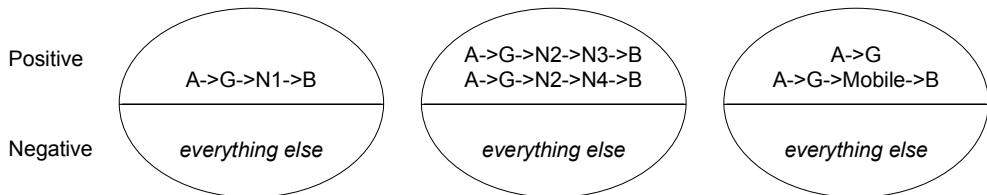


Figure 12.21: Semantics of A_Comm (Fig. 12.20)

12.5.3 Adding Negative Behaviour

The refinement examples in the previous section categorized earlier inconclusive traces as positive. Similarly, earlier inconclusive traces may be categorized as negative, either by specifying the negative traces explicitly through the use of refuse, or by using assert. In our network example, we decide that M_Comm is a complete description of the possible behaviours, and that everything not in the interaction should be considered negative. This gives us the interaction in Fig. 12.20.

The semantics of A_Comm is illustrated in Fig. 12.21. Comparing this with Fig. 12.19,

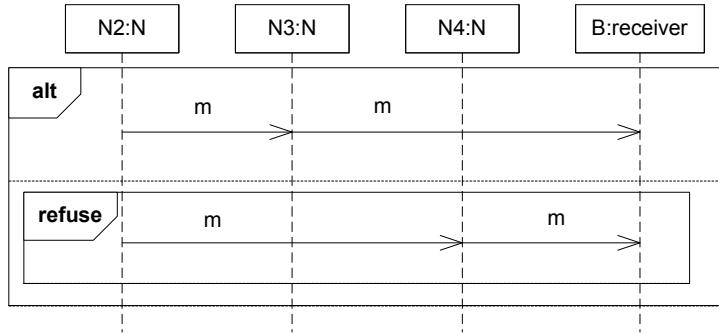


Figure 12.22: Redefining positive behaviour as negative

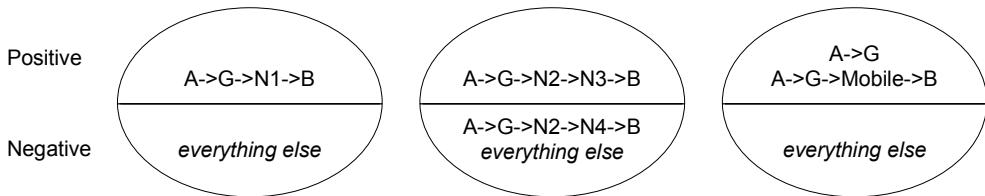


Figure 12.23: Semantics of A_Comm with the refinement in Fig. 12.22

which illustrates the semantics of M_Comm, we see that A_Comm is obviously a refinement of M_Comm, as we have the same positive trace-sets for both specifications and the original empty negative trace-sets are subsets of any set.

12.5.4 Redefining Positive Behaviour as Negative

Refinement may also be used to reduce the set of positive traces by redefining them as negative. Looking at the specification in Fig. 12.20, we may decide that there really is no need to have both communication choices specified by the alt-construct. A refinement of this sub-specification could then be as given by Fig. 12.22. The complete semantics for this refinement is illustrated in Fig. 12.23. We see that the refined specification only affects the obligation in the middle. By definition (12.19), this is a valid refinement step as the negative trace-set is extended and the traces that were previously positive are now either positive or negative.

Another possible refinement of A_Comm could be to specify how the choice between the different communication paths should be made. In the case of our emergency network, using a mobile phone should only be an option if the main network fails. In the interaction in Fig. 12.24, the node G makes the choice between the different alternatives specified by the xalt-construct. Similarly, N2 makes the choice between the alt-operands.

We have assumed that G and N2 have information about the capacity of the different nodes. This may in practice be achieved either by continuous information back from the nodes (not shown in the described behaviour) or through evaluating the communication historically relative to known parameters of the nodes. For our purpose, it is not significant how G and N2 get their data. It is interesting, however, that for xalt the two first guards may both be true, both false,

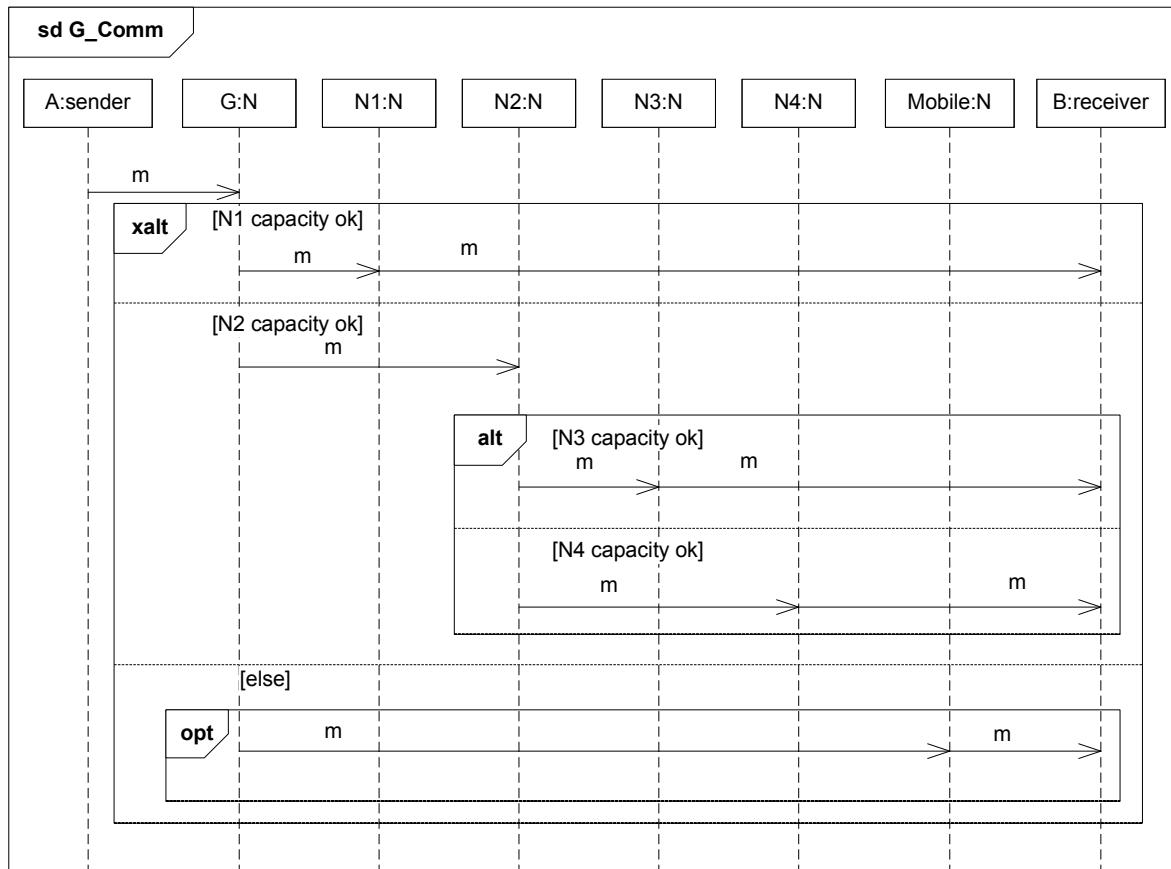


Figure 12.24: Introducing guards

or one true and one false. All of these situations represent cases in real life. If both guards are true, the choice between the two paths may be done arbitrarily. If both guards are false, the else operand comes into effect.

We have not specified what should happen if both guards are false in the alt-fragment. However, according to definition (12.17) giving the semantics of guarded alt, this is equal to the empty trace, i.e. no further communication takes place.

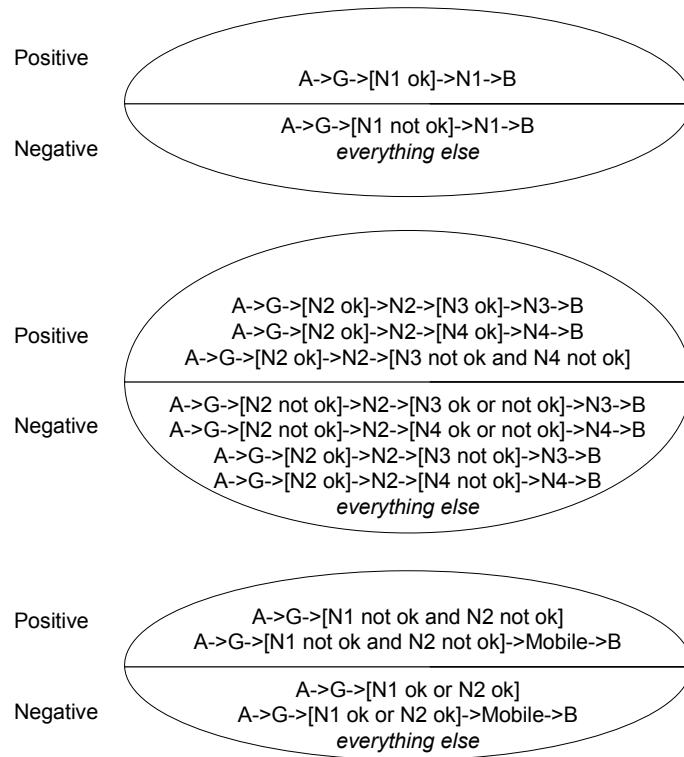


Figure 12.25: Semantics of Fig. 12.24

Fig. 12.25 illustrates the semantics of G_Comm. All traces with a false guard are negative as specified by definitions (12.16)–(12.18). This makes G_Comm a valid refinement of A_Comm. In general, introducing guards in an alt- or xalt-construct will always be a legal refinement step as proved in Appendix 12.E.

12.5.5 Adding More Details

As another example, assume that our sender and receiver suspect that somewhere inside the network there is someone listening to and possibly manipulating their messages. They would like to encrypt their messages and agree (openly) to exchange information to set up a secret key that they shall use for subsequent encryption. Following the procedure outlined by Simon Singh in [Sin99] on how to achieve exchanging of secret keys through insecure communication, we need to be able to describe a number of similar sequences differing basically in the value of some critical numbers.

In Fig. 12.26 we have shown the protocol with a generalized notation for `xalt`. We have supplied the `xalt` with an extra clause which gives one or more parameters with finite domains associated. This generalized notation is identical to replicating the operand for all values of the variable inside the domain.

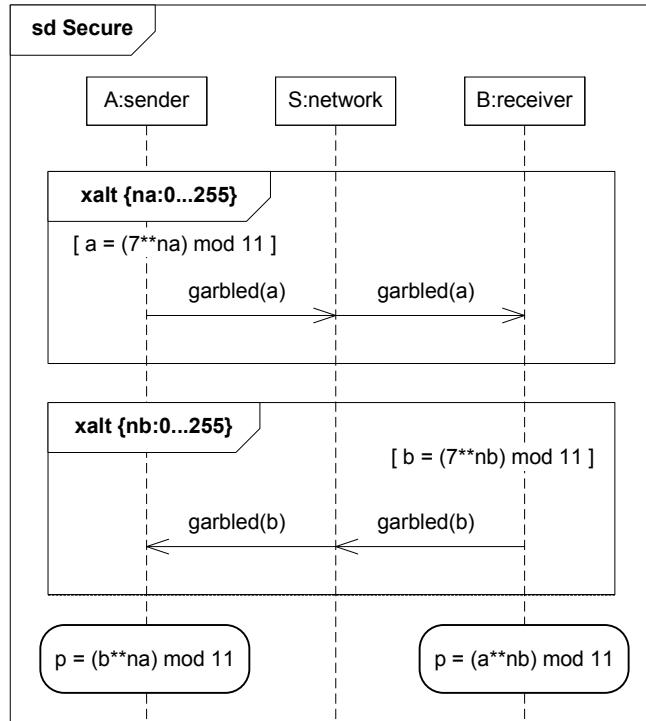


Figure 12.26: Generalized `xalt` for the description of establishing a common secret key

The behaviour of Fig. 12.26 means that the sender chooses a natural number (between 0 and 255 in this example) and from that calculates another natural (here in the range 0...10), and this calculated number is transmitted over the insecure network to the receiver. The receiver does exactly the same the other way with a number that he/she chooses. From the numbers that they initially chose and the numbers that they received from each other, they are able to calculate a common key, p . This key is secret since the network does not have sufficient information to calculate it directly. (Of course, in a real situation the one-way function will be more complicated and the numbers far larger.)

To give a couple of concrete examples, we assume in Fig. 12.27 that the sender has only the naturals 2 and 3 to choose from, while the receiver chooses only from 4 and 5. The specification in Fig. 12.27 gives rise to four interaction obligations (with $p = 1, 5, 9$ or 10), one for each possible combination of values for the two lifelines. The choice between these should be nondeterministic, giving the intruder four possible values for the key. With more alternatives for na and nb , as in the original specification, we get a lot of obligations and potential keys making it difficult for the intruder to find the correct key by plain guessing or by trial-and-error.

In Fig. 12.28 we indicate a possible decomposition of the sender A in the first `xalt`-construct in Fig. 12.26. A is decomposed into a random generator and a sender lifeline C. The generator

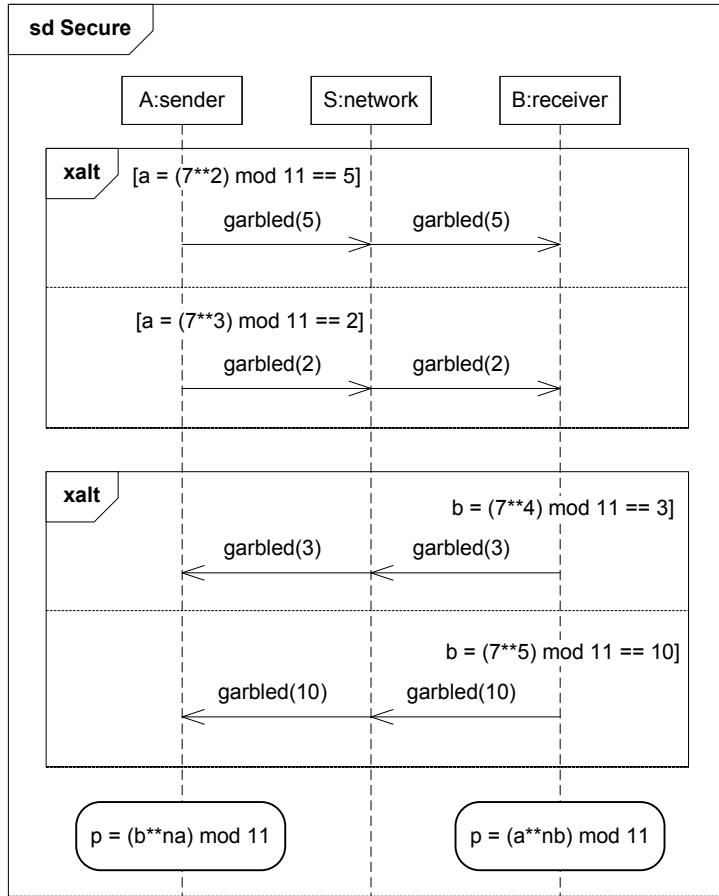


Figure 12.27: A few example-values for the generalized xalt

loops a sufficient number of times, each time sending either 0 or 1 to the sender. Taken together, these messages will constitute the binary representation of the number na in Fig. 12.26. Using **xalt** here, means that both 0 and 1 must be possible in each round in the loop, giving a totally nondeterministic choice for na .

Simple calculations show that we will get the same possible values for na in both diagrams, leading to the same obligations and the same values of the parameter a in both cases, meaning that the decomposition is indeed a valid refinement.

12.6 Implementation

In this section we explain what we mean by an implementation and what it means for an implementation to be correct with respect to a STAIRS specification.

Intuitively, if the specification has only one interaction obligation, a correct implementation may only produce traces belonging to the positive and inconclusive trace sets of the obligation, i.e. no negative trace must be produced by the implementation. With more than one interaction obligation, we may in general find the same trace being positive in one obligation while negative in another.

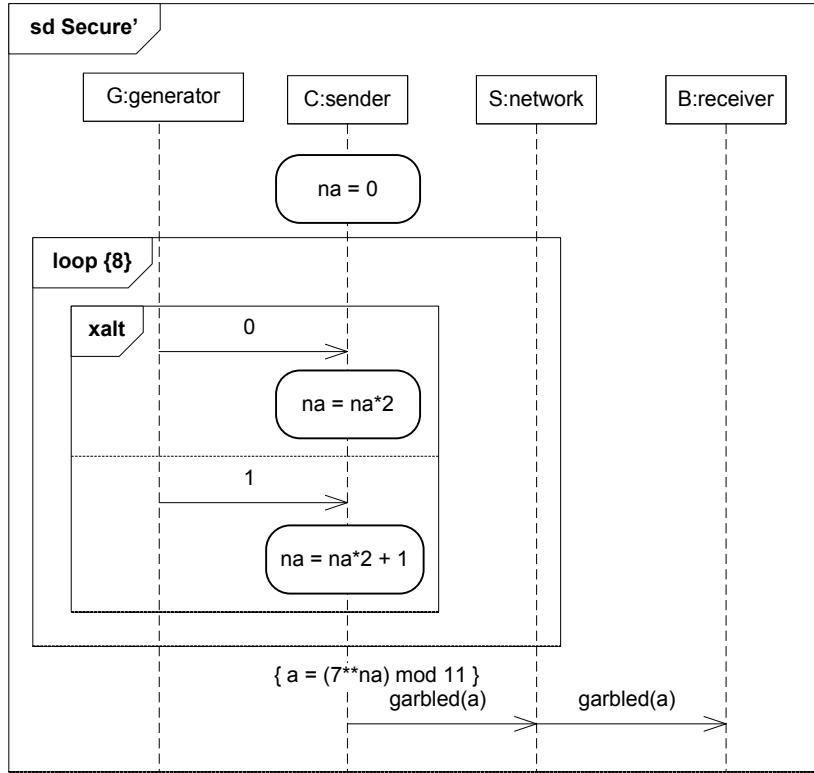


Figure 12.28: Refining generalized xalt by loop (and xalt)

Semantically, we represent implementations in the same way as we represent interactions, namely by sets of interaction obligations. From a semantic point of view, an implementation is a special kind of specification characterized by the following three criteria:

- Its interaction obligations contain no inconclusive traces. Hence, each interaction obligation is of the form $(p, \mathcal{H} \setminus p)$, where $p \neq \emptyset$.
- Whatever typecorrect input it receives from its environment it has at least one output (doing nothing is for example also a response). This means that for any possible environment behaviour, the implementation has at least one trace that is consistent with this behaviour. This corresponds to the notion of winning strategy in F [BS01].
- It behaves causally. Its behaviour at any point in time depends only on what has happened in its past. This is obviously a characteristic of any real-life system (but not necessarily a characteristic of a specification expressed by an interaction). This corresponds to the notion of strong causality in F [BS01].

We say that an implementation I implements a STAIRS specification S if and only if I is a limited refinement of S , i.e. $\llbracket S \rrbracket \rightsquigarrow_l \llbracket I \rrbracket$. This means that an implementation may not add interaction obligations beyond those given by the specification.

12.7 Conclusions

In this paper we have explored different kinds of nondeterminism and underspecification, and motivated the need for having two different operators (`alt` and `xalt`) for specifying alternative behaviours. Basically, `alt` defines implicit nondeterminism in the sense of underspecification or abstraction, while `xalt` defines inherent nondeterminism in the form of explicit choices that must all be present in a valid implementation. We claim that together, these two operators are sufficient to capture the necessary distinctions.

In this paper we have also proposed an extension to STAIRS making it possible to use guards to choose between both implicit (specified by `alt`) and explicit (specified by `xalt`) nondeterminism. In particular, the proposed semantics ensures that adding guards to a specification is a valid refinement step. It is straightforward to combine this extension with Timed STAIRS [HRS05a], which extends STAIRS with time and three-event semantics.

12.7.1 Related Work

Most formalisms do not distinguish between nondeterminism and underspecification as we have done here. In [WM01], Walicki and Meldal makes a similar distinction in the setting of algebraic specifications. Their main motivation is that underspecification may sometimes in fact lead to overspecification, and that in these cases it would be better to use explicit nondeterminism.

In LSC (Live Sequence Charts) [DH99, HM03], charts, locations, messages and conditions may all be characterized as either mandatory or provisional. Provisional charts are called existential and they may happen if their initial condition holds. This is comparable to potential alternatives in STAIRS. Mandatory charts in LSC are called universal. Their interpretation is that provided their initial condition holds, these charts must happen. A universal chart specifies all allowed traces, and is therefore *not* the same as mandatory alternatives in STAIRS, which only specifies some of the traces that must be present in an implementation.

In [CK04], Cengarle and Knapp define the semantics of UML 2.0 interactions by notions of positive and negative satisfaction. This approach has many similarities with ours, but they do not distinguish between underspecification and explicit nondeterminism as we do in STAIRS. With respect to negative traces, their semantics is somewhat different from ours. For alternatives, they define that a trace is negative only if it is negative in both operands. Also, they define that for all possible traces, the trace is negative if a prefix of it is specified as negative, even though the complete trace itself is not described by the diagram. This allows for earlier identification of negative traces. In contrast, we regard such a trace as inconclusive, arguing that if a trace is not described in the diagram, then the specifier has either not thought about the situation or not wanted to classify it as either positive or negative.

In this paper we have modelled data in interactions indirectly through special events representing its use in assignments, constraints, and guards. An example of an alternative approach may be found in [JP01], where Jonsson and Padilla define a global semantics for an MSC (Message Sequence Chart) by using an Abstract Execution Machine. Here, data are included in the model by associating with each instance an environment consisting of its local variables together with those received as message parameters.

Acknowledgements. The research on which this paper reports has partly been carried out within

the context of the IKT-2010 project SARDAS (15295/431) funded by Research Council of Norway. We thank Rolv Bræk, Birger Møller Pedersen, Knut Eilif Husa, Mass Soldal Lund, Atle Refsdal, Judith Rossebø, Manfred Broy, Ina Schieferdecker, Thomas Weigert and the anonymous reviewers for helpful feedback.

References

- [BS01] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.
- [CK04] M. V. Cengerle and A. Knapp. UML 2.0 interactions: semantics and refinement. In Jan Jürjens, Eduardo B. Fernandez, Robert France, and Bernhard Rumpe, editors, *Proc. 3rd Int. Wsh. Critical Systems Development with UML*, Technical report TUM-I0415, pages 85–99. Institut für Informatik, Technische Universität München, 2004.
- [CM88] K. M. Chandy and J. Misra. *Parallel Program Design, A Foundation*. Addison-Wesley, 1988.
- [DH99] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. In *Proc. Formal Methods for Open Object-Based Distributed Systems*, pages 293–311. Kluwer, 1999.
- [HHR05a] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. In S. Leue and T. J. Systä, editors, *Scenarios: Models, Transformations and Tools*, volume 3466 of *LNCS*, pages 1–25. Springer, 2005.
- [HHR05b] Ø. Haugen, K.E. Husa, R.K. Runde, and K. Stølen. Stairs towards formal design with sequence diagrams. *Journal of Software and Systems Modeling*, 4(4):349–458, 2005.
- [HHR06] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309, Department of Informatics, University of Oslo, 2006.
- [HM03] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.
- [HS03] Ø. Haugen and K. Stølen. STAIRS — Steps to analyze interactions with refinement semantics. In *Proc. International Conference on UML*, volume 2863 of *LNCS*, pages 388–402. Springer, 2003.
- [Jac89] J. Jacob. On the derivation of secure components. In *Proc. IEEE Symposium on Security and Privacy*, pages 242–247. IEEE Press, 1989.
- [JL00] R. Joshi and K.R.M. Leino. A semantic approach to secure information flow. *Science of Computer Programming*, 37:113–138, 2000.
- [JP01] B. Jonsson and G. Padilla. An execution semantics for MSC-2000. In R. Reed and J. Reed, editors, *Proc. SDL Forum*, volume 2078 of *LNCS*, pages 365–378. Springer, 2001.
- [OMG04] Object Management Group. *UML 2.0 Superstructure Specification*, document: ptc/04-10-02 edition, 2004.
- [RHS05] R.K. Runde, Ø. Haugen, and K. Stølen. How to transform UML neg into a useful construct. In *Proc. Norsk Informatikkonferanse NIK'2005*, pages 55–66. Tapir, 2005.
- [Ros95] A. W. Roscoe. CSP and determinism in security modelling. In *Proc. IEEE Symposium on Security and Privacy*, pages 114–127. IEEE Press, 1995.
- [Sin99] S. Singh. *The Code Book: the Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Fourth Estate, London, 1999.

[WM01] M. Walicki and S. Meldal. Nondeterminism vs. underspecification. In *Proc. World Multi-Conference on Systemics, Cybernetics and Informatics*, 2001.

12.A Refinement by Adding Assignments and Constraints

Intuitively, adding assignments to an interaction means adding more information to the specification and should therefore be considered a valid refinement step. Similarly, adding constraints means redefining positive behaviours as negative and should also be a valid refinement step.

As assignments and constraints are not part of the externally visible behaviour, adding new assignments and constraints will always constitute a black-box refinement according to definition (12.26). However, black-box refinement is often not sufficient as it in general also allows removal of assignments and constraints.

By definition (12.20) of glass-box refinement, removing assignments and constraints are not allowed. Constraints may be strengthened, but adding assignments and constraints is not allowed as this means inserting new events into the traces.

In order to allow addition, but not removal, of assignments and constraints, we need a refinement relation which requires that the traces of the original interaction are traces also of the refinement when abstracting away a *subset* of the concrete *check-* and *write*-events.

Formally, we first define \mathcal{C} and \mathcal{W} to be the set of all possible *check*- and *write*-events, respectively:

$$\mathcal{C} \stackrel{\text{def}}{=} \{ \text{check}(\sigma) \mid \sigma \in \text{Var} \rightarrow \text{Val} \} \quad (12.29)$$

$$\mathcal{W} \stackrel{\text{def}}{=} \{ \text{write}(\sigma, \sigma') \mid \sigma, \sigma' \in \text{Var} \rightarrow \text{Val} \} \quad (12.30)$$

We then define the function $\text{filt}(t)$ to be the set of all traces that are equal to the trace t when removing a subset of the *check*- and *write*-events in t :

$$\text{filt}(t) \stackrel{\text{def}}{=} \{ h \in \mathcal{H} \mid h \triangleleft t \wedge \overline{(\mathcal{C} \cup \mathcal{W})} \circledcirc h = \overline{(\mathcal{C} \cup \mathcal{W})} \circledcirc t \} \quad (12.31)$$

where $\overline{(\mathcal{C} \cup \mathcal{W})}$ means set complement (i.e. all events not in \mathcal{C} or \mathcal{W}) and $h \triangleleft t$ means that h is a subtrace of t (but not necessarily a consecutive subsequence), formally defined by:

$$h_1 \triangleleft h_2 \stackrel{\text{def}}{=} \exists p \in \{1, 2\}^\infty : \pi_2((\{1\} \times \mathcal{E}) \circledcirc (p, h_2)) = h_1 \quad (12.32)$$

where A^∞ is the set of all infinite sequences over the set A , π_2 is a projection operator returning the second element of a pair, and the filtering operator \circledcirc is a generalization of \circledcirc , filtering pairs of sequences with respect to pairs of elements (for a formal definition of \circledcirc , see [BS01]). The infinite sequence p may be understood as an oracle, determining which of the events in h_2 that are present in the subtrace h_1 .

The filt function is overloaded to sets of traces in standard pointwise manner, i.e.:

$$\text{filt}(s) \stackrel{\text{def}}{=} \bigcup_{t \in s} \text{filt}(t) \quad (12.33)$$

Finally, we redefine refinement of interaction obligations by:

$$(p, n) \rightsquigarrow_r (p', n') \stackrel{\text{def}}{=} n \subseteq \text{filt}(n') \wedge p \subseteq \text{filt}(p') \cup \text{filt}(n') \quad (12.34)$$

12.B Identity of skip

Lemma 1 For all syntactically well-formed interactions d :

$$\forall (p, n) \in \llbracket d \rrbracket : \{\langle\rangle\} \succsim p = p \wedge \{\langle\rangle\} \succsim n = n$$

P : By induction on the structure of d .

B :

a. C : $d = \text{skip}$

$$\langle 1 \rangle 1. \llbracket \text{skip} \rrbracket = \{(\{\langle\rangle\}, \emptyset)\}$$

P : Definition (12.1) of skip.

$$\langle 1 \rangle 2. \{\langle\rangle\} \succsim p = p, \text{i.e. } \{\langle\rangle\} \succsim \{\langle\rangle\} = \{\langle\rangle\}$$

P : $\langle 1 \rangle 1$ and definition (12.3) of \succsim .

$$\langle 1 \rangle 3. \{\langle\rangle\} \succsim n = n, \text{i.e. } \{\langle\rangle\} \succsim \emptyset = \emptyset$$

P : $\langle 1 \rangle 1$ and definition (12.3) of \succsim .

$\langle 1 \rangle 4.$ Q.E.D.

b. C : $d = e$, where e is an event,

$d = \text{assign}(var, expr)$, or

$d = \text{constr}(c)$

P : All cases follow the same pattern, defined below.

L : $ev = e$

$cond_1 = \text{true}$

$cond_2 = \text{false}$

in the case $d = e$

L : $ev = \text{write}(\sigma, \sigma')$

$cond_1 =$

$$(\sigma'(var) = expr(\sigma) \wedge \forall v \in Var : (v = var \vee \sigma'(v) = \sigma(v)))$$

$cond_2 = \text{false}$

in the case $d = \text{assign}(var, expr)$

L : $ev = \text{check}(\sigma)$

$cond_1 = c(\sigma)$

$cond_2 = \neg c(\sigma)$

in the case $d = \text{constr}(c)$

$$\langle 1 \rangle 1. \llbracket d \rrbracket = \{(\{\langle ev \rangle \mid cond_1\}, \{\langle ev \rangle \mid cond_2\})\}$$

P : Definition (12.2) of an event, definition (12.15) of assign, or definition (12.16) of constr.

$$\langle 1 \rangle 2. \{\langle\rangle\} \succsim \{\langle evt \rangle \mid c\} = \{\langle evt \rangle \mid c\} \text{ for arbitrary event } evt \text{ and arbitrary condition } c.$$

$\langle 2 \rangle 1.$ Case: c is a contradiction, i.e. $\{\langle evt \rangle \mid c\} = \emptyset$.

P : $\{\langle\rangle\} \succsim \emptyset = \emptyset$ by definition (12.3) of \succsim .

$\langle 2 \rangle 2.$ Case: c is not a contradiction

P : For all traces t , $\langle \rangle \cap t = t$, giving
 $\{\langle \rangle\} \asymp \{\langle evt \rangle \mid c\} = \{\langle evt \rangle \mid c\}$ by definition (12.3) of \asymp .
(2)3. Q.E.D.

P : The cases are exhaustive.
(1)3. $\{\langle \rangle\} \asymp p = p$, i.e. $\{\langle \rangle\} \asymp \{\langle ev \rangle \mid cond_1\} = \{\langle ev \rangle \mid cond_1\}$
P : $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$.
(1)4. $\{\langle \rangle\} \asymp n = n$, i.e. $\{\langle \rangle\} \asymp \{\langle ev \rangle \mid cond_2\} = \{\langle ev \rangle \mid cond_2\}$
P : $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$.
(1)5. Q.E.D.

I :

L : d be an interaction constructed from the (sub-)interactions d_1, \dots, d_m using the composition operators defined in this paper.

A : $\forall i \in [1, m] : \forall (p_i, n_i) \in \llbracket d_i \rrbracket : \{\langle \rangle\} \asymp p_i = p_i \wedge \{\langle \rangle\} \asymp n_i = n_i$ (induction hypothesis)
P : $\forall (p, n) \in \llbracket d \rrbracket : \{\langle \rangle\} \asymp p = p \wedge \{\langle \rangle\} \asymp n = n$, i.e.
 $\{\langle \rangle\} \asymp p = p \wedge \{\langle \rangle\} \asymp n = n$ for arbitrary $(p, n) \in \llbracket d \rrbracket$ by \forall -rule.

c. C : $d = \text{seq } [D]$, D a list of interactions

P : By induction on the length of D .

(1)1. Base case: $D = d_1$, i.e. $d = \text{seq } [d_1]$

(2)1. $\llbracket \text{seq } [d_1] \rrbracket = \llbracket d_1 \rrbracket$
P : Definition (12.7) of seq.

(2)2. Q.E.D.

P : (2)1 and the induction hypothesis.

(1)2. Induction step: $D = D'$, d_i for $i \in [1, m]$, i.e. $d = \text{seq } [D', d_i]$

A : $\forall (p', n') \in \llbracket \text{seq } [D'] \rrbracket : \{\langle \rangle\} \asymp p' = p' \wedge \{\langle \rangle\} \asymp n' = n'$
(induction hypothesis 2)

P : $\{\langle \rangle\} \asymp p = p \wedge \{\langle \rangle\} \asymp n = n$ for arbitrary
 $(p, n) \in \llbracket \text{seq } [D', d_i] \rrbracket$

(2)1. Choose $(p', n') \in \llbracket \text{seq } [D'] \rrbracket$ and $(p_i, n_i) \in \llbracket d_i \rrbracket$ such that $p = p' \asymp p_i$ and
 $n = n' \asymp p_i \cup n' \asymp n_i \cup p' \asymp n_i$

P : Definitions (12.5)–(12.7) of seq.

(2)2. $\{\langle \rangle\} \asymp p = p$, i.e. $\{\langle \rangle\} \asymp (p' \asymp p_i) = p' \asymp p_i$

P : (2)1 and associativity of \asymp (lemma 11 in [HHR06]), which gives that
 $\{\langle \rangle\} \asymp (p' \asymp p_i)$ is equal to $(\{\langle \rangle\} \asymp p') \asymp p_i$, which is equal to $p' \asymp p_i$ by
induction hypothesis 2.

(2)3. $\{\langle \rangle\} \asymp n = n$,

$$\begin{aligned} &\text{i.e. } \{\langle \rangle\} \asymp (n' \asymp p_i \cup n' \asymp n_i \cup p' \asymp n_i) \\ &= n' \asymp p_i \cup n' \asymp n_i \cup p' \asymp n_i \end{aligned}$$

(3)1. $\{\langle \rangle\} \asymp (n' \asymp p_i) = n' \asymp p_i$

P : Associativity of \asymp (lemma 11 in [HHR06]) and induction hypothesis 2.

(3)2. $\{\langle \rangle\} \asymp (n' \asymp n_i) = n' \asymp n_i$

P : Associativity of \asymp (lemma 11 in [HHR06]) and induction hypothesis 2.

$\langle 3 \rangle 3. \{ \langle \rangle \} \succsim (p' \succsim n_i) = p' \succsim n_i$

P : Associativity of \succsim (lemma 11 in [HHR06]) and induction hypothesis 2.

$\langle 3 \rangle 4. \text{Q.E.D.}$

P : $\langle 2 \rangle 1$, distributivity of \succsim over \cup (lemma 14 in [HHR06]) and associativity of \cup .

$\langle 2 \rangle 4. \text{Q.E.D.}$

$\langle 1 \rangle 3. \text{Q.E.D.}$

d. C : $d = \text{refuse } [d_1]$

$\langle 1 \rangle 1. [\text{refuse } [d_1]] = \{(\emptyset, p_1 \cup n_1) \mid (p_1, n_1) \in [d_1]\}$

P : Definition (12.8) of refuse .

$\langle 1 \rangle 2. \{ \langle \rangle \} \succsim p = p$, i.e. $\{ \langle \rangle \} \succsim \emptyset = \emptyset$

P : $\langle 1 \rangle 1$ and definition (12.3) of \succsim .

$\langle 1 \rangle 3. \{ \langle \rangle \} \succsim n = n$, i.e. $\{ \langle \rangle \} \succsim (p_1 \cup n_1) = p_1 \cup n_1$

$\langle 2 \rangle 1. \{ \langle \rangle \} \succsim (p_1 \cup n_1) = (\{ \langle \rangle \} \succsim p_1) \cup (\{ \langle \rangle \} \succsim n_1)$

P : By distributivity of \succsim over \cup (lemma 14 in [HHR06]).

$\langle 2 \rangle 2. \{ \langle \rangle \} \succsim p_1 = p_1$

P : The induction hypothesis.

$\langle 2 \rangle 3. \{ \langle \rangle \} \succsim n_1 = n_1$

P : The induction hypothesis.

$\langle 2 \rangle 4. \text{Q.E.D.}$

P : $\langle 1 \rangle 1$ and $\langle 2 \rangle 1$ – $\langle 2 \rangle 3$.

$\langle 1 \rangle 4. \text{Q.E.D.}$

e. C : $d = \text{assert } [d_1]$

$\langle 1 \rangle 1. \text{Choose } (p_1, n_1) \in [d_1] \text{ such that } p = p_1 \text{ and } n = n_1 \cup (\mathcal{H} \setminus p_1)$

P : Definition (12.9) of assert .

$\langle 1 \rangle 2. \{ \langle \rangle \} \succsim p = p$, i.e. $\{ \langle \rangle \} \succsim p_1 = p_1$

P : $\langle 1 \rangle 1$ and the induction hypothesis.

$\langle 1 \rangle 3. \{ \langle \rangle \} \succsim n = n$, i.e. $\{ \langle \rangle \} \succsim (n_1 \cup (\mathcal{H} \setminus p_1)) = n_1 \cup (\mathcal{H} \setminus p_1)$

$\langle 2 \rangle 1. \{ \langle \rangle \} \succsim (n_1 \cup (\mathcal{H} \setminus p_1)) = (\{ \langle \rangle \} \succsim n_1) \cup (\{ \langle \rangle \} \succsim (\mathcal{H} \setminus p_1))$

P : By distributivity of \succsim over \cup (lemma 14 in [HHR06]).

$\langle 2 \rangle 2. \{ \langle \rangle \} \succsim n_1 = n_1$

P : The induction hypothesis.

$\langle 2 \rangle 3. \{ \langle \rangle \} \succsim (\mathcal{H} \setminus p_1) = \mathcal{H} \setminus p_1$

$\langle 3 \rangle 1. \mathcal{H} \setminus p_1 \subseteq \{ \langle \rangle \} \succsim (\mathcal{H} \setminus p_1)$

P : Definition (12.3) of \succsim and $\langle \rangle \cap t = t$ for all traces t .

$\langle 3 \rangle 2. \{ \langle \rangle \} \succsim (\mathcal{H} \setminus p_1) \subseteq \mathcal{H} \setminus p_1$

P : Proof by contradiction.

A : 1. $h \in \{ \langle \rangle \} \succsim (\mathcal{H} \setminus p_1)$

2. $h \notin \mathcal{H} \setminus p_1$, i.e. $h \in p_1$

P : false

$\langle 4 \rangle 1. h \in \{ \langle \rangle \} \succsim p_1$

P : Assumption 2 and the induction hypothesis.

- $\langle 4 \rangle 2.$ Choose $h_2 \in \mathcal{H} \setminus p_1$ such that
 $\forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc \langle \rangle \sim e.l \circledcirc h_2$
P : Assumption 1 and definition (12.3) of \sim .
- $\langle 4 \rangle 3.$ Choose $h'_2 \in p_1$ such that
 $\forall l \in \mathcal{L} : e.l \circledcirc h = e.l \circledcirc \langle \rangle \sim e.l \circledcirc h'_2$
P : $\langle 4 \rangle 1$ and definition (12.3) of \sim .
- $\langle 4 \rangle 4.$ $\forall l \in \mathcal{L} : e.l \circledcirc h_2 = e.l \circledcirc h'_2$
P : $\langle 4 \rangle 2, \langle 4 \rangle 3$ and $\langle \rangle \sim t = t$ for all traces t .
- $\langle 4 \rangle 5.$ $\{\langle \rangle\} \sim p_1 =$
 $\{h' \in \mathcal{H} \mid \exists h'_2 \in p_1 : \forall l \in \mathcal{L} : e.l \circledcirc h' = e.l \circledcirc h'_2\} = p_1$
P : The induction hypothesis, definition (12.3) of \sim and $\langle \rangle \sim t = t$ for all traces t .
- $\langle 4 \rangle 6.$ $\{h' \in \mathcal{H} \mid \forall l \in \mathcal{L} : e.l \circledcirc h' = e.l \circledcirc h'_2\} \subseteq p_1$
P : $\langle 4 \rangle 3$ and $\langle 4 \rangle 5$.
- $\langle 4 \rangle 7.$ $\{h' \in \mathcal{H} \mid \forall l \in \mathcal{L} : e.l \circledcirc h' = e.l \circledcirc h_2\} \subseteq p_1$
P : $\langle 4 \rangle 4$ and $\langle 4 \rangle 6$.
- $\langle 4 \rangle 8.$ $h_2 \in p_1$
P : $\langle 4 \rangle 7, h_2 \in \{h' \in \mathcal{H} \mid \forall l \in \mathcal{L} : e.l \circledcirc h' = e.l \circledcirc h_2\}$ and elementary set theory ($x \in A \wedge A \subseteq B \Rightarrow x \in B$).
- $\langle 4 \rangle 9.$ Q.E.D.
P : Contradiction by $\langle 4 \rangle 2$ and $\langle 4 \rangle 8$.
- $\langle 3 \rangle 3.$ Q.E.D.
P : By elementary set theory ($A \subseteq B \wedge B \subseteq A \Rightarrow A = B$).
- $\langle 2 \rangle 4.$ Q.E.D.
P : $\langle 1 \rangle 1$ and $\langle 2 \rangle 1 - \langle 2 \rangle 3$.
- $\langle 1 \rangle 4.$ Q.E.D.
- f. C : $d = \text{alt}[d_1, \dots, d_m]$
- $\langle 1 \rangle 1.$ For all $i \in [1, m]$, choose $(p_i, n_i) \in \llbracket d_i \rrbracket$ such that $p = \bigcup_{i \in [1, m]} p_i$ and $n = \bigcup_{i \in [1, m]} n_i$
P : Definition (12.10) of alt and definition (12.11) of \sqcup .
- $\langle 1 \rangle 2.$ $\{\langle \rangle\} \sim p = p$, i.e. $\{\langle \rangle\} \sim \bigcup_{i \in [1, m]} p_i = \bigcup_{i \in [1, m]} p_i$
 $\langle 2 \rangle 1.$ $\{\langle \rangle\} \sim \bigcup_{i \in [1, m]} p_i = \bigcup_{i \in [1, m]} (\{\langle \rangle\} \sim p_i)$
P : Distributivity of \sim over \cup (lemma 14 in [HHR06]).
- $\langle 2 \rangle 2.$ $\bigcup_{i \in [1, m]} (\{\langle \rangle\} \sim p_i) = \bigcup_{i \in [1, m]} p_i$
P : The induction hypothesis.
- $\langle 2 \rangle 3.$ Q.E.D.
P : $\langle 1 \rangle 1$ and $\langle 2 \rangle 1 - \langle 2 \rangle 2$.
- $\langle 1 \rangle 3.$ $\{\langle \rangle\} \sim n = n$, i.e. $\{\langle \rangle\} \sim \bigcup_{i \in [1, m]} n_i = \bigcup_{i \in [1, m]} n_i$
 $\langle 2 \rangle 1.$ $\{\langle \rangle\} \sim \bigcup_{i \in [1, m]} n_i = \bigcup_{i \in [1, m]} (\{\langle \rangle\} \sim n_i)$
P : Distributivity of \sim over \cup (lemma 14 in [HHR06]).
- $\langle 2 \rangle 2.$ $\bigcup_{i \in [1, m]} (\{\langle \rangle\} \sim n_i) = \bigcup_{i \in [1, m]} n_i$
P : The induction hypothesis.
- $\langle 2 \rangle 3.$ Q.E.D.

P : $\langle 1 \rangle 1$ and $\langle 2 \rangle 1 - \langle 2 \rangle 2$.
 $\langle 1 \rangle 4$. Q.E.D.

g. C : $d = \text{xalt } [d_1 \dots d_m]$

$\langle 1 \rangle 1$. Choose $i \in [1, m]$ such that $(p, n) \in [\![\text{seq}[\text{constr}(c_i), d_i]]\!]$

P : Definition (12.12) of xalt .

$\langle 1 \rangle 2$. $\{\langle \rangle\} \lesssim p = p$

P : $\langle 1 \rangle 1$ and the induction hypothesis.

$\langle 1 \rangle 3$. $\{\langle \rangle\} \lesssim n = n$

P : $\langle 1 \rangle 1$ and the induction hypothesis.

$\langle 1 \rangle 4$. Q.E.D.

h. C : $d = \text{alt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]$

$\langle 1 \rangle 1$. For all $i \in [1, m]$, choose $(p'_i, n'_i) \in [\![\text{seq}[\text{constr}(c_i), d_i]]\!]$ such that $p = \bigcup_{i \in [1, m]} p'_i \cup \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c_j)(\sigma)\}$ and $n = \bigcup_{i \in [1, m]} n'_i$

P : Definition (12.17) of guarded alt and definition (12.11) of \biguplus .

$\langle 1 \rangle 2$. $\forall i \in [1, m] : \forall (p'_i, n'_i) \in [\![\text{seq}[\text{constr}(c_i), d_i]]\!] :$

$\{\langle \rangle\} \lesssim p'_i = p'_i \wedge \{\langle \rangle\} \lesssim n'_i = n'_i$

P : The induction hypothesis and the induction cases (c) and (b) of seq and constr.

$\langle 1 \rangle 3$. $\{\langle \rangle\} \lesssim p = p$,

i.e. $\{\langle \rangle\} \lesssim (\bigcup_{i \in [1, m]} p'_i \cup \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c_j)(\sigma)\}) = \bigcup_{i \in [1, m]} p'_i \cup \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c_j)(\sigma)\}$

$\langle 2 \rangle 1$. $\{\langle \rangle\} \lesssim \bigcup_{i \in [1, m]} p'_i = \bigcup_{i \in [1, m]} (\{\langle \rangle\} \lesssim p'_i)$

P : Distributivity of \lesssim over \cup (lemma 14 in [HHR06]).

$\langle 2 \rangle 2$. $\bigcup_{i \in [1, m]} (\{\langle \rangle\} \lesssim p'_i) = \bigcup_{i \in [1, m]} p'_i$

P : $\langle 1 \rangle 2$.

$\langle 2 \rangle 3$. $\{\langle \rangle\} \lesssim \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c_j)(\sigma)\} = \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c_j)(\sigma)\}$

P : Definition (12.3) of \lesssim .

$\langle 2 \rangle 4$. Q.E.D.

P : $\langle 1 \rangle 1$, $\langle 2 \rangle 1 - \langle 2 \rangle 3$ and distributivity of \lesssim over \cup (lemma 14 in [HHR06]).

$\langle 1 \rangle 4$. $\{\langle \rangle\} \lesssim n = n$, i.e. $\{\langle \rangle\} \lesssim \bigcup_{i \in [1, m]} n'_i = \bigcup_{i \in [1, m]} n'_i$

$\langle 2 \rangle 1$. $\{\langle \rangle\} \lesssim \bigcup_{i \in [1, m]} n'_i = \bigcup_{i \in [1, m]} (\{\langle \rangle\} \lesssim n'_i)$

P : Distributivity of \lesssim over \cup (lemma 14 in [HHR06]).

$\langle 2 \rangle 2$. $\bigcup_{i \in [1, m]} (\{\langle \rangle\} \lesssim n'_i) = \bigcup_{i \in [1, m]} n'_i$

P : $\langle 1 \rangle 2$.

$\langle 2 \rangle 3$. Q.E.D.

P : $\langle 1 \rangle 1$, $\langle 2 \rangle 1$ and $\langle 2 \rangle 2$.

$\langle 1 \rangle 5$. Q.E.D.

i. C : $d = \text{xalt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]$

$\langle 1 \rangle 1$. Choose $i \in [1, m]$ such that $(p, n) \in [\![\text{seq}[\text{constr}(c_i), d_i]]\!]$

P : Definition (12.18) of guarded xalt.

$\langle 1 \rangle 2$. $\forall i \in [1, m] : \forall (p'_i, n'_i) \in [\![\text{seq}[\text{constr}(c_i), d_i]]\!] :$

- $\{\langle\rangle\} \succsim p'_i = p'_i \wedge \{\langle\rangle\} \succsim n'_i = n'_i$
- P : The induction hypothesis and the induction cases (c) and (b) of seq and constr.
- (1)3. $\{\langle\rangle\} \succsim p = p$
P : (1)1 and (1)2.
- (1)4. $\{\langle\rangle\} \succsim n = n$
P : (1)1 and (1)2.
- (1)5. Q.E.D.

j. C : $d = \text{loop } I [d_1]$

- (1)1. For all $j \in I$, choose $(p_j, n_j) \in \mu_j[\![d_1]\!]$ such that $p = \bigcup_{j \in I} p_j$ and $n = \bigcup_{j \in I} n_j$
P : Definition (12.14) of loop and definition (12.11) of \bigcup .
- (1)2. $\forall j \in I : \forall (p_j, n_j) \in \mu_j[\![d_1]\!] : \{\langle\rangle\} \succsim p_j = p_j \wedge \{\langle\rangle\} \succsim n_j = n_j$
P : By induction on j .
- (2)1. Base case: $j = 0$
(3)1. $(p_j, n_j) = (\{\langle\rangle\}, \emptyset)$
P : Definition (12.13) of μ_0 .
- (3)2. $\{\langle\rangle\} \succsim p_j = p_j$, i.e. $\{\langle\rangle\} \succsim \{\langle\rangle\} = \{\langle\rangle\}$
P : (3)1 and definition (12.3) of \succsim .
- (3)3. $\{\langle\rangle\} \succsim n_j = n_j$, i.e. $\{\langle\rangle\} \succsim \emptyset = \emptyset$
P : (3)1 and definition (12.3) of \succsim .
- (3)4. Q.E.D.
- (2)2. Base case: $j = 1$
(3)1. $(p_j, n_j) \in [\![d_1]\!]$
P : Definition (12.13) of μ_0 .
- (3)2. $\{\langle\rangle\} \succsim p_j = p_j$
P : (3)1 and the induction hypothesis.
- (3)3. $\{\langle\rangle\} \succsim n_j = n_j$
P : (3)1 and the induction hypothesis.
- (3)4. Q.E.D.
- (2)3. Induction step: $j > 1$
A : $\forall (p_j, n_j) \in \mu_{j-1}[\![d_1]\!] : \{\langle\rangle\} \succsim p_j \wedge \{\langle\rangle\} \succsim n_j = n_j$
P : $\forall (p_j, n_j) \in \mu_j[\![d_1]\!] : \{\langle\rangle\} \succsim p_j \wedge \{\langle\rangle\} \succsim n_j = n_j$, i.e. $\{\langle\rangle\} \succsim p_j \wedge \{\langle\rangle\} \succsim n_j = n_j$ for arbitrary $(p_j, n_j) \in \mu_j[\![d_1]\!]$ by \forall -rule.
- (3)1. $(p_j, n_j) \in \mu_{j-1}[\![d_1]\!] \succsim [\![d_1]\!]$
P : Definition (12.13) of μ_n .
- (3)2. Choose $(p'_j, n'_j) \in \mu_{j-1}[\![d_1]\!]$ and $(p''_j, n''_j) \in [\![d_1]\!]$ such that $p_j = p'_j \succsim p''_j$ and $n_j = n'_j \succsim p''_j \cup n'_j \succsim n''_j \cup p'_j \succsim n''_j$
P : (3)1 and definitions (12.5)–(12.6) of \succsim .
- (3)3. $\{\langle\rangle\} \succsim p_j = p_j$, i.e. $\{\langle\rangle\} \succsim (p'_j \succsim p''_j) = p'_j \succsim p''_j$
P : (3)1 and associativity of \succsim (lemma 11 in [HHR06]), which gives that $\{\langle\rangle\} \succsim (p'_j \succsim p''_j)$ is equal to $(\{\langle\rangle\} \succsim p'_j) \succsim p''_j$, which is equal to $p'_j \succsim p''_j$ by induction hypothesis 2.
- (3)4. $\{\langle\rangle\} \succsim n_j = n_j$,

i.e. $\{\langle\rangle\} \succsim (n'_j \succsim p''_j \cup n'_j \succsim n''_j \cup p'_j \succsim n''_j)$
 $= n'_j \succsim p''_j \cup n'_j \succsim n''_j \cup p'_j \succsim n''_j$

$\langle 4 \rangle 1.$ $\{\langle\rangle\} \succsim (n'_j \succsim p''_j) = n'_j \succsim p''_j$
P : Associativity of \succsim (lemma 11 in [HHR06]) and induction hypothesis
2.

$\langle 4 \rangle 2.$ $\{\langle\rangle\} \succsim (n'_j \succsim n''_j) = n'_j \succsim n''_j$
P : Associativity of \succsim (lemma 11 in [HHR06]) and induction hypothesis
2.

$\langle 4 \rangle 3.$ $\{\langle\rangle\} \succsim (p'_j \succsim n''_j) = p'_j \succsim n''_j$
P : Associativity of \succsim (lemma 11 in [HHR06]) and induction hypothesis
2.

$\langle 4 \rangle 4.$ Q.E.D.
P : $\langle 3 \rangle 1$, distributivity of \succsim over \cup (lemma 14 in [HHR06]) and associativity of \cup .

$\langle 3 \rangle 5.$ Q.E.D.

$\langle 2 \rangle 4.$ Q.E.D.

$\langle 1 \rangle 3.$ $\{\langle\rangle\} \succsim p = p$, i.e. $\{\langle\rangle\} \succsim \bigcup_{j \in I} p_j = \bigcup_{j \in I} p_j$

$\langle 2 \rangle 1.$ $\{\langle\rangle\} \succsim \bigcup_{j \in I} p_j = \bigcup_{j \in I} (\{\langle\rangle\} \succsim p_j)$
P : Distributivity of \succsim over \cup (lemma 14 in [HHR06]).

$\langle 2 \rangle 2.$ $\bigcup_{j \in I} (\{\langle\rangle\} \succsim p_j) = \bigcup_{j \in I} p_j$
P : The induction hypothesis.

$\langle 2 \rangle 3.$ Q.E.D.
P : $\langle 1 \rangle 1$ and $\langle 2 \rangle 1-\langle 2 \rangle 2$.

$\langle 1 \rangle 4.$ $\{\langle\rangle\} \succsim n = n$, i.e. $\{\langle\rangle\} \succsim \bigcup_{j \in I} n_j = \bigcup_{j \in I} n_j$

$\langle 2 \rangle 1.$ $\{\langle\rangle\} \succsim \bigcup_{j \in I} n_j = \bigcup_{j \in I} (\{\langle\rangle\} \succsim n_j)$
P : Distributivity of \succsim over \cup (lemma 14 in [HHR06]).

$\langle 2 \rangle 2.$ $\bigcup_{j \in I} (\{\langle\rangle\} \succsim n_j) = \bigcup_{j \in I} n_j$
P : The induction hypothesis.

$\langle 2 \rangle 3.$ Q.E.D.
P : $\langle 1 \rangle 1$ and $\langle 2 \rangle 1-\langle 2 \rangle 2$.

$\langle 1 \rangle 5.$ Q.E.D.

□

Lemma 2 For all syntactically well-formed interactions d :

$$\forall (p, n) \in [\![d]\!]: p \succsim \{\langle\rangle\} = p \wedge n \succsim \{\langle\rangle\} = n$$

P : By induction on the structure of d .

P :

All cases except from seq and the induction step for loop are symmetrical to the cases in the proof of lemma 1, but using the symmetrical lemma 15 in [HHR06] instead of lemma 14 in [HHR06]. The cases for seq and loop are treated below.

A : $\forall i \in [1, m] : \forall (p_i, n_i) \in \llbracket d_i \rrbracket : p_i \succsim \{\langle\rangle\} = p_i \wedge n_i \succsim \{\langle\rangle\} = n_i$ (induction hypothesis)

P : $\forall (p, n) \in \llbracket d \rrbracket : p \succsim \{\langle\rangle\} = p \wedge n \succsim \{\langle\rangle\} = n$, i.e.
 $p \succsim \{\langle\rangle\} = p \wedge n \succsim \{\langle\rangle\} = n$ for arbitrary $(p, n) \in \llbracket d \rrbracket$ by \forall -rule.

c. C : $d = \text{seq } [D]$, D a list of interactions

$\langle 1 \rangle 1.$ Case: $D = d_1$, i.e. $d = \text{seq } [d_1]$

$\langle 2 \rangle 1.$ $\llbracket \text{seq } [d_1] \rrbracket = \llbracket d_1 \rrbracket$

P : Definition (12.7) of seq.

$\langle 2 \rangle 2.$ Q.E.D.

P : $\langle 2 \rangle 1$ and the induction hypothesis.

$\langle 1 \rangle 2.$ Case: $D = D'$, d_i for $i \in [1, m]$, i.e. $d = \text{seq } [D', d_i]$

$\langle 2 \rangle 1.$ Choose $(p', n') \in \llbracket \text{seq } [D'] \rrbracket$ and $(p_i, n_i) \in \llbracket d_i \rrbracket$ such that $p = p' \succsim p_i$ and $n = n' \succsim p_i \cup n' \succsim n_i \cup p' \succsim n_i$

P : Definitions (12.5)–(12.7) of seq.

$\langle 2 \rangle 2.$ $p \succsim \{\langle\rangle\} = p$, i.e. $(p' \succsim p_i) \succsim \{\langle\rangle\} = p' \succsim p_i$

P : $\langle 2 \rangle 1$ and associativity of \succsim (lemma 11 in [HHR06]), which gives that $(p' \succsim p_i) \succsim \{\langle\rangle\}$ is equal to $p' \succsim (p_i \succsim \{\langle\rangle\})$, which is equal to $p' \succsim p_i$ by the induction hypothesis.

$\langle 2 \rangle 3.$ $n \succsim \{\langle\rangle\} = n$,

i.e. $(n' \succsim p_i \cup n' \succsim n_i \cup p' \succsim n_i) \succsim \{\langle\rangle\}$

$= n' \succsim p_i \cup n' \succsim n_i \cup p' \succsim n_i$

$\langle 3 \rangle 1.$ $(n' \succsim p_i) \succsim \{\langle\rangle\} = n' \succsim p_i$

P : Associativity of \succsim (lemma 11 in [HHR06]) and the induction hypothesis.

$\langle 3 \rangle 2.$ $(n' \succsim n_i) \succsim \{\langle\rangle\} = n' \succsim n_i$

P : Associativity of \succsim (lemma 11 in [HHR06]) and the induction hypothesis.

$\langle 3 \rangle 3.$ $(p' \succsim n_i) \succsim \{\langle\rangle\} = p' \succsim n_i$

P : Associativity of \succsim (lemma 11 in [HHR06]) and the induction hypothesis.

$\langle 3 \rangle 4.$ Q.E.D.

P : $\langle 2 \rangle 1$, distributivity of \succsim over \cup (lemma 15 in [HHR06]) and associativity of \cup .

$\langle 2 \rangle 4.$ Q.E.D.

$\langle 1 \rangle 3.$ Q.E.D.

P : The cases are exhaustive by definition (12.7) of seq.

j. C : $d = \text{loop } I [d_1]$

$\langle 2 \rangle 3.$ Case: $j > 1$

$\langle 3 \rangle 1.$ $(p_j, n_j) \in \mu_{j-1} \llbracket d_1 \rrbracket \succsim \llbracket d_1 \rrbracket$

P : Definition (12.13) of μ_n .

$\langle 3 \rangle 2.$ Choose $(p'_j, n'_j) \in \mu_{j-1} \llbracket d_1 \rrbracket$ and $(p''_j, n''_j) \in \llbracket d_1 \rrbracket$ such that $p_j = p'_j \succsim p''_j$ and $n_j = n'_j \succsim p''_j \cup n'_j \succsim n''_j \cup p'_j \succsim n''_j$

- P : $\langle 3 \rangle 1$ and definitions (12.5)–(12.6) of \succsim .
- $\langle 3 \rangle 3.$ $p_j \succsim \{\langle \rangle\} = p_j$, i.e. $(p'_j \succsim p''_j) \succsim \{\langle \rangle\} = p'_j \succsim p''_j$
 P : $\langle 3 \rangle 1$ and associativity of \succsim (lemma 11 in [HHR06]), which gives that $(p'_j \succsim p''_j) \succsim \{\langle \rangle\}$ is equal to $p'_j \succsim (p''_j \succsim \{\langle \rangle\})$, which is equal to $p'_j \succsim p''_j$ by the induction hypothesis.
- $\langle 3 \rangle 4.$ $n_j \succsim \{\langle \rangle\} = n_j$,
 i.e. $(n'_j \succsim p''_j \cup n'_j \succsim n''_j \cup p'_j \succsim n''_j) \succsim \{\langle \rangle\}$
 $= n'_j \succsim p''_j \cup n'_j \succsim n''_j \cup p'_j \succsim n''_j$
- $\langle 4 \rangle 1.$ $(n'_j \succsim p''_j) \succsim \{\langle \rangle\} = n'_j \succsim p''_j$
 P : Associativity of \succsim (lemma 11 in [HHR06]) and the induction hypothesis.
- $\langle 4 \rangle 2.$ $(n'_j \succsim n''_j) \succsim \{\langle \rangle\} = n'_j \succsim n''_j$
 P : Associativity of \succsim (lemma 11 in [HHR06]) and the induction hypothesis.
- $\langle 4 \rangle 3.$ $(p'_j \succsim n''_j) \succsim \{\langle \rangle\} = p'_j \succsim n''_j$
 P : Associativity of \succsim (lemma 11 in [HHR06]) and the induction hypothesis.
- $\langle 4 \rangle 4.$ Q.E.D.
 P : $\langle 3 \rangle 1$, distributivity of \succsim over \cup (lemma 15 in [HHR06]) and associativity of \cup .
- $\langle 3 \rangle 5.$ Q.E.D.

□

Theorem 1 skip is the left identity element for weak sequencing

- P : $\text{seq}[\text{skip}, d] = d$
- $\langle 1 \rangle 1.$ $\forall (p, n) \in \llbracket d \rrbracket : \{\langle \rangle\} \succsim p = p \wedge \{\langle \rangle\} \succsim n = n$
 P : Lemma 1
- $\langle 1 \rangle 2.$ $\{(\{\langle \rangle\} \succsim p, \{\langle \rangle\} \succsim n) \mid (p, n) \in \llbracket d \rrbracket\} = \{(p, n) \mid (p, n) \in \llbracket d \rrbracket\}$
 P : $\langle 1 \rangle 1$.
- $\langle 1 \rangle 3.$ $\{(\{\langle \rangle\}, \emptyset)\} \succsim \{(p, n) \mid (p, n) \in \llbracket d \rrbracket\} = \{(p, n) \mid (p, n) \in \llbracket d \rrbracket\}$
 P : $\langle 1 \rangle 2$ and definitions (12.3)–(12.6) of \succsim .
- $\langle 1 \rangle 4.$ $\llbracket \text{skip} \rrbracket \succsim \llbracket d \rrbracket = \llbracket d \rrbracket$
 P : $\langle 1 \rangle 3$, definition (12.1) of skip and elementary set theory
 $(\{a \mid a \in A\} = A \text{ for all sets } A)$.
- $\langle 1 \rangle 5.$ $\llbracket \text{seq}[\text{skip}, d] \rrbracket = \llbracket d \rrbracket$
 P : $\langle 1 \rangle 4$ and definition (12.7) of seq.
- $\langle 1 \rangle 6.$ Q.E.D.

□

Theorem 2 skip is the right identity element for weak sequencing

$$P : \text{seq}[d, \text{skip}] = d$$

$$P :$$

Symmetrical to the proof of theorem 1, using lemma 2 instead of lemma 1.

□

12.C Comparing the Guarded and Unguarded Versions of alt and xalt

Theorem 3 The definitions of guarded and unguarded alt are consistent

Setting the guards to true in definition (12.17) of guarded alt, results in the same semantics as definition (12.10) of unguarded alt when abstracting away all check-events introduced by the guarded alt.

$$P :$$

$$\begin{aligned} & [\![\text{alt} [\text{true} \rightarrow d_1, \dots, \text{true} \rightarrow d_m]]\!] \quad (\text{with abstraction}) \\ &= \{ \biguplus \{ o_1, \dots, o_m, (\{ \langle \rangle \mid (\bigwedge_{j \in [1, m]} \neg \text{true})(\sigma), \emptyset) \} \mid \\ & \quad \forall i \in [1, m] : o_i \in [\![\text{seq} [\text{skip}, d_i]]\!] \} \} \quad (\text{see } (*) \text{ below}) \\ &= \{ \biguplus \{ o_1, \dots, o_m, (\emptyset, \emptyset) \} \mid \forall i \in [1, m] : o_i \in [\![\text{seq} [\text{skip}, d_i]]\!] \} \quad (\text{see } (**)\text{ below}) \\ &= \{ \biguplus \{ o_1, \dots, o_m \} \mid \forall i \in [1, m] : o_i \in [\![\text{seq} [\text{skip}, d_i]]\!] \} \quad (\text{def. (12.11) of } \biguplus) \\ &= \{ \biguplus \{ o_1, \dots, o_m \} \mid \forall i \in [1, m] : o_i \in [\![d_i]\!] \} \quad (\text{lemma 1}) \\ &= [\![\text{alt} [d_1, \dots, d_m]]\!] \quad (\text{def. (12.10) of alt}) \end{aligned}$$

(*) definition (12.17), calculation of constr(true) on page 183, abstracting away all check-events introduced by definition (12.17) and definition (12.1) of skip.

$$\begin{aligned} (**)\ \{ \langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg \text{true})(\sigma) \} &= \{ \langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \text{false})(\sigma) \} \\ &= \{ \langle \text{check}(\sigma) \rangle \mid \text{false}(\sigma) \} = \{ \langle \text{check}(\sigma) \rangle \mid \text{false} \} = \emptyset \end{aligned}$$

□

Theorem 4 The definitions of guarded and unguarded xalt are consistent.

Setting the guards to true in definition (12.18) of guarded xalt, results in the same semantics as definition (12.12) of unguarded xalt when abstracting away all check-events introduced by the guarded xalt.

$$P :$$

$$\begin{aligned} & [\![\text{xalt} [\text{true} \rightarrow d_1, \dots, \text{true} \rightarrow d_m]]\!] \quad (\text{with abstraction}) \\ &= \bigcup_{i \in [1, m]} [\![\text{seq} [\text{skip}, d_i]]\!] \quad (\text{see } (*) \text{ below}) \\ &= \bigcup_{i \in [1, m]} [\![d_i]\!] \quad (\text{theorem 1}) \\ &= [\![\text{xalt} [d_1, \dots, d_m]]\!] \quad (\text{def. (12.12) of xalt}) \end{aligned}$$

(*) definition (12.18), calculation of $\text{constr}(\text{true})$ on page 183, abstracting away all *check*-events introduced by definition (12.18) and definition (12.1) of skip.

□

12.D Reflexivity and Transitivity of Limited Refinement

Lemma 3

For all syntactically well-formed interactions d : $\llbracket d \rrbracket \neq \emptyset$

P : Straightforward by induction on the structure of d .

□

Theorem 5 The limited refinement operator \rightsquigarrow_l is reflexive.

P : $d \rightsquigarrow_l d$

$\langle 1 \rangle 1.$ $d \rightsquigarrow_g d$

P : By reflexivity of \rightsquigarrow_g (theorem 8 in [HHR06]).

$\langle 1 \rangle 2.$ Choose arbitrary $o' \in \llbracket d \rrbracket$

P : $\llbracket d \rrbracket$ is non-empty by lemma 3.

$\langle 1 \rangle 3.$ Choose $o = o'$.

P : $\langle 1 \rangle 2.$

$\langle 1 \rangle 4.$ $o' \rightsquigarrow_r o'$

P : By reflexivity of \rightsquigarrow_r (lemma 25 in [HHR06]).

$\langle 1 \rangle 5.$ $\forall o' \in \llbracket d \rrbracket : \exists o \in \llbracket d \rrbracket : o \rightsquigarrow_r o'$

P : $\langle 1 \rangle 2, \langle 1 \rangle 3$ and $\langle 1 \rangle 4.$

$\langle 1 \rangle 6.$ Q.E.D.

P : $\langle 1 \rangle 1, \langle 1 \rangle 5$ and definition (12.21) of \rightsquigarrow_l .

□

Theorem 6 The limited refinement operator \rightsquigarrow_l is transitive.

A : a. $d \rightsquigarrow_l d'$

b. $d' \rightsquigarrow_l d''$

P : $d \rightsquigarrow_l d''$

$\langle 1 \rangle 1.$ $d \rightsquigarrow_g d''$

P : The assumptions, definition (12.21) of \rightsquigarrow_l , and transitivity of \rightsquigarrow_g (theorem 9 in [HHR06]).

$\langle 1 \rangle 2.$ Choose arbitrary $o'' = (p'', n'') \in \llbracket d'' \rrbracket$

P : $\llbracket d'' \rrbracket$ is non-empty by lemma 3.

$\langle 1 \rangle 3.$ Choose $o' = (p', n') \in \llbracket d' \rrbracket$ such that $(p', n') \rightsquigarrow_r (p'', n'')$

P : $\langle 1 \rangle 2$, assumption 2 and definition (12.21) of \rightsquigarrow_l .

$\langle 1 \rangle 4.$ Choose $o = (p, n) \in \llbracket d \rrbracket$ such that $(p, n) \rightsquigarrow_r (p', n')$

P : $\langle 1 \rangle 3$, assumption 1 and definition (12.21) of \rightsquigarrow_l .

$\langle 1 \rangle 5. (p, n) \rightsquigarrow_r (p'', n'')$

P : $\langle 1 \rangle 4, \langle 1 \rangle 3$ and transitivity of \rightsquigarrow_r (lemma 26 in [HHR06]).

$\langle 1 \rangle 6. \forall o'' \in [\![d'']\!]: \exists o \in [\![d]\!]: o \rightsquigarrow_r o''$

P : $\langle 1 \rangle 2, \langle 1 \rangle 4$ and $\langle 1 \rangle 5$.

$\langle 1 \rangle 7. \text{Q.E.D.}$

P : $\langle 1 \rangle 1, \langle 1 \rangle 6$ and definition (12.21) of \rightsquigarrow_l .

□

12.E Monotonicity Results

General proof sketch for all monotonicity proofs for limited refinement, \rightsquigarrow_l :

The first part of definition (12.21) of \rightsquigarrow_l follows from the assumption(s) and the corresponding monotonicity theorem for general refinement, \rightsquigarrow_g . For the second part of definition (12.21), the proof is symmetrical to the corresponding monotonicity proof for \rightsquigarrow_g . The difference is that for \rightsquigarrow_g we choose an arbitrary interaction obligation for the original interaction and find a refining interaction obligation in the refinement, while for \rightsquigarrow_l we choose an arbitrary interaction obligation in the refining interaction and find a corresponding interaction obligation in the original interaction.

12.E.1 Interactions without Data

Lemma 4 (*To be used when proving monotonicity with respect to refuse.*)

A : $(p, n) \rightsquigarrow_r (p', n')$

P : $(\emptyset, p \cup n) \rightsquigarrow_r (\emptyset, p' \cup n')$

$\langle 1 \rangle 1.$ Requirement 1: $p \cup n \subseteq p' \cup n'$

P : $p \subseteq p' \cup n'$ and $n \subseteq n'$ by the assumption and definition (12.19) of \rightsquigarrow_r .

$\langle 1 \rangle 2.$ Requirement 2: $\emptyset \subseteq \emptyset \cup (p' \cup n')$

P : Trivial.

$\langle 1 \rangle 3. \text{Q.E.D.}$

P : Definition (12.19) of \rightsquigarrow_r .

□

Theorem 7 *Monotonicity of \rightsquigarrow_g with respect to the refuse operator*

A : $d_1 \rightsquigarrow_g d'_1$

P : $\text{refuse } [d_1] \rightsquigarrow_g \text{refuse } [d'_1]$

P : Each obligation $o \in [\![\text{refuse } [d_1]]\!]$ is constructed from an obligation $o_1 \in [\![d_1]\!]$. By the assumption, we may select an obligation $o'_1 \in [\![d'_1]\!]$ such that $o_1 \rightsquigarrow_r o'_1$. Using o'_1 we then construct an obligation $o' \in [\![\text{refuse } [d'_1]]\!]$ and prove by lemma 4 that $o \rightsquigarrow_r o'$.

- $\langle 1 \rangle 1.$ Choose arbitrary $o = (p, n) \in [\text{refuse } [d_1]]$
 P : $[\text{refuse } [d_1]]$ is non-empty by lemma 3.
- $\langle 1 \rangle 2.$ Choose $(p_1, n_1) \in [d_1]$ such that $p = \emptyset$ and $n = p_1 \cup n_1$
 P : $\langle 1 \rangle 1$ and definition (12.8) of refuse.
- $\langle 1 \rangle 3.$ Choose $(p'_1, n'_1) \in [d'_1]$ such that $(p_1, n_1) \rightsquigarrow_r (p'_1, n'_1)$
 P : $\langle 1 \rangle 2$, the assumption and definition (12.20) of \rightsquigarrow_g .
- $\langle 1 \rangle 4.$ $o' = (p', n') = (\emptyset, p'_1 \cup n'_1) \in [\text{refuse } [d'_1]]$
 P : $\langle 1 \rangle 3$ and definition (12.8) of refuse.
- $\langle 1 \rangle 5.$ $(p, n) \rightsquigarrow_r (p', n')$
 P : $\langle 1 \rangle 2, \langle 1 \rangle 3, \langle 1 \rangle 4$ and lemma 4.
- $\langle 1 \rangle 6.$ $\forall o \in [\text{refuse } [d_1]] : \exists o' \in [\text{refuse } [d'_1]] : o \rightsquigarrow_r o'$
 P : $\langle 1 \rangle 1, \langle 1 \rangle 4, \langle 1 \rangle 5$ and \forall -rule.
- $\langle 1 \rangle 7.$ Q.E.D.
 P : $\langle 1 \rangle 6$ and definition (12.20) of \rightsquigarrow_g .

□

Theorem 8 Monotonicity of \rightsquigarrow_l with respect to the refuse operator

- A : $d_1 \rightsquigarrow_l d'_1$
 P : $\text{refuse } [d_1] \rightsquigarrow_l \text{refuse } [d'_1]$
- $\langle 1 \rangle 1.$ $\text{refuse } [d_1] \rightsquigarrow_g \text{refuse } [d'_1]$
 P : The assumption and theorem 7 (monotonicity of \rightsquigarrow_g with respect to refuse).
- $\langle 1 \rangle 2.$ Choose arbitrary $o' = (p', n') \in [\text{refuse } [d'_1]]$
 P : $[\text{refuse } [d'_1]]$ is non-empty by lemma 3.
- $\langle 1 \rangle 3.$ Choose $(p'_1, n'_1) \in [d'_1]$ such that $p' = \emptyset$ and $n' = p'_1 \cup n'_1$
 P : $\langle 1 \rangle 2$ and definition (12.8) of refuse.
- $\langle 1 \rangle 4.$ Choose $(p_1, n_1) \in [d_1]$ such that $(p_1, n_1) \rightsquigarrow_r (p'_1, n'_1)$
 P : $\langle 1 \rangle 3$, the assumption and definition (12.21) of \rightsquigarrow_l .
- $\langle 1 \rangle 5.$ $o = (p, n) = (\emptyset, p_1 \cup n_1) \in [\text{refuse } [d_1]]$
 P : $\langle 1 \rangle 4$ and definition (12.8) of refuse.
- $\langle 1 \rangle 6.$ $(p, n) \rightsquigarrow_r (p', n')$
 P : $\langle 1 \rangle 3, \langle 1 \rangle 4, \langle 1 \rangle 5$ and lemma 4.
- $\langle 1 \rangle 7.$ $\forall o' \in [\text{refuse } [d'_1]] : \exists o \in [\text{refuse } [d_1]] : o \rightsquigarrow_r o'$
 P : $\langle 1 \rangle 2, \langle 1 \rangle 5, \langle 1 \rangle 6$ and \forall -rule.
- $\langle 1 \rangle 8.$ Q.E.D.
 P : $\langle 1 \rangle 1, \langle 1 \rangle 7$ and definition (12.21) of \rightsquigarrow_l .

□

Theorem 9 *Monotonicity of \rightsquigarrow_l with respect to the loop operator*

A : $d_1 \rightsquigarrow_l d'_1$

P : $\text{loop } I [d_1] \rightsquigarrow_l \text{loop } I [d'_1]$

$\langle 1 \rangle 1.$ $\text{loop } I [d_1] \rightsquigarrow_g \text{loop } I [d'_1]$

P : The assumption and monotonicity of \rightsquigarrow_g with respect to loop (theorem 16 in [HHR06]).

$\langle 1 \rangle 2.$ Choose arbitrary $o' = (p', n') \in [\text{loop } I [d'_1]]$

P : $[\text{loop } I [d'_1]]$ is non-empty by lemma 3.

$\langle 1 \rangle 3.$ For all $i \in I$, choose $(p'_i, n'_i) \in \mu_i[d'_1]$ such that

$$p' = \bigcup_{i \in I} p'_i \text{ and } n' = \bigcup_{i \in I} n'_i$$

P : Definition (12.14) of loop and definition (12.11) of \bigcup .

$\langle 1 \rangle 4.$ For all $i \in I$, choose $(p_i, n_i) \in \mu_i[d_1]$ such that $(p_i, n_i) \rightsquigarrow_r (p'_i, n'_i)$

$\langle 2 \rangle 1.$ $\forall i \in I: \forall o' \in \mu_i[d'_1]: \exists o \in \mu_i[d_1]: o \rightsquigarrow_r o'$

P : By induction on i .

$\langle 3 \rangle 1.$ $\exists o \in \mu_i[d_1]: o \rightsquigarrow_r o'$ for arbitrary $i \in I, o' \in \mu_i[d'_1]$

$\langle 4 \rangle 1.$ C : $i = 0$

$\langle 5 \rangle 1.$ $o' = \{\{\langle \rangle\}, \emptyset\}$

P : Definition (12.13) of μ_0 .

$\langle 5 \rangle 2.$ $o' = \{\{\langle \rangle\}, \emptyset\} \in \mu_0[d_1]$

P : Definition (12.13) of μ_0 .

$\langle 5 \rangle 3.$ $o' \rightsquigarrow_r o'$

P : Reflexivity of \rightsquigarrow_r (lemma 25 in [HHR06]).

$\langle 5 \rangle 4.$ Q.E.D.

P : $\langle 5 \rangle 2$ and $\langle 5 \rangle 3$.

$\langle 4 \rangle 2.$ C : $i = 1$

$\langle 5 \rangle 1.$ $o' \in [d'_1]$

P : Definition (12.13) of μ_1 .

$\langle 5 \rangle 2.$ Choose $o \in [d_1]$ such that $o \rightsquigarrow_r o'$

P : $\langle 5 \rangle 1$, the assumption and definition (12.21) of \rightsquigarrow_l .

$\langle 5 \rangle 3.$ $o \in \mu_1[d_1]$

P : $\langle 5 \rangle 2$ and definition (12.13) of μ_1 .

$\langle 5 \rangle 4.$ Q.E.D.

P : $\langle 5 \rangle 2$ and $\langle 5 \rangle 3$.

$\langle 4 \rangle 3.$ C : $i > 1$

$\langle 5 \rangle 1.$ A : $\forall o' \in \mu_k[d'_1]: \exists o \in \mu_k[d_1]: o \rightsquigarrow_r o'$
(induction hypothesis)

P : $\forall o' \in \mu_{k+1}[d'_1]: \exists o \in \mu_{k+1}[d_1]: o \rightsquigarrow_r o'$

$\langle 6 \rangle 1.$ $\exists o \in \mu_{k+1}[d_1]: o \rightsquigarrow_r o'$ for arbitrary $o' = (p', n') \in \mu_{k+1}[d'_1]$

$\langle 7 \rangle 1.$ Choose $(p'_k, n'_k) \in \mu_k[d'_1]$ and $(p'_1, n'_1) \in [d'_1]$ such that $(p', n') = (p'_k, n'_k) \succsim (p'_1, n'_1)$

P : $\langle 6 \rangle 1$, definition (12.13) of μ_n and definition (12.6) of \succsim .

$\langle 7 \rangle 2.$ Choose $(p_k, n_k) \in \mu_k[d_1]$ such that

$$(p_k, n_k) \rightsquigarrow_r (p'_k, n'_k)$$

P : $\langle 7 \rangle 1$ and the induction hypothesis.

- $\langle 7 \rangle 3.$ Choose $(p_1, n_1) \in \llbracket d_1 \rrbracket$ such that $(p_1, n_1) \rightsquigarrow_r (p'_1, n'_1)$
 $P : \langle 7 \rangle 1$ and the main assumption.
- $\langle 7 \rangle 4.$ $o = (p, n) = (p_k, n_k) \succsim (p_1, n_1) \in \mu_{k+1} \llbracket d_1 \rrbracket$
 $P : \langle 7 \rangle 2, \langle 7 \rangle 3$, definition (12.13) of μ_n and definition (12.6) of \succsim .
- $\langle 7 \rangle 5.$ $o \rightsquigarrow_r o'$
 $P : \langle 7 \rangle 1, \langle 7 \rangle 2, \langle 7 \rangle 3, \langle 7 \rangle 4$ and monotonicity of \rightsquigarrow_r with respect to \succsim (lemma 30 in [HHR06]).
- $\langle 7 \rangle 6.$ Q.E.D.
- $P : \langle 7 \rangle 4$ and $\langle 7 \rangle 5.$
- $\langle 6 \rangle 2.$ Q.E.D.
- $P : \forall\text{-rule.}$
- $\langle 5 \rangle 2.$ Q.E.D.
- $\langle 4 \rangle 4.$ Q.E.D.
- $P : \text{The cases are exhaustive as } i \text{ must be a natural number.}$
- $\langle 3 \rangle 2.$ Q.E.D.
- $P : \forall\text{-rule.}$
- $\langle 2 \rangle 2.$ Q.E.D.
- $P : \langle 1 \rangle 3 \text{ and } \langle 2 \rangle 1.$
- $\langle 1 \rangle 5.$ $o = (p, n) = (\bigcup_{i \in I} p_i, \bigcup_{i \in I} n_i) \in \llbracket \text{loop } I [d_1] \rrbracket$
 $P : \langle 1 \rangle 4$, definition (12.14) of loop and definition (12.11) of \biguplus .
- $\langle 1 \rangle 6.$ $(p, n) \rightsquigarrow_r (p', n')$
- $\langle 2 \rangle 1.$ Requirement 1: $n \subseteq n'$, i.e. $\bigcup_{i \in I} n_i \subseteq \bigcup_{i \in I} n'_i$
 $P : \forall i \in I : n_i \subseteq n'_i$ by $\langle 1 \rangle 4$ and definition (12.19) of \rightsquigarrow_r .
- $\langle 2 \rangle 2.$ Requirement 2: $p \subseteq p \cup n'$, i.e. $\bigcup_{i \in I} p_i \subseteq \bigcup_{i \in I} p'_i \cup \bigcup_{i \in I} n'_i$
 $P : \forall i \in I : p_i \subseteq p'_i \cup n'_i$ by $\langle 1 \rangle 4$ and definition (12.19) of \rightsquigarrow_r .
- $\langle 2 \rangle 3.$ Q.E.D.
- $P : \text{Definition (12.19) of } \rightsquigarrow_r.$
- $\langle 1 \rangle 7.$ $\forall o' \in \llbracket \text{loop } I [d'_1] \rrbracket : \exists o \in \llbracket \text{loop } I [d_1] \rrbracket : o \rightsquigarrow_r o'$
 $P : \langle 1 \rangle 2, \langle 1 \rangle 5, \langle 1 \rangle 6$ and $\forall\text{-rule.}$
- $\langle 1 \rangle 8.$ Q.E.D.
- $P : \langle 1 \rangle 1, \langle 1 \rangle 7$ and definition (12.21) of \rightsquigarrow_l .

□

Theorem 10 Monotonicity of \rightsquigarrow_l with respect to the seq operator

- $A : \forall i \in [1, m] : d_i \rightsquigarrow_l d'_i$
- $P : \text{seq } [d_1, \dots, d_m] \rightsquigarrow_l \text{seq } [d'_1, \dots, d'_m]$
- $\langle 1 \rangle 1.$ $\text{seq } [d_1, \dots, d_m] \rightsquigarrow_g \text{seq } [d'_1, \dots, d'_m]$
 $P : \text{The assumption and monotonicity of } \rightsquigarrow_g \text{ with respect to seq (theorem 13 in [HHR06])}.$
- $\langle 1 \rangle 2.$ Choose arbitrary $o' = (p', n') \in \llbracket \text{seq } [d'_1, \dots, d'_m] \rrbracket$
 $P : \llbracket \text{seq } [d'_1, \dots, d'_m] \rrbracket \text{ is non-empty by lemma 3.}$
- $\langle 1 \rangle 3.$ $\exists o \in \llbracket \text{seq } [d_1, \dots, d_m] \rrbracket : o \rightsquigarrow_r o'$
 $P : \text{By induction on } m, \text{ the number of seq-operands.}$

$\langle 2 \rangle 1.$ Base case: $m = 1$

$\langle 3 \rangle 1.$ $o' \in [\![d'_1]\!]$

P : $\langle 1 \rangle 2$ and definition (12.7) of seq.

$\langle 3 \rangle 2.$ Choose $o \in [\![d_1]\!]$ such that $o \rightsquigarrow_r o'$

P : $\langle 3 \rangle 1$, the assumption and definition (12.21) of \rightsquigarrow_l .

$\langle 3 \rangle 3.$ $o \in [\![\text{seq} [d_1]]\!]$

P : $\langle 3 \rangle 2$ and definition (12.7) of seq.

$\langle 3 \rangle 4.$ Q.E.D.

P : $\langle 3 \rangle 2$ and $\langle 3 \rangle 3$.

$\langle 2 \rangle 2.$ Induction step: $m = k + 1$

A : $\forall o' \in [\![\text{seq} [d'_1, \dots, d'_k]]\!] : \exists o \in [\![\text{seq} [d_1, \dots, d_k]]\!] : o \rightsquigarrow_r o'$ (induction hypothesis)

P : $\forall o' \in [\![\text{seq} [d'_1, \dots, d'_{k+1}]]\!] :$

$\exists o \in [\![\text{seq} [d_1, \dots, d_{k+1}]]\!] : o \rightsquigarrow_r o'$

$\langle 3 \rangle 1.$ Choose arbitrary $o' = (p', n') \in [\![\text{seq} [d'_1, \dots, d'_{k+1}]]\!]$

P : $[\![\text{seq} [d'_1, \dots, d'_{k+1}]]\!]$ is non-empty by lemma 3.

$\langle 3 \rangle 2.$ Choose $(p'_{1k}, n'_{1k}) \in [\![\text{seq} [d'_1, \dots, d'_k]]\!]$ and

$(p'_{k+1}, n'_{k+1}) \in [\![d'_{k+1}]\!]$ such that $(p', n') = (p'_{1k}, n'_{1k}) \succsim (p'_{k+1}, n'_{k+1})$

P : $\langle 3 \rangle 1$ and definitions (12.6)–(12.7) of seq.

$\langle 3 \rangle 3.$ Choose $(p_{1k}, n_{1k}) \in [\![\text{seq} [d_1, \dots, d_k]]\!]$ such that

$(p_{1k}, n_{1k}) \rightsquigarrow_r (p'_{1k}, n'_{1k})$

P : $\langle 3 \rangle 2$ and the induction hypothesis.

$\langle 3 \rangle 4.$ Choose $(p_{k+1}, n_{k+1}) \in [\![d_{k+1}]\!]$ such that

$(p_{k+1}, n_{k+1}) \rightsquigarrow_r (p'_{k+1}, n'_{k+1})$

P : $\langle 3 \rangle 2$, the main assumption and definition (12.21) of \rightsquigarrow_l .

$\langle 3 \rangle 5.$ $o = (p, n) = (p_{1k}, n_{1k}) \succsim (p_{k+1}, n_{k+1}) \in [\![\text{seq} [d_1, \dots, d_{k+1}]]\!]$

P : $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, and definitions (12.6)–(12.7) of seq.

$\langle 3 \rangle 6.$ $o \rightsquigarrow_r o'$

P : $\langle 3 \rangle 1$ – $\langle 3 \rangle 5$ and monotonicity of \rightsquigarrow_r with respect to \succsim (lemma 30 in [HHR06]).

$\langle 3 \rangle 7.$ Q.E.D.

P : $\langle 3 \rangle 5$ and $\langle 3 \rangle 6$.

$\langle 2 \rangle 3.$ Q.E.D.

P : The cases are exhaustive as m is a natural number.

$\langle 1 \rangle 4.$ $\forall o' \in [\![d']\!] : \exists o \in [\![d]\!] : o \rightsquigarrow_r o'$

P : $\langle 1 \rangle 2$, $\langle 1 \rangle 3$ and \forall -rule.

$\langle 1 \rangle 5.$ Q.E.D.

P : $\langle 1 \rangle 1$, $\langle 1 \rangle 4$ and definition (12.21) of \rightsquigarrow_l .

□

Theorem 11 *Monotonicity of \rightsquigarrow_l with respect to the alt operator*

A : $\forall i \in [1, m] : d_i \rightsquigarrow_l d'_i$

P : $\text{alt } [d_1, \dots, d_m] \rightsquigarrow_l \text{alt } [d'_1, \dots, d'_m]$

$\langle 1 \rangle 1.$ $\text{alt } [d_1, \dots, d_m] \rightsquigarrow_g \text{alt } [d'_1, \dots, d'_m]$

P : The assumption and monotonicity of \rightsquigarrow_g with respect to alt (theorem 11 in [HHR06]).

$\langle 1 \rangle 2.$ Choose arbitrary $o' = (p', n') \in [\text{alt } [d'_1, \dots, d'_m]]$

P : $[\text{alt } [d'_1, \dots, d'_m]]$ is non-empty by lemma 3.

$\langle 1 \rangle 3.$ For all $i \in [1, m]$, choose $(p'_i, n'_i) \in [d'_i]$ such that $p' = \bigcup_{i \in [1, m]} p'_i$ and $n' = \bigcup_{i \in [1, m]} n'_i$

P : $\langle 1 \rangle 2$, definition (12.10) of alt and definition (12.11) of \uplus .

$\langle 1 \rangle 4.$ For all $i \in [1, m]$, choose $(p_i, n_i) \in [d_i]$ such that $(p_i, n_i) \rightsquigarrow_r (p'_i, n'_i)$

P : $\langle 1 \rangle 3$, the assumption and definition (12.21) of \rightsquigarrow_l .

$\langle 1 \rangle 5.$ $o = (p, n) = (\bigcup_{i \in [1, m]} p_i, \bigcup_{i \in [1, m]} n_i) \in [\text{alt } [d_1, \dots, d_m]]$

P : $\langle 1 \rangle 4$, definition (12.10) of alt and definition (12.11) of \uplus .

$\langle 1 \rangle 6.$ $(p, n) \rightsquigarrow_r (p', n')$

$\langle 2 \rangle 1.$ Requirement 1: $n \subseteq n'$, i.e. $\bigcup_{i \in [1, m]} n_i \subseteq \bigcup_{i \in [1, m]} n'_i$

P : $\forall i \in [1, m] : n_i \subseteq n'_i$ by $\langle 1 \rangle 4$ and definition (12.19) of \rightsquigarrow_r .

$\langle 2 \rangle 2.$ Requirement 2: $p \subseteq p' \cup n'$, i.e. $\bigcup_{i \in [1, m]} p_i \subseteq \bigcup_{i \in [1, m]} p'_i \cup \bigcup_{i \in [1, m]} n'_i$

P : $\forall i \in [1, m] : p_i \subseteq p'_i \cup n'_i$ by $\langle 1 \rangle 4$ and definition (12.19) of \rightsquigarrow_r .

$\langle 2 \rangle 3.$ Q.E.D.

P : Definition (12.19) of \rightsquigarrow_r .

$\langle 1 \rangle 7.$ $\forall o' \in [\text{alt } [d'_1, \dots, d'_m]] : \exists o \in [\text{alt } [d_1, \dots, d_m]] : o \rightsquigarrow_r o'$

P : $\langle 1 \rangle 2, \langle 1 \rangle 5, \langle 1 \rangle 6$ and \forall -rule.

$\langle 1 \rangle 8.$ Q.E.D.

P : $\langle 1 \rangle 1, \langle 1 \rangle 7$ and definition (12.21) of \rightsquigarrow_l .

□

Theorem 12 *Monotonicity of \rightsquigarrow_l with respect to the xalt operator*

A : $\forall i \in [1, m] : d_i \rightsquigarrow_l d'_i$

P : $\text{xalt } [d_1, \dots, d_m] \rightsquigarrow_l \text{xalt } [d'_1, \dots, d'_m]$

$\langle 1 \rangle 1.$ $\text{xalt } [d_1, \dots, d_m] \rightsquigarrow_g \text{xalt } [d'_1, \dots, d'_m]$

P : The assumption and monotonicity of \rightsquigarrow_g with respect to xalt (theorem 12 in [HHR06]).

$\langle 1 \rangle 2.$ Choose arbitrary $o' = (p', n') \in [\text{xalt } [d'_1, \dots, d'_m]]$

P : $[\text{xalt } [d'_1, \dots, d'_m]]$ is non-empty by lemma 3.

$\langle 1 \rangle 3.$ Choose $i \in [1, m]$ such that $(p', n') \in [d'_i]$

P : $\langle 1 \rangle 2$ and definition (12.12) of xalt.

$\langle 1 \rangle 4.$ Choose $o = (p, n) \in [d_i]$ such that $(p, n) \rightsquigarrow_r (p', n')$

P : $\langle 1 \rangle 3$, the assumption and definition (12.21) of \rightsquigarrow_l .

$\langle 1 \rangle 5.$ $(p, n) \in [\text{xalt } [d_1, \dots, d_m]]$

P : $\langle 1 \rangle 4$ and definition (12.12) of xalt.

$\langle 1 \rangle 6.$ $\forall o' \in [\text{alt } [d'_1, \dots, d'_m]] : \exists o \in [\text{alt } [d_1, \dots, d_m]] : o \rightsquigarrow_r o'$

P : $\langle 1 \rangle 2, \langle 1 \rangle 4, \langle 1 \rangle 5$ and \forall -rule.
 $\langle 1 \rangle 7.$ Q.E.D.
P : $\langle 1 \rangle 1, \langle 1 \rangle 6$ and definition (12.21) of \rightsquigarrow_l .

□

12.E.2 Interactions with Data

Lemma 5 (*To be used when proving monotonicity with respect to guarded alt.*)

A : a. $\forall i \in [1, m] : c'_i \Rightarrow c_i$
b. $\forall i \in [1, m] : (p_i, n_i) \rightsquigarrow_r (p'_i, n'_i)$
L : a. $p = (\bigcup_{i \in [1, m]} p_i) \cup \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c_j)(\sigma)\}$
b. $n = \bigcup_{i \in [1, m]} n_i$
c. $p' = (\bigcup_{i \in [1, m]} p'_i) \cup \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c'_j)(\sigma)\}$
d. $n' = \bigcup_{i \in [1, m]} n'_i$
P : $(p, n) \rightsquigarrow_r (p', n')$

$\langle 1 \rangle 1.$ Requirement 1: $n \subseteq n'$, i.e. $\bigcup_{i \in [1, m]} n_i \subseteq \bigcup_{i \in [1, m]} n'_i$
P : $\forall i \in [1, m] : n_i \subseteq n'_i$ by assumption 2 and definition (12.19) of \rightsquigarrow_r .
 $\langle 1 \rangle 2.$ Requirement 2; $p \subseteq p' \cup n'$,
i.e. $(\bigcup_{i \in [1, m]} p_i) \cup \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c_j)(\sigma)\}$
 $\subseteq (\bigcup_{i \in [1, m]} p'_i) \cup \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c'_j)(\sigma)\} \cup \bigcup_{i \in [1, m]} n'_i$
 $\langle 2 \rangle 1.$ $\{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c_j)(\sigma)\} \subseteq \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c'_j)(\sigma)\}$
 $\langle 3 \rangle 1.$ $\forall j \in [1, m] : c'_j(\sigma) \Rightarrow c_j(\sigma)$
P : Assumption 1.
 $\langle 3 \rangle 2.$ $\forall j \in [1, m] : \neg c_j(\sigma) \Rightarrow \neg c'_j(\sigma)$
P : $\langle 3 \rangle 1$ and propositional logic ($a \Rightarrow b$ is equivalent to $\neg b \Rightarrow \neg a$).
 $\langle 3 \rangle 3.$ $\bigwedge_{j \in [1, m]} \neg c_j(\sigma) \Rightarrow \bigwedge_{j \in [1, m]} \neg c'_j(\sigma)$
P : $\langle 3 \rangle 2$ and propositional logic ($a_1 \Rightarrow b_1$ and $a_2 \Rightarrow b_2$ gives $a_1 \wedge a_2 \Rightarrow b_1 \wedge b_2$).
 $\langle 3 \rangle 4.$ Q.E.D.
P : $\langle 3 \rangle 3$ and elementary set theory ($a \Rightarrow a'$ gives $\{x \mid a\} \subseteq \{x \mid a'\}$).
 $\langle 2 \rangle 2.$ $\bigcup_{i \in [1, m]} p_i \subseteq \bigcup_{i \in [1, m]} p'_i \cup \bigcup_{i \in [1, m]} n'_i$
P : $\forall i \in [1, m] : p_i \subseteq p'_i \cup n'_i$ by assumption 2 and definition (12.19) of \rightsquigarrow_r .
 $\langle 2 \rangle 3.$ Q.E.D.
P : $\langle 2 \rangle 1$ and $\langle 2 \rangle 2$.
 $\langle 1 \rangle 3.$ Q.E.D.
P : Definition (12.19) of \rightsquigarrow_r .

□

Lemma 6 *Strengthening constraints*

$$(c' \Rightarrow c) \Rightarrow (\text{constr}(c) \rightsquigarrow_l \text{constr}(c'))$$

A : $c' \Rightarrow c$

P : $\text{constr}(c) \rightsquigarrow_l \text{constr}(c')$

P : $\text{constr}(c)$ and $\text{constr}(c')$ has only one interaction obligation each, where the different $\langle \text{check}(\sigma) \rangle$ -traces are positive or negative depending on the value of $c(\sigma)$ and $c'(\sigma)$, respectively. The trace $\langle \text{check}(\sigma) \rangle$ is negative for $\text{constr}(c)$ if $c(\sigma)$ is false. In this case, $c'(\sigma)$ is false as well (by the assumption), meaning that the trace is also negative for $\text{constr}(c')$ as required. The trace $\langle \text{check}(\sigma) \rangle$ is positive for $\text{constr}(c)$ if $c(\sigma)$ is true. As required, this trace is either positive or negative in $\text{constr}(c')$, depending on the value of $c'(\sigma)$.

$\langle 1 \rangle 1.$ $\llbracket \text{constr}(c) \rrbracket = \{(p, n)\}$

where $p = \{\langle \text{check}(\sigma) \rangle \mid c(\sigma)\}$ and $n = \{\langle \text{check}(\sigma) \rangle \mid \neg c(\sigma)\}$

P : Definition (12.16) of constr .

$\langle 1 \rangle 2.$ $\llbracket \text{constr}(c') \rrbracket = \{(p', n')\}$

where $p' = \{\langle \text{check}(\sigma) \rangle \mid c'(\sigma)\}$ and $n' = \{\langle \text{check}(\sigma) \rangle \mid \neg c'(\sigma)\}$

P : Definition (12.16) of constr .

$\langle 1 \rangle 3.$ $(p, n) \rightsquigarrow_r (p', n')$

$\langle 2 \rangle 1.$ Requirement 1: $n \subseteq n'$,

i.e. $\{\langle \text{check}(\sigma) \rangle \mid \neg c(\sigma)\} \subseteq \{\langle \text{check}(\sigma) \rangle \mid \neg c'(\sigma)\}$

P : The assumption gives $c'(\sigma) \Rightarrow c(\sigma)$, i.e. $\neg c(\sigma) \Rightarrow \neg c'(\sigma)$.

$\langle 2 \rangle 2.$ Requirement 2: $p \subseteq p' \cup n'$,

i.e. $\{\langle \text{check}(\sigma) \rangle \mid c(\sigma)\} \subseteq \{\langle \text{check}(\sigma) \rangle \mid c'(\sigma)\} \cup \{\langle \text{check}(\sigma) \rangle \mid \neg c'(\sigma)\}$

P : Trivial, as the right side equals $\{\langle \text{check}(\sigma) \rangle \mid \sigma \in \text{Var} \rightarrow \text{Val}\}$.

$\langle 2 \rangle 3.$ Q.E.D.

P : Definition (12.19) of \rightsquigarrow_r .

$\langle 1 \rangle 4.$ $\forall o \in \llbracket \text{constr}(c) \rrbracket : \exists o' \in \llbracket \text{constr}(c') \rrbracket : o \rightsquigarrow_r o'$

P : $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ and $\langle 1 \rangle 3$.

$\langle 1 \rangle 5.$ $\forall o' \in \llbracket \text{constr}(c') \rrbracket : \exists o \in \llbracket \text{constr}(c) \rrbracket : o \rightsquigarrow_r o'$

P : $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ and $\langle 1 \rangle 3$.

$\langle 1 \rangle 6.$ Q.E.D.

P : $\langle 1 \rangle 4$, $\langle 1 \rangle 5$ and definitions (12.20)–(12.21) of \rightsquigarrow_l .

□

Lemma 7 $(c' \Rightarrow c \wedge d \rightsquigarrow_g d') \Rightarrow (\text{seq}[\text{constr}(c), d] \rightsquigarrow_g \text{seq}[\text{constr}(c'), d'])$

A : a. $c' \Rightarrow c$

b. $d \rightsquigarrow_g d'$

P : $\text{seq}[\text{constr}(c), d] \rightsquigarrow_g \text{seq}[\text{constr}(c'), d']$

$\langle 1 \rangle 1.$ $\text{constr}(c) \rightsquigarrow_g \text{constr}(c')$

P : Assumption 1, lemma 6 and definition (12.21) of \rightsquigarrow_l .

$\langle 1 \rangle 2.$ $d \rightsquigarrow_g d'$

P : Assumption 2.

$\langle 1 \rangle 3.$ Q.E.D.

P : By monotonicity of \rightsquigarrow_g with respect to the seq operator (theorem 13 in [HHR06]).

□

Lemma 8 $(c' \Rightarrow c \wedge d \rightsquigarrow_l d') \Rightarrow (\text{seq}[\text{constr}(c), d] \rightsquigarrow_l \text{seq}[\text{constr}(c'), d'])$

A : a. $c' \Rightarrow c$

b. $d \rightsquigarrow_l d'$

P : $\text{seq}[\text{constr}(c), d] \rightsquigarrow_l \text{seq}[\text{constr}(c'), d']$

$\langle 1 \rangle 1.$ $\text{constr}(c) \rightsquigarrow_l \text{constr}(c')$

P : Assumption 1 and lemma 6.

$\langle 1 \rangle 2.$ $d \rightsquigarrow_l d'$

P : Assumption 2.

$\langle 1 \rangle 3.$ Q.E.D.

P : Theorem 10 (monotonicity of \rightsquigarrow_l with respect to the seq operator).

□

Theorem 13 *Monotonicity of \rightsquigarrow_g with respect to the guarded alt operator*

For guarded alt,

(i) the operands may be refined separately, and

(ii) constraining the guards is a valid refinement step.

A : a. $\forall i \in [1, m] : c'_i \Rightarrow c_i$

b. $\forall i \in [1, m] : d_i \rightsquigarrow_g d'_i$

P : $\text{alt}[c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m] \rightsquigarrow_g \text{alt}[c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]$

P : Each obligation $o \in \llbracket \text{alt}[c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m] \rrbracket$ is constructed as the inner union of a set of obligations $o_i \in \llbracket \text{seq}[\text{constr}(c_i), d_i] \rrbracket$ (for $i \in [1, m]$), together with the extra no-guard-true obligation. By the assumptions and lemma 7, we may for each o_i select a refining obligation $o'_i \in \llbracket \text{seq}[\text{constr}(c'_i), d'_i] \rrbracket$. Using these o'_i 's and the extra no-guard-true obligation, we then construct an obligation $o' \in \llbracket \text{alt}[c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m] \rrbracket$. Each negative trace in o is negative in one of the o_i 's. By definition (12.19) of refinement it is also negative in the corresponding o'_i and therefore negative in o' as required. Similarly, each positive trace in o is either positive in the no-guard-true obligation and therefore positive also in o' , or positive in one of the o_i 's meaning that by definition (12.19) it is positive or negative in the corresponding o'_i , and therefore positive or negative in o' as required.

$\langle 1 \rangle 1.$ Choose arbitrary $o = (p, n) \in \llbracket \text{alt}[c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m] \rrbracket$

P : $\llbracket \text{alt}[c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m] \rrbracket$ is non-empty by lemma 3.

$\langle 1 \rangle 2.$ For all $i \in [1, m]$, choose $(p_i, n_i) \in \llbracket \text{seq}[\text{constr}(c_i), d_i] \rrbracket$ such that

$p = (\bigcup_{i \in [1, m]} p_i) \cup \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c_j)(\sigma)\}$ and $n = \bigcup_{i \in [1, m]} n_i$

P : $\langle 1 \rangle 1$, definition (12.17) of guarded alt and definition (12.11) of \biguplus .

- $\langle 1 \rangle 3.$ For all $i \in [1, m]$, choose $(p'_i, n'_i) \in [\text{seq}[\text{constr}(c'_i), d'_i]]$ such that $(p_i, n_i) \rightsquigarrow_r (p'_i, n'_i)$
- P : $\langle 1 \rangle 2$, the assumptions, lemma 7 and definition (12.20) of \rightsquigarrow_g .
- $\langle 1 \rangle 4.$ $o' = (p', n') = ((\bigcup_{i \in [1, m]} p'_i) \cup \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c'_j)(\sigma)\}, \bigcup_{i \in [1, m]} n'_i)$
 $\in [\text{alt}[c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]]$
- P : $\langle 1 \rangle 3$, definition (12.17) of guarded alt and definition (12.11) of \bigcup .
- $\langle 1 \rangle 5.$ $(p, n) \rightsquigarrow_r (p', n')$
- P : Assumption 1, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $\langle 1 \rangle 4$ and lemma 5.
- $\langle 1 \rangle 6.$ $\forall o \in [\text{alt}[c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]] :$
 $\exists o' \in [\text{alt}[c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]] : o \rightsquigarrow_r o'$
- P : $\langle 1 \rangle 1$, $\langle 1 \rangle 4$, $\langle 1 \rangle 5$ and \forall -rule.
- $\langle 1 \rangle 7.$ Q.E.D.
- P : $\langle 1 \rangle 6$ and definition (12.20) of \rightsquigarrow_g .

□

Theorem 14 *Monotonicity of \rightsquigarrow_g with respect to the guarded xalt operator*

For guarded xalt,

- (i) the operands may be refined separately, and
- (ii) constraining the guards is a valid refinement step.

- A : a. $\forall i \in [1, m] : c'_i \Rightarrow c_i$
b. $\forall i \in [1, m] : d_i \rightsquigarrow_g d'_i$
- P : $\text{xalt}[c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m] \rightsquigarrow_g \text{xalt}[c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]$
- P : For each obligation $o \in [\text{xalt}[c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]]$, there exists an i such that $o \in [\text{seq}[\text{constr}(c_i), d_i]]$. By the assumptions and lemma 7, we may select an obligation $o' \in [\text{seq}[\text{constr}(c'_i), d'_i]]$ such that $o \rightsquigarrow_r o'$. By definition (12.18) of guarded xalt, o' is also an obligation in $[\text{xalt}[c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]]$.

- $\langle 1 \rangle 1.$ Choose arbitrary $o = (p, n) \in [\text{xalt}[c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]]$
- P : $[\text{xalt}[c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]]$ is non-empty by lemma 3.
- $\langle 1 \rangle 2.$ Choose $i \in [1, m]$ such that $(p, n) \in [\text{seq}[\text{constr}(c_i), d_i]]$
- P : $\langle 1 \rangle 1$ and definition (12.17) of guarded xalt.
- $\langle 1 \rangle 3.$ Choose $o' = (p', n') \in [\text{seq}[\text{constr}(c'_i), d'_i]]$ such that $(p, n) \rightsquigarrow_r (p', n')$
- P : $\langle 1 \rangle 2$, the assumptions, lemma 7 and definition (12.20) of \rightsquigarrow_g .
- $\langle 1 \rangle 4.$ $(p', n') \in [\text{xalt}[c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]]$
- P : $\langle 1 \rangle 3$ and definition (12.18) of guarded xalt.
- $\langle 1 \rangle 5.$ $\forall o \in [\text{alt}[c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]] :$
 $\exists o' \in [\text{alt}[c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]] : o \rightsquigarrow_r o'$
- P : $\langle 1 \rangle 1$, $\langle 1 \rangle 3$, $\langle 1 \rangle 4$ and \forall -rule.
- $\langle 1 \rangle 6.$ Q.E.D.
- P : $\langle 1 \rangle 5$ and definition (12.20) of \rightsquigarrow_g .

□

Theorem 15 *Monotonicity of \rightsquigarrow_l with respect to the guarded alt operator*

- A : a. $\forall i \in [1, m] : c'_i \Rightarrow c_i$
- b. $\forall i \in [1, m] : d_i \rightsquigarrow_l d'_i$
- P : $\text{alt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m] \rightsquigarrow_l \text{alt } [c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]$
- $\langle 1 \rangle 1.$ $\text{alt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m] \rightsquigarrow_g \text{alt } [c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]$
 P : The assumptions and theorem 13 (monotonicity of \rightsquigarrow_g with respect to guarded alt).
- $\langle 1 \rangle 2.$ Choose arbitrary $o' = (p', n') \in [\text{alt } [c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]]$
 P : $[\text{alt } [c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]]$ is non-empty by lemma 3.
- $\langle 1 \rangle 3.$ For all $i \in [1, m]$, choose $(p'_i, n'_i) \in [\text{seq } [\text{constr}(c'_i), d'_i]]$ such that
 $p' = (\bigcup_{i \in [1, m]} p'_i) \cup \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c'_j)(\sigma)\}$ and $n' = \bigcup_{i \in [1, m]} n'_i$
 P : $\langle 1 \rangle 2$, definition (12.17) of guarded alt and definition (12.11) of \biguplus .
- $\langle 1 \rangle 4.$ For all $i \in [1, m]$, choose $(p_i, n_i) \in [\text{seq } [\text{constr}(c_i), d_i]]$ such that $(p_i, n_i) \rightsquigarrow_r (p'_i, n'_i)$
 P : $\langle 1 \rangle 3$, the assumptions, lemma 8 and definition (12.21) of \rightsquigarrow_l .
- $\langle 1 \rangle 5.$ $o = (p, n) = ((\bigcup_{i \in [1, m]} p_i) \cup \{\langle \text{check}(\sigma) \rangle \mid (\bigwedge_{j \in [1, m]} \neg c_j)(\sigma)\}, \bigcup_{i \in [1, m]} n_i)$
 $\in [\text{alt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]]$
 P : $\langle 1 \rangle 4$, definition (12.17) of guarded alt and definition (12.11) of \biguplus .
- $\langle 1 \rangle 6.$ $(p, n) \rightsquigarrow_r (p', n')$
 P : Assumption 1, $\langle 1 \rangle 3$, $\langle 1 \rangle 4$, $\langle 1 \rangle 5$ and lemma 5.
- $\langle 1 \rangle 7.$ $\forall o' \in [\text{alt } [c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]] :$
 $\exists o \in [\text{alt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]] : o \rightsquigarrow_r o'$
 P : $\langle 1 \rangle 2$, $\langle 1 \rangle 5$, $\langle 1 \rangle 6$ and \forall -rule.
- $\langle 1 \rangle 8.$ Q.E.D.
 P : $\langle 1 \rangle 1$, $\langle 1 \rangle 7$ and definition (12.21) of \rightsquigarrow_l .

□

Theorem 16 *Monotonicity of \rightsquigarrow_l with respect to the guarded xalt operator*

- A : a. $\forall i \in [1, m] : c'_i \Rightarrow c_i$
- b. $\forall i \in [1, m] : d_i \rightsquigarrow_l d'_i$
- P : $\text{xalt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m] \rightsquigarrow_l \text{xalt } [c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]$
- $\langle 1 \rangle 1.$ $\text{xalt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m] \rightsquigarrow_g \text{xalt } [c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]$
 P : The assumptions and theorem 14 (monotonicity of \rightsquigarrow_g with respect to guarded xalt).
- $\langle 1 \rangle 2.$ Choose arbitrary $o' = (p', n') \in [\text{xalt } [c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]]$
 P : $[\text{xalt } [c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]]$ is non-empty by lemma 3.
- $\langle 1 \rangle 3.$ Choose $i \in [1, m]$ such that $(p', n') \in [\text{seq } [\text{constr}(c'_i), d'_i]]$
 P : $\langle 1 \rangle 2$ and definition (12.18) of guarded xalt.
- $\langle 1 \rangle 4.$ Choose $o = (p, n) \in [\text{seq } [\text{constr}(c_i), d_i]]$ such that $(p, n) \rightsquigarrow_r (p', n')$
 P : $\langle 1 \rangle 3$, the assumptions, lemma 8 and definition (12.21) of \rightsquigarrow_l .
- $\langle 1 \rangle 5.$ $(p, n) \in [\text{xalt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]]$
 P : $\langle 1 \rangle 4$ and definition (12.18) of guarded xalt.
- $\langle 1 \rangle 6.$ $\forall o' \in [\text{alt } [c'_1 \rightarrow d'_1, \dots, c'_m \rightarrow d'_m]] :$
 $\exists o \in [\text{alt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m]] : o \rightsquigarrow_r o'$

P : $\langle 1 \rangle 2, \langle 1 \rangle 4, \langle 1 \rangle 5$ and \forall -rule.
 $\langle 1 \rangle 7.$ Q.E.D.
P : $\langle 1 \rangle 1, \langle 1 \rangle 6$ and definition (12.21) of \rightsquigarrow_l .

□

Chapter 13

The Pragmatics of STAIRS

Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen

Publication.

Published in *Proc. Formal Methods for Components and Objects (FMCO 2005)*, volume 4111 of *LNCS*, pages 88–114. Springer, 2006.

Abstract.

STAIRS is a method for the compositional development of interactions in the setting of UML 2.0. In addition to defining denotational trace semantics for the main aspects of interactions, STAIRS focuses on how interactions may be developed through successive refinement steps. In this tutorial paper, we concentrate on explaining the practical relevance of STAIRS. Guidelines are given on how to create interactions using the different STAIRS operators, and how these may be refined. The pragmatics is illustrated by a running example.

Note.

Minor error corrections have been made with respect to the original paper.

13.1 Introduction

STAIRS [HHR05a] is a method for the compositional development of interactions in the setting of UML 2.0 [OMG05]. In contrast to e.g. UML state machines and Java programs, interactions are usually incomplete specifications, typically describing example runs of the system. STAIRS is designed to deal with this incompleteness. Another important feature of STAIRS is the possibility to distinguish between alternative behaviours representing underspecification and alternative behaviours that must all be present in an implementation, for instance due to inherent nondeterminism.

STAIRS is not intended to be a complete methodology for system development, but should rather be seen as a supplement to methodologies like e.g. RUP [Kru04]. In particular, STAIRS focuses on refinement, which is a development step where the specification is made more complete by information being added to it in such a way that any valid implementation of the refined specification will also be a valid implementation of the original specification.

In this paper we focus on refinement relations. We define general refinement, which in turn has four special cases referred to as narrowing, supplementing, detailing and limited refinement. Narrowing means to reduce the set of possible system behaviours, thus reducing underspecification. Supplementing, on the other hand, means to add new behaviours to the specification, taking into account the incomplete nature of interactions, while detailing means to add more details to the specification by decomposing lifelines. By general refinement, the nondeterminism required of an implementation may be increased freely, while limited refinement is a special case restricting this possibility.

Previous work on STAIRS has focused on its basic ideas, explaining the various concepts such as the distinction between underspecification and inherent nondeterminism [HHR05a, RHS05b], time [HHR05b], and negation [RHS05a], as well as how these are formalized. In this paper, we take the theory of STAIRS one step further, focusing on its practical consequences by giving practical guidelines on how to use STAIRS. In particular, we explain how to use the various STAIRS operators when creating specifications in the form of interactions, and how these specifications may be further developed through valid refinement steps.

The paper is organized as follows: In Sect. 13.2 we give a brief introduction to interactions and their semantic model as we have defined it in STAIRS. Section 13.3 is an exampleguided walkthrough of the main STAIRS operators for creating interactions, particularly focusing on alternatives and negation. For each operator we give both its formal definition and guidelines for its practical usage. Section 13.4 gives the pragmatics of refining interactions. In Sect. 13.5 we explain how STAIRS relates to other similar approaches, in particular UML 2.0, while we give some concluding remarks in Sect. 13.6.

13.2 The Semantic Model of STAIRS

In this section we give a brief introduction to interactions and their trace semantics as defined in STAIRS. The focus here is on the semantic model. Definitions of concrete syntactical operators will mainly be presented together with the discussion of these operators later in this paper. For a thorough account of the STAIRS semantics, see [HHR05b] and the extension with data in [RHS05b].

An interaction describes one or more positive (i.e. valid) and/or negative (i.e. invalid) behaviours. As a very simple example, the sequence diagram in Fig. 13.1 specifies a scenario in which a client sends the message `cancel(appointment)` to an appointment system, which subsequently sends the message `appointmentCancelled` back to the client, together with a suggestion for a new appointment to which the client answers with the message `yes`. The client finally receives the message `appointmentMade`.

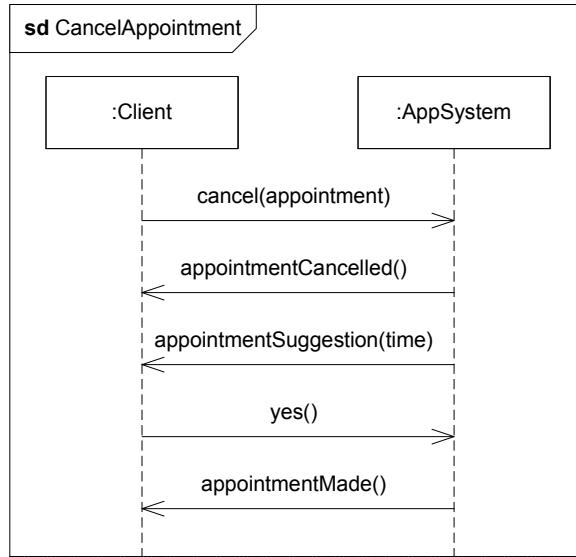


Figure 13.1: Example interaction: CancelAppointment

Formally, we use denotational trace semantics to explain the meaning of a single interaction. A trace is a sequence of events, representing a system run. The most typical examples of events are the sending and the reception of a message, where a message is a triple (s, tr, re) consisting of a signal s , a transmitter lifeline tr and a receiver lifeline re . For a message m , we let $!m$ and $?m$ denote the sending and the reception of m , respectively. As will be explained in Sect. 13.3.2, we also have some special events representing the use of data in e.g. constraints and guards.

The diagram in Fig. 13.1 includes ten events, two for each message. These are combined by the implicit weak sequencing operator `seq`, which will be formally defined at the end of this section. Informally, the set of traces described by such a diagram is the set of all possible sequences consisting of its events such that the send event is ordered before the corresponding receive event, and events on the same lifeline are ordered from top down. Shortening each message to the first and the capitalized letter of its signal, we thus get that Fig. 13.1 specifies two positive traces $\langle !c, ?c, !aC, ?aC, !aS, ?aS, !y, ?y, !aM, ?aM \rangle$ and $\langle !c, ?c, !aC, !aS, ?aC, ?aS, !y, ?y, !aM, ?aM \rangle$, where the only difference is the relative ordering of the two events `?aC` and `!aS`. Figure 13.1 gives no negative traces.

Formally, we let \mathcal{H} denote the set of all well-formed traces. A trace is well-formed if, for each message, the send event is ordered before the corresponding receive event. An *interaction obligation* (p, n) is a pair of trace-sets which gives a classification of all of the traces in \mathcal{H} into three categories: the positive traces p , the negative traces n and the inconclusive traces $\mathcal{H} \setminus (p \cup n)$. The inconclusive traces result from the incompleteness of interactions, representing traces that

are not described as positive or negative by the current interaction. We say that the interaction obligation is contradictory if the same trace is both positive and negative, i.e. if $p \cap n \neq \emptyset$. To give a visual presentation of an interaction obligation, we use an oval divided into three regions as shown in Fig. 13.2.

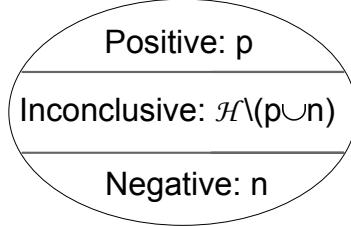


Figure 13.2: Illustrating an interaction obligation

As explained in the introduction, one of the main advantages of STAIRS is its ability to distinguish between traces that an implementation *may* exhibit (e.g. due to underspecification), and traces that it *must* exhibit (e.g. due to inherent nondeterminism). Semantically, this distinction is captured by stating that the semantics of an interaction d is a set of m interaction obligations, $\llbracket d \rrbracket = \{(p_1, n_1), \dots, (p_m, n_m)\}$. Intuitively, the traces allowed by an interaction obligation (i.e. its positive and inconclusive traces) represent potential alternatives, where being able to produce only one of these traces is sufficient for an implementation. On the other hand, the different interaction obligations represent mandatory alternatives, in the sense that each obligation specifies traces of which at least one must be possible for a correct implementation of the specification.

We are now ready to give the formal definition of seq. First, weak sequencing of trace sets is defined by:

$$s_1 \succsim s_2 \stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \forall l \in \mathcal{L} : h \upharpoonright l = h_1 \upharpoonright l \circ h_2 \upharpoonright l\} \quad (13.1)$$

where \mathcal{L} is the set of all lifelines, \circ is the concatenation operator on sequences, and $h \upharpoonright l$ is the trace h with all events not taking place on the lifeline l removed. Basically, this definition gives all traces that may be constructed by selecting one trace from each operand and combining them in such a way that the events from the first operand must come before the events from the second operand (i.e. the events are ordered from top down) for all lifelines. Events from different operands may come in any order, as long as sending comes before reception for each message (as required by $h \in \mathcal{H}$). Notice that weak sequencing with an empty set as one of the operands yields the empty set.

Weak sequencing of interaction obligations is defined by:

$$(p_1, n_1) \succsim (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \succsim p_2, (n_1 \succsim p_2) \cup (n_1 \succsim n_2) \cup (p_1 \succsim n_2)) \quad (13.2)$$

Finally, seq is defined by

$$\begin{aligned} \llbracket \text{seq } [d] \rrbracket &\stackrel{\text{def}}{=} \llbracket d \rrbracket \\ \llbracket \text{seq } [D, d] \rrbracket &\stackrel{\text{def}}{=} \{o_1 \succsim o_2 \mid o_1 \in \llbracket \text{seq } [D] \rrbracket \wedge o_2 \in \llbracket d \rrbracket\} \end{aligned} \quad (13.3)$$

where d is a single interaction and D a list of one or more interactions. For a further discussion of seq, see Sect. 13.3.4.

13.3 The Pragmatics of Creating Interactions

In this section, we focus on the different syntactical constructs of interactions in order to explain the main theory of STAIRS and how these constructs should be used in practical development. For each construct, we demonstrate its usage in a practical example, to motivate its formal semantics and the pragmatic rules and guidelines that conclude each subsection.

As our example, we will use a system for appointment booking to be used by e.g. doctors and dentists. The appointment system should have the following functionality:

- MakeAppointment: The client may ask for an appointment.
- CancelAppointment: The client may cancel an appointment.
- Payment: The system may send an invoice message asking the client to pay for the previous or an unused appointment.

The interactions specifying this system will be developed in a stepwise manner. In Sect. 13.4 we will justify that all of these development steps are valid refinement steps in STAIRS.

13.3.1 The Use of alt Versus xalt

Consider again the interaction in Fig. 13.1. As explained in Sect. 13.2, this interaction specifies two different traces, depending on whether the client receives the message appointmentCancelled before or after the system sends the message appointmentSuggestion. Which one of these we actually get when running the final system, will typically depend on the kind of communication used between the client and our system. If the communication is performed via SMS or over the internet, we may have little or no delay, meaning that the first of these messages may be received before the second is sent. On the other hand, if communication is performed by sending letters via ordinary mail, both messages (i.e. letters) will probably be sent before the first one arrives at the client.

Seeing that the means of communication is not specified in the interaction, all of these are acceptable implementations. Also, it is sufficient for an implementation to have only one of these available. Hence, the two traces of Fig. 13.1 exemplify underspecification. In the semantics, this is represented by the two traces being grouped together in the same interaction obligation.

The underspecification in Fig. 13.1 is an implicit consequence of weak sequencing. Alternatives representing underspecification may also be specified explicitly by using the operator alt, as in the specification of MakeAppointment in Fig. 13.3. In this interaction, when requesting a new appointment the client may ask for either a specific date or a specific hour of the day (for instance if the client prefers his appointments to be after work or during the lunch break). As we do not require the system to offer both of these alternatives, they are specified using alt. After the client has asked for an appointment, the appointment is set up according to the referenced interaction DecideAppTime.

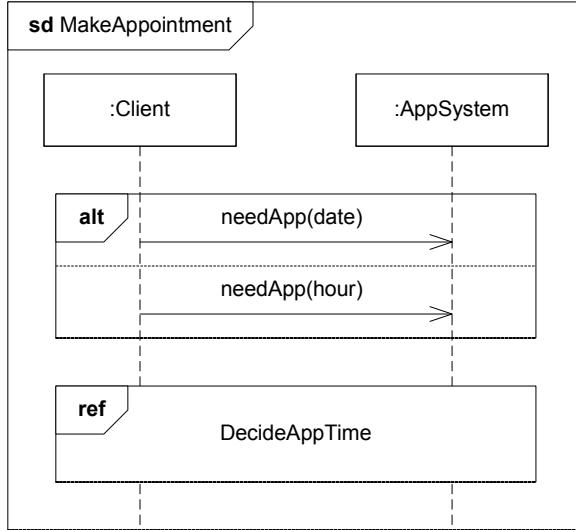


Figure 13.3: MakeAppointment

The specification of `DecideAppTime` is given in Fig. 13.4. Here, the system starts with suggesting an appointment, and the client then answers either yes or no. Finally, the client gets a receipt according to his answer. As the system must be able to handle both yes and no as reply messages, these alternatives are *not* instances of underspecification. Specifying these by using `alt` would therefore be insufficient. Instead, they are specified by the `xalt` operator (first introduced in [HS03]) in order to capture alternative traces where an implementation must be able to perform all alternatives.

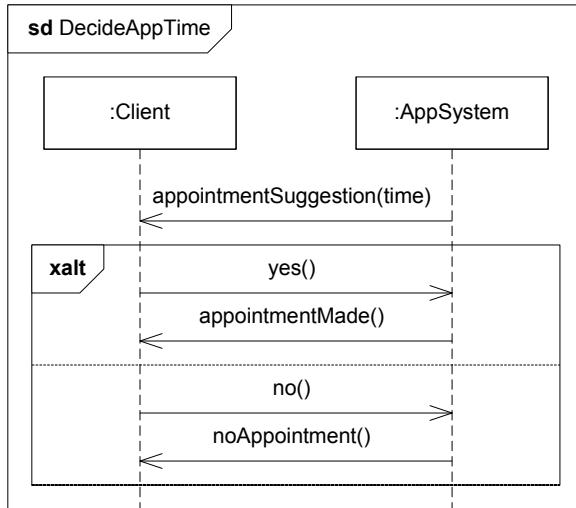


Figure 13.4: DecideAppTime

In Fig. 13.5, which gives a revised version of `CancelAppointment` from Fig. 13.1, another use of `xalt` is demonstrated. In this case, `xalt` is used to model alternatives where the conditions under

which each of them may be chosen is not known. This interaction specifies that if a client tries to cancel an appointment, he may either get an error message or he may get a confirmation of the cancellation, after which the system tries to schedule another appointment (in `DecideAppTime`). In Sect. 13.3.2 we demonstrate how guards may be added as a means to constrain the applicability of the two alternatives in this example.

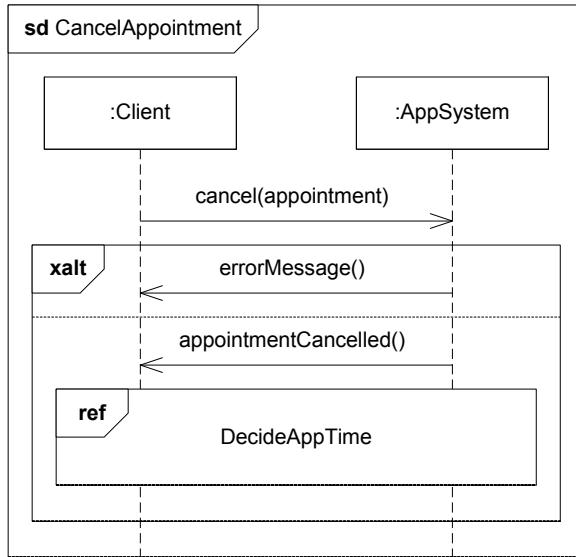


Figure 13.5: `CancelAppointment` revisited

A third use of `xalt` is to specify inherent nondeterminism, as in a coin toss where both heads and tails should be possible outcomes. More examples, and a discussion of the relationship between `alt` and `xalt`, may be found in [RHS05b] and [RRS06].

The crucial question when specifying alternatives is: Do these alternatives represent similar traces in the sense that implementing only one is sufficient? If yes, use `alt`. Otherwise, use `xalt`.

Formally, the operands of an `xalt` result in distinct interaction obligations in order to model the situation that they must all be possible for an implementation. On the other hand, `alt` combines interaction obligations in order to model underspecification:

$$[\![\text{xalt } [d_1, \dots, d_m]]\!] \stackrel{\text{def}}{=} \bigcup_{i \in [1, m]} [\![d_i]\!] \quad (13.4)$$

$$[\![\text{alt } [d_1, \dots, d_m]]\!] \stackrel{\text{def}}{=} \{ \biguplus \{o_1, \dots, o_m\} \mid \forall i \in [1, m] : o_i \in [\![d_i]\!] \} \quad (13.5)$$

where m is the number of interaction operands and the inner union of interaction obligations, \biguplus , is defined as:

$$\biguplus_{i \in [1, m]} (p_i, n_i) \stackrel{\text{def}}{=} \left(\bigcup_{i \in [1, m]} p_i, \bigcup_{i \in [1, m]} n_i \right) \quad (13.6)$$

The difference between `alt` and `xalt` is also illustrated in Fig. 13.6, which is an informal illustration of the semantics of Fig. 13.3. The dotted lines should be interpreted as parentheses

grouping the semantics of sub-interactions, and the second seq-operand is the semantics of the referenced interaction DecideAppTime. In each interaction obligation of Fig. 13.6, every trace that is not positive is inconclusive, as Fig. 13.3 gives no negative traces.

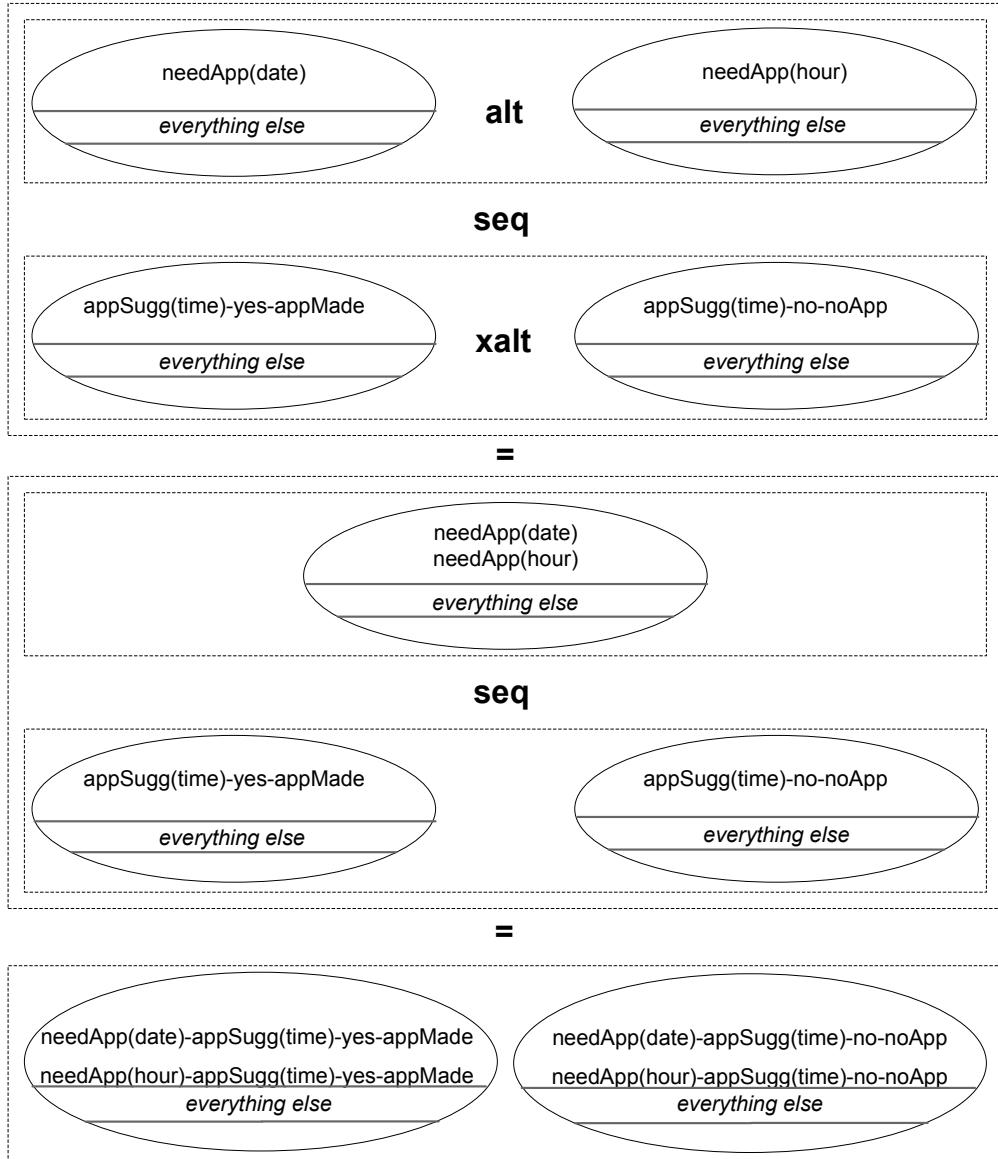


Figure 13.6: Illustrating MakeAppointment

Every interaction using the STAIRS-operators except for infinite loop is equivalent to an interaction having xalt as the top-level operator. This is because xalt describes mandatory alternatives. If there are only finitely many alternatives (which is the case if there is no infinite loop) they may be listed one by one. In particular, we have that all of these operators distribute over xalt. For instance, we have that the interaction alt [xalt [d_1, d_2], d_3] is equivalent to the interaction xalt [alt [d_1, d_3], alt [d_2, d_3]], and similarly for interactions with more than two operands and for the other operators.

The pragmatics of alt vs xalt

- Use alt to specify alternatives that represent similar traces, i.e. to model
 - underspecification.
- Use xalt to specify alternatives that must all be present in an implementation, i.e. to model
 - inherent nondeterminism, as in the specification of a coin toss.
 - alternative traces due to different inputs that the system must be able to handle (as in Fig. 13.4);
 - alternative traces where the conditions for these being positive are abstracted away (as in Fig. 13.5).

13.3.2 The Use of Guards

In Fig. 13.5, xalt was used in order to specify that the system should be able to respond with either an error message or with the receipt message appointmentCancelled (followed by DecideAppTime), if a client wants to cancel an appointment. With the current specification, the choice between these alternatives may be performed nondeterministically, but as suggested in the previous section, it is more likely that there exist some conditions for when each of the alternatives may be chosen. In Fig. 13.7 these conditions are made explicit by adding them to the specification in the form of guards as a step in the development process.

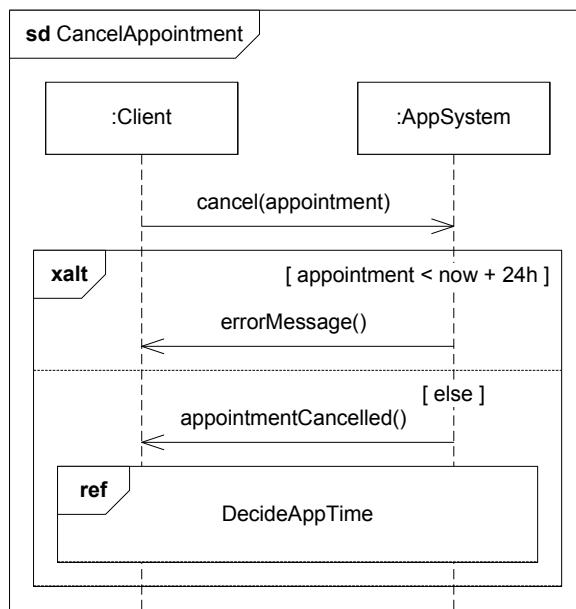


Figure 13.7: CancelAppointment revisited

For the first alternative, the guard is used to specify that if the client wants to cancel an appointment less than 24 hours in advance, he will get an error message. In general, the guard

`else` may be used as a short-hand for the conjunction of the negation of all the other guards. This means that for the second alternative of Fig. 13.7, the appointment will be cancelled and the system will try to schedule a new appointment only if the appointment is at least 24 hours away.

Similarly, in Fig. 13.8, guards are added to the alt-construct of Fig. 13.3 in order to constrain the situations in which each of the alternatives `needApp(date)` and `needApp(hour)` is positive. The guards specify that the client may only ask for an appointment at today or at a later date, or between the hours of 7 A.M. and 5 P.M. We recommend that one always makes sure that the guards of an alt-construct are exhaustive. Therefore, Fig. 13.8 also adds an alternative where the client asks for an appointment without specifying either date or hour. This alternative has the guard `true`, and may always be chosen. As this example demonstrates, the guards of an alt-construct may be overlapping. This is also the case for xalt.

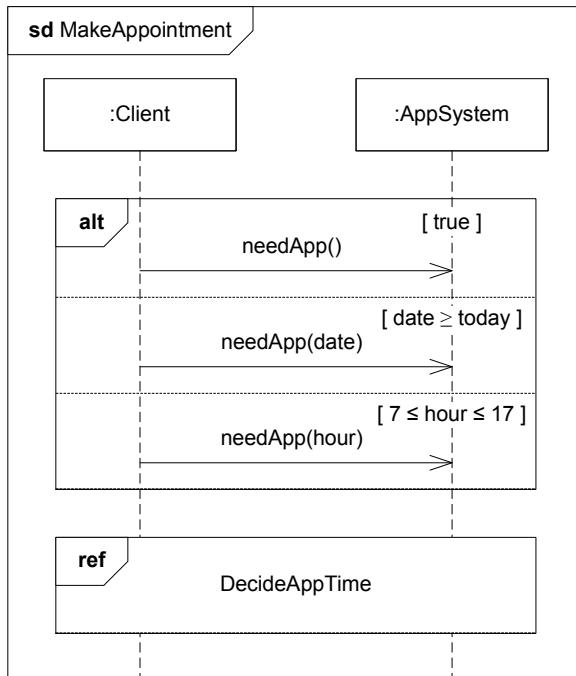


Figure 13.8: MakeAppointment revisited

In order to capture guards and more general constraints in the semantics, the semantics is extended with the notion of a state. A state σ is a total function assigning a value (in the set Val) to each variable (in the set Var). Formally, $\sigma \in Var \rightarrow Val$. Semantically, a constraint is represented by the special event $check(\sigma)$, where σ is the state in which the constraint is evaluated:

$$\llbracket \text{constr}(c) \rrbracket \stackrel{\text{def}}{=} \{ (\langle check(\sigma) \rangle \mid c(\sigma)), \langle check(\sigma) \rangle \mid \neg c(\sigma)) \} \quad (13.7)$$

The semantics of guarded xalt is defined by:

$$\llbracket \text{xalt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m] \rrbracket \stackrel{\text{def}}{=} \bigcup_{i \in [1, m]} \llbracket \text{seq } [\text{constr}(c_i), d_i] \rrbracket \quad (13.8)$$

Notice that for all states, a constraint is either true and the trace $\langle \text{check}(\sigma) \rangle$ is positive, or the constraint is false and the trace $\langle \text{check}(\sigma) \rangle$ is negative. For guarded xalt (and similarly for alt defined below), this has the consequence that a guard must cover *all* possible situations in which the specified traces are positive, since a false guard means that the traces described by this alternative are negative. When relating specifications with and without guards, an alt/xalt-operand without a guard is interpreted as having the guard true. This interpretation, together with the use of constr in the definition of guards, ensures that adding guards to a specification (as in the examples above) is a valid refinement step as will be explained in Sect. 13.4.

The semantics of guarded alt is defined by:

$$\begin{aligned} \llbracket \text{alt } [c_1 \rightarrow d_1, \dots, c_m \rightarrow d_m] \rrbracket &\stackrel{\text{def}}{=} \\ \{ \biguplus \{o_1, \dots, o_m\} \mid \forall i \in [1, m] : o_i \in \llbracket \text{seq } [\text{constr}(c_i), d_i] \rrbracket \} \end{aligned} \quad (13.9)$$

The UML 2.0 standard ([OMG05]) states that if all guards in an alt-construct are false then the empty trace $\langle \rangle$ (i.e. doing nothing) should be positive. In [RHS05b], we gave a definition of guarded alt which was consistent with the standard. However, implicitly adding the empty trace as positive implies that alt is no longer associative. For this reason, we have omitted this implicit trace from definition (13.9).

Definition (13.9) is consistent with our general belief that everything which is not explicitly described in an interaction should be regarded as inconclusive for that diagram. If all guards are false, all of the described traces are negative and the interaction has an empty set of positive traces. To avoid confusion between our definition and that of UML, we recommend to always make sure that the guards of an alt-construct are exhaustive. If desired, one of the alternatives may be the empty diagram, skip, defining the empty trace as positive:

$$\llbracket \text{skip} \rrbracket \stackrel{\text{def}}{=} \{(\{\langle \rangle\}, \emptyset)\} \quad (13.10)$$

The pragmatics of guards

- Use guards in an alt/xalt-construct to constrain the situations in which the different alternatives are positive.
- Always make sure that for each alternative, the guard is sufficiently general to capture all possible situations in which the described traces are positive.
- In an alt-construct, make sure that the guards are exhaustive. If doing nothing is valid, specify this by using the empty diagram, skip.

13.3.3 The Use of refuse, veto and assert

As explained in the introduction, interactions are incomplete specifications, specifying only example runs as opposed to the complete behaviour of the system. In this setting, it is particularly important to specify not only positive, but also negative traces, stating what the system is *not* allowed to do. In STAIRS, negative traces are defined by using one of the operators refuse, veto, or assert. The operators refuse and veto are both used to specify that the traces of its operand should be considered negative. They differ only in that veto has the empty trace as positive, while refuse does not have any positive traces at all. The importance of this difference will be

explained later in this section, after the formal definitions. The assert operator specifies that only the traces in its operand are positive and that all other traces are negative.

In the revised version of DecideAppTime given in Fig. 13.9, these three operators are used in order to add negative traces to the specification in Fig. 13.4. Figure 13.9 also adds some positive traces via the loop-construct, which may be interpreted as an alt between performing the contents of the loop 0, 1, 2, 3, or 4 times. For a formal definition of loop, see [RHS05b].

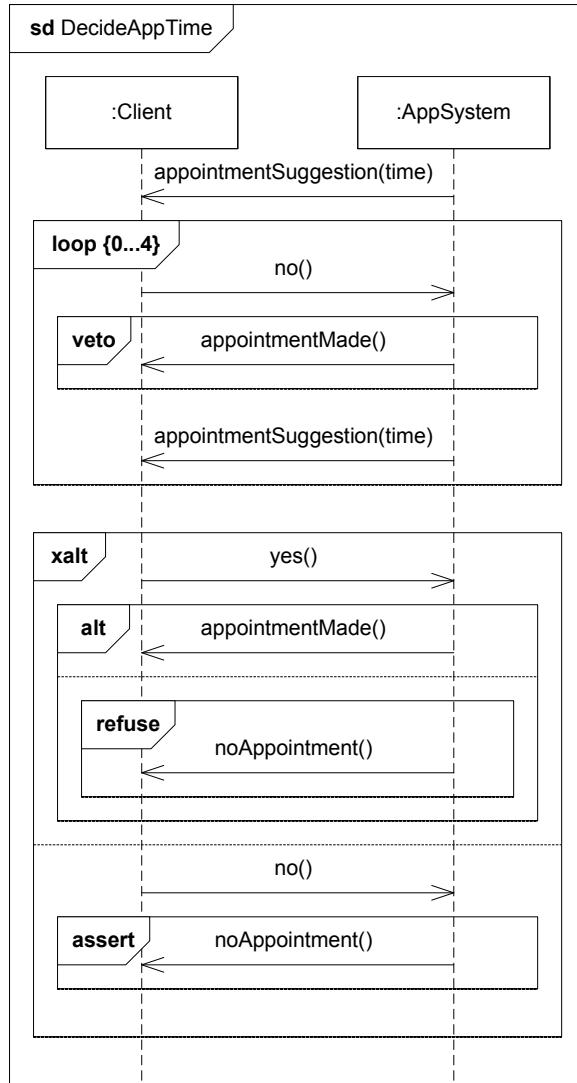


Figure 13.9: DecideAppTime revisited

Inside the loop, veto is used to specify that after the client has answered no to the suggested appointment, the system should not send the message AppointmentMade before suggesting another appointment. In the first xalt-operand, alt in combination with refuse is used to specify that the client should get the receipt message AppointmentMade when he accepts the suggested appointment, and that it would be negative if he instead got the message noAppointment. In the second xalt-operand, assert is used to specify that the system should send the message noAp-

pointment after the client has answered with the final no, and that no other traces are allowed. This is in contrast to the first `xalt`-operand, which defines one positive and one negative trace, but leaves all other traces inconclusive.

Formally, `refuse` and `veto` are defined by:

$$\llbracket \text{refuse } [d] \rrbracket \stackrel{\text{def}}{=} \{(\emptyset, p \cup n) \mid (p, n) \in \llbracket d \rrbracket\} \quad (13.11)$$

$$\text{veto } [d] \stackrel{\text{def}}{=} \text{alt } [\text{skip}, \text{refuse } [d]] \quad (13.12)$$

Both operators define that all traces described by its operand should be considered negative. The difference between `refuse` and `veto` is that while `refuse` has *no* positive traces, `veto` has the empty trace as positive, meaning that doing nothing is positive for `veto`. To understand the importance of this difference, it is useful to imagine that for each lifeline, each interaction fragment is implemented as a subroutine. Entering a new interaction fragment will then correspond to calling the subroutine that implements this fragment. For an interaction fragment with `refuse` as its main operator, no such subroutine may exist, as there are no positive traces. Hence, the program fails to continue in such a case. However, an interaction fragment with `veto` as its main operator, corresponds to an empty routine that immediately returns and the program may continue with the interaction fragment that follows.

The choice of operator for a concrete situation, will then depend on the question: Should doing nothing be possible in this otherwise negative situation? If yes, use `veto`. If no, use `refuse`.

Consider again Fig. 13.9. Here, `veto` is used inside the loop construct as sending the message `no` (then doing nothing), and then sending `AppointmentSuggestion(time)` should be positive. On the other hand, `refuse` is used in the first `xalt` operand, as we did not want to specify the message `yes` followed by doing nothing as positive.

Using `assert` ensures that for each interaction obligation of its operand, at least one of the described positive traces will be implemented by the final system, as all inconclusive traces are redefined as negative. Formally:

$$\llbracket \text{assert } [d] \rrbracket \stackrel{\text{def}}{=} \{(p, n \cup (\mathcal{H} \setminus p)) \mid (p, n) \in \llbracket d \rrbracket\} \quad (13.13)$$

The pragmatics of negation

- To effectively constrain the implementation, the specification should include a reasonable set of negative traces.
- Use `refuse` when specifying that one of the alternatives in an `alt`-construct represents negative traces.
- Use `veto` when the empty trace (i.e. doing nothing) should be positive, as when specifying a negative message in an otherwise positive scenario.
- Use `assert` on an interaction fragment when all possible positive traces for that fragment have been described.

13.3.4 The Use of `seq`

As explained in Sect. 13.2, the weak sequencing operator `seq` is the implicit composition operator for interactions, defined by the following invariants:

- The ordering of events within each of the operands is maintained in the result.
- Events on different lifelines from different operands may come in any order.
- Events on the same lifeline from different operands are ordered such that an event of the first operand comes before that of the second operand, and so on.

Consider again the revised specification of CancelAppointment in Fig. 13.7. In the second `xalt`-operand, the system sends the message `appointmentCancelled` to the client, and subsequently the referenced interaction `DecideAppTime` is performed. Here, the first thing to happen is that the system sends the message `AppointmentSuggestion` to the client (as specified in Fig. 13.9).

As `seq` is the operator used for sequencing interaction fragments, this means that in general no synchronization takes place at the beginning of an interaction fragment, i.e. that different lifelines may enter the fragment at different points in time. In the context of Fig. 13.7, this means that there is no ordering between the reception of the message `appointmentCancelled` and the sending of the message `AppointmentSuggestion`, in exactly the same way as there would have been no ordering between these if the specification had been written in one single diagram instead of using a referenced interaction.

In Fig. 13.10, traces are added to the specification of CancelAppointment in the case where the client wants to cancel an appointment less than 24 hours before it is supposed to take place.

The first `alt`-operand specifies that the system may give an error message (as before). The second operand specifies that the sending of the message `appointmentCancelled` alone is negative, while the third operand specifies that sending the message `appointmentCancelled` and then performing (the positive traces of) `Payment` (specified in Fig. 13.11) is positive.

This example demonstrates that a trace (e.g. `appointmentCancelled` followed by `Payment`) is not necessarily negative even if a prefix of it (e.g. `appointmentCancelled`) is. This means that the total trace must be considered when categorizing it as positive, negative or inconclusive. Another consequence is that every trace which is not explicitly shown in the interaction should be inconclusive. For instance, in Fig. 13.10 all traces where the message `appointmentCancelled` is followed by something other than `Payment`, are still inconclusive.

The formal definition of `seq` was given in Sect. 13.2. As no synchronization takes place at the beginning of each `seq`-operand, it follows from the definitions that i.e. $\text{seq } [d_1, \text{alt } [d_2, d_3]] = \text{alt } [\text{seq } [d_1, d_2], \text{seq } [d_1, d_3]]$ and that $\text{loop } \{2\} [d] = \text{seq } [d, d]$ as could be expected.

The pragmatics of weak sequencing

- Be aware that by weak sequencing,
 - a positive sub-trace followed by a positive sub-trace is positive.
 - a positive sub-trace followed by a negative sub-trace is negative.
 - a negative sub-trace followed by a positive sub-trace is negative.
 - a negative sub-trace followed by a negative sub-trace is negative.
 - the remaining trace combinations are inconclusive.

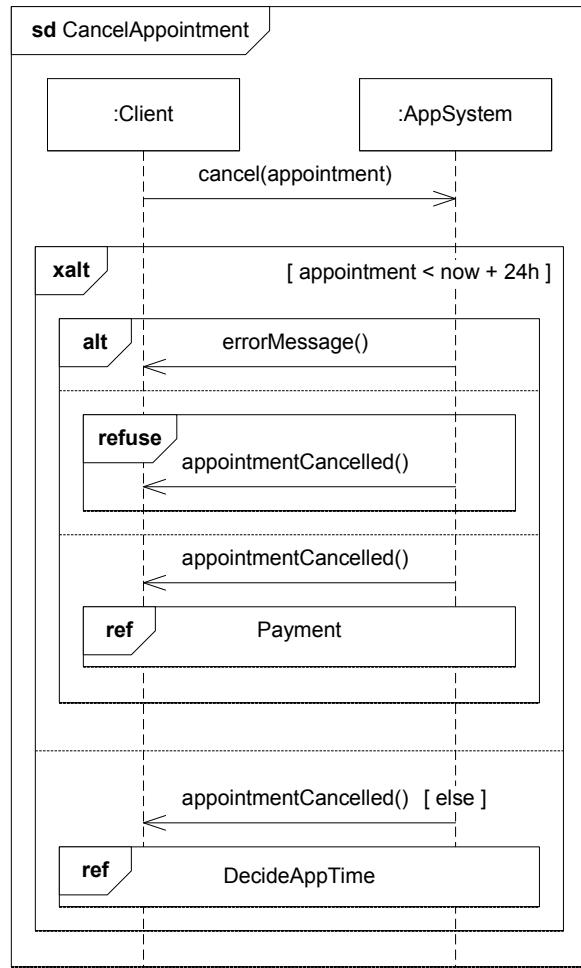


Figure 13.10: CancelAppointment revisited

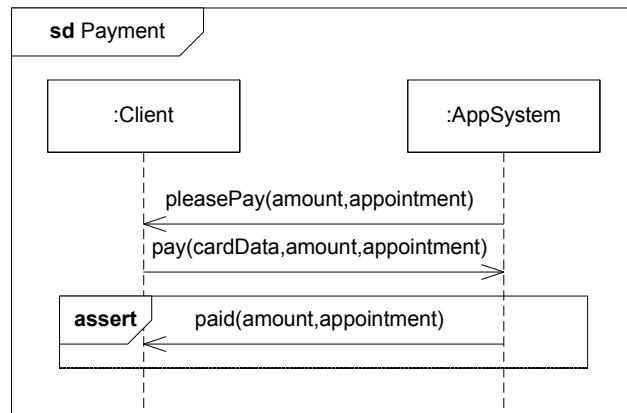


Figure 13.11: Payment

13.4 The Pragmatics of Refining Interactions

In a development process, specifications may be changed for several reasons, including capturing new user requirements, giving a more detailed design, or correcting errors. STAIRS focuses on those changes which may be defined as refinements. In this section, we explain some main kinds of refinement in STAIRS, and demonstrate how each of the development steps taken in the example in Sect. 13.3 are valid refinement steps.

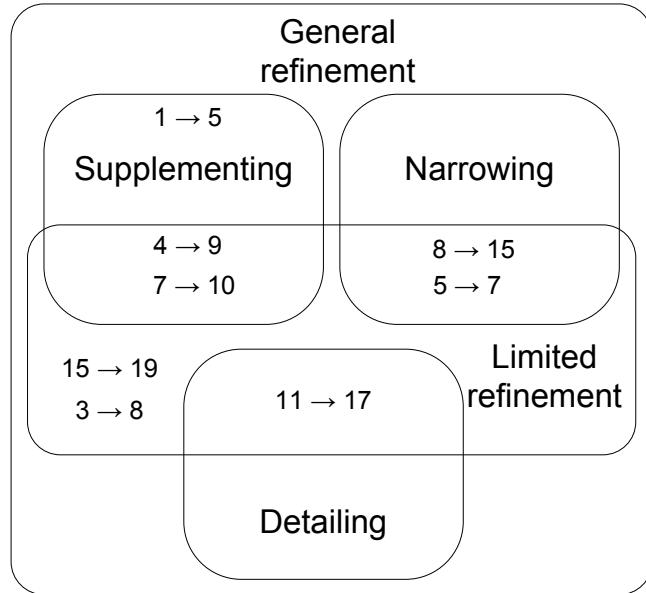


Figure 13.12: The refinement relations of STAIRS

Figure 13.12 illustrates how the different refinement notions presented in this paper are related. Supplementing, narrowing and detailing are all special cases of the general refinement notion. Limited refinement is a restricted version of general refinement, which limits the possibility to increase the nondeterminism required of an implementation. In Fig. 13.12, we have also illustrated what refinement relation is used for each of the development steps in our running example. For instance, the placement of $1 \rightarrow 5$ means that Fig. 5 is a supplementing and general refinement of Fig. 1, but not a limited refinement.

In the discussion of each of the five refinement notions, we will refer to the following definition of compositionality:

Definition 1 (Compositionality) *A refinement operator \rightsquigarrow is compositional if it is*

- *reflexive: $d \rightsquigarrow d$*
- *transitive: $d \rightsquigarrow d' \wedge d' \rightsquigarrow d'' \Rightarrow d \rightsquigarrow d''$*

- monotonic with respect to refuse, veto, (guarded) alt, (guarded) xalt and seq:

$$\begin{aligned}
 d \rightsquigarrow d' &\Rightarrow \text{refuse } [d] \rightsquigarrow \text{refuse } [d'] \\
 d \rightsquigarrow d' &\Rightarrow \text{veto } [d] \rightsquigarrow \text{veto } [d'] \\
 d_1 \rightsquigarrow d'_1, \dots, d_m \rightsquigarrow d'_m &\Rightarrow \text{alt } [d_1, \dots, d_m] \rightsquigarrow \text{alt } [d'_1, \dots, d'_m] \\
 d_1 \rightsquigarrow d'_1, \dots, d_m \rightsquigarrow d'_m &\Rightarrow \text{xalt } [d_1, \dots, d_m] \rightsquigarrow \text{xalt } [d'_1, \dots, d'_m] \\
 d_1 \rightsquigarrow d'_1, \dots, d_m \rightsquigarrow d'_m &\Rightarrow \text{seq } [d_1, \dots, d_m] \rightsquigarrow \text{seq } [d'_1, \dots, d'_m]
 \end{aligned}$$

Transitivity enables the stepwise development of interactions, while monotonicity is important as it means that the different parts of an interaction may be refined separately.

13.4.1 The Use of Supplementing

As interactions are incomplete specifications typically describing only example runs, we may usually find many possible traces that are inconclusive in a given interaction obligation. By supplementing, inconclusive traces are re-categorized as either positive or negative as illustrated for a single interaction obligation in Fig. 13.13. Supplementing is an activity where new situations are considered, and will most typically be used during the early phases of system development. Examples of supplementing includes capturing new user requirements and adding fault tolerance to the system.

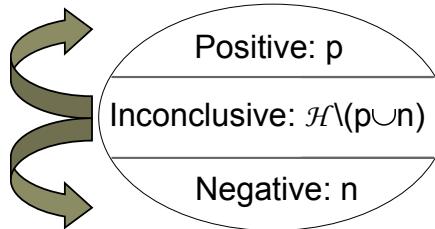


Figure 13.13: Supplementing of interaction obligations

DecideAppTime in Fig. 13.9 is an example of supplementing, as it adds both positive and negative traces to the specification in Fig. 13.4. All traces that were positive in the original specification, are still positive in the refinement. Another example of supplementing is CancelAppointment in Fig. 13.10, which adds traces to the specification in Fig. 13.7. Again, all traces that were positive in the original specification remain positive in the refinement, and the negative traces remain negative.

Formally, supplementing of interaction obligations is defined by:

$$(p, n) \rightsquigarrow_s (p', n') \stackrel{\text{def}}{=} p \subseteq p' \wedge n \subseteq n' \quad (13.14)$$

For an interaction with a set of interaction obligations as its semantics, we require that each obligation for the original interaction must have a refining obligation in the semantics of the refined interaction. This ensures that the alternative traces (e.g. the inherent nondeterminism) required by an interaction are also required by the refinement. Formally:

$$d \rightsquigarrow_s d' \stackrel{\text{def}}{=} \forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow_s o' \quad (13.15)$$

Supplementing is compositional as defined by Definition 1.

The pragmatics of supplementing

- Use supplementing to add positive or negative traces to the specification.
- When supplementing, all of the original positive traces must remain positive and all of the original negative traces must remain negative.
- Do not use supplementing on the operand of an assert.

13.4.2 The Use of Narrowing

Narrowing means to reduce underspecification by redefining positive traces as negative, as illustrated in Fig. 13.14. As for supplementing, negative traces must remain negative in the refinement.

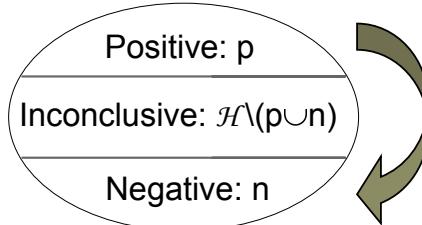


Figure 13.14: Narrowing of interaction obligations

One example of narrowing, is adding guards to `CancelAppointment` in Fig. 13.7. In the original specification in Fig. 13.5, we had for instance no constraint on the alternative with the message `appointmentCancelled`, while in the refinement this alternative is negative if it occurs less than 24 hours prior the appointment.

In general, adding guards to an `alt/xalt`-construct is a valid refinement through narrowing. Seeing that an operand without a guard is interpreted as having true as guard, this is a special case of a more general rule, stating that a valid refinement may limit a guard as long as the refined condition implies the original one. This ensures that all of the positive traces of the refinement were also positive (and not negative) in the original specification.

Another example of narrowing is given in `MakeAppointment` in Fig. 13.15. Here, the `refuse`-operator is used to specify that the client may *not* ask for an appointment at a specific hour. This means that even though these traces were positive in the specification in Fig. 13.8, they are now considered negative in the sense that asking for a specific hour is not an option in the final implementation.

Formally, narrowing of interaction obligations is defined by:

$$(p, n) \rightsquigarrow_n (p', n') \stackrel{\text{def}}{=} p' \subseteq p \wedge n' = n \cup p \setminus p' \quad (13.16)$$

and narrowing of interactions by:

$$d \rightsquigarrow_n d' \stackrel{\text{def}}{=} \forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow_n o' \quad (13.17)$$

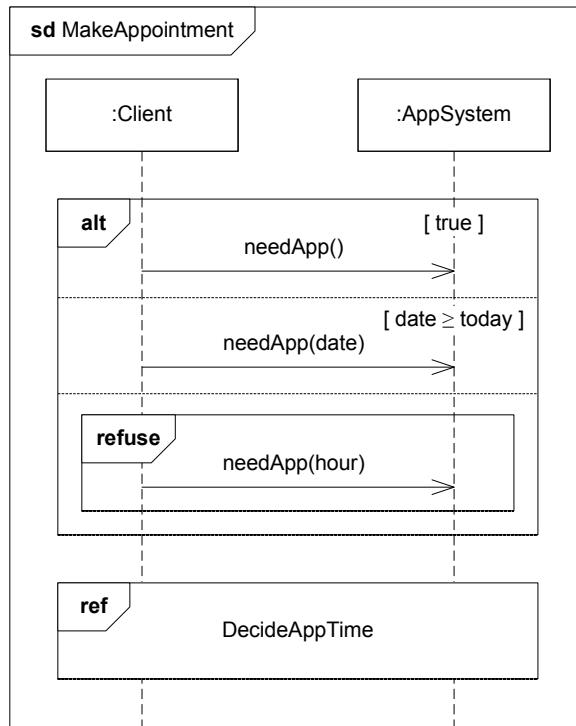


Figure 13.15: MakeAppointment revisited

Narrowing is compositional as defined by Definition 1. In addition, the narrowing operator \rightsquigarrow_n is monotonic with respect to assert.

The pragmatics of narrowing

- Use narrowing to remove underspecification by redefining positive traces as negative.
- In cases of narrowing, all of the original negative traces must remain negative.
- Guards may be added to an alt-construct as a legal narrowing step.
- Guards may be added to an xalt-construct as a legal narrowing step.
- Guards may be narrowed, i.e. the refined condition must imply the original one.

13.4.3 The Use of Detailing

Detailing means reducing the level of abstraction by decomposing one or more lifelines, i.e. by structural decomposition. As illustrated in Fig. 13.16, positive traces remain positive and negative traces remain negative in relation to detailing. The only change is that the traces of the refinement may include more details, for instance internal messages that are not visible in the more abstract specification.

Figure 13.17 is a detailing refinement of Payment in Fig. 13.11. In this case, the lifeline AppSystem is decomposed into the two lifelines Calendar, taking care of appointments, and Billing, handling payments. This decomposition has two effects with respect to the traces of

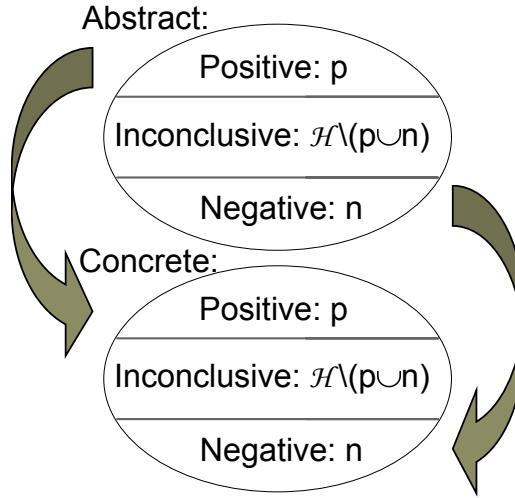


Figure 13.16: Detailing of interaction obligations

the original specification. First of all, internal communication between Billing and Calendar is revealed (i.e. the messages `needPay` and `paymentReceived`), and secondly, Billing has replaced AppSystem as the sender/receiver of messages to and from the client. In general, some of the client's messages could also have been sent to/from Calendar.

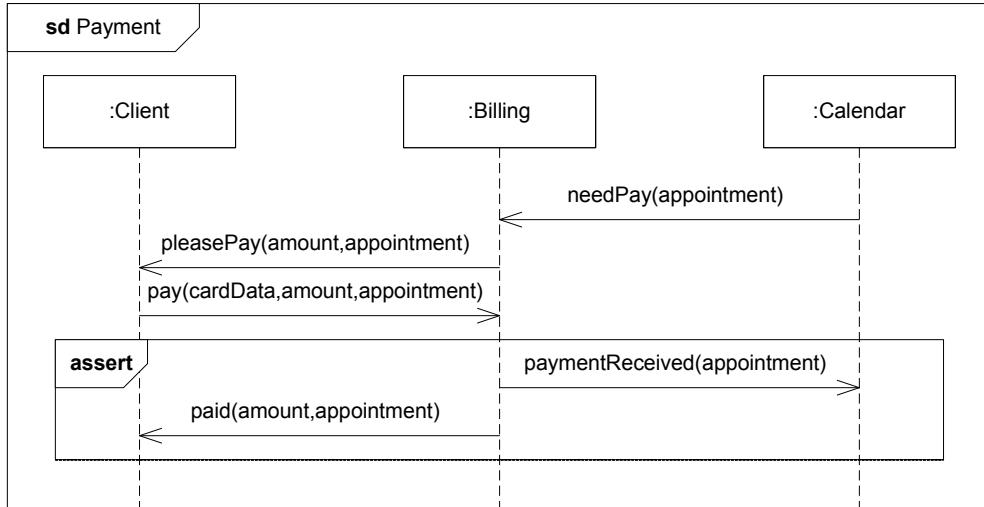


Figure 13.17: Payment with decomposition

We say that an interaction is a detailing refinement if we get the same positive and negative traces as in the original specification when both hiding the internal communication in the decomposition and allowing for a possible change in the sender/receiver of a message. Formally, the lifeline decomposition will in each case be described by a mapping L from concrete to abstract lifelines. For the above example, we get

$$L = ID[\text{Billing} \mapsto \text{AppSystem}][\text{Calendar} \mapsto \text{AppSystem}]$$

where ID is the identity mapping, and $L[A \mapsto B]$ is the mapping L updated so that A maps to B .

Formally, we need to define a substitution function $\text{subst}(t, L)$, which substitutes lifelines in the trace t according to the mapping L . First, we define substitution on single events:

$$\text{subst}(e, L) \stackrel{\text{def}}{=} \begin{cases} k(s, L(\text{tr}), L(\text{re})) & \text{if } e = k(s, \text{tr}, \text{re}), k \in \{!, ?\} \\ e & \text{otherwise} \end{cases} \quad (13.18)$$

In general, a trace t may be seen as a function from indices to events. This trace function may be represented as a mapping where each element $i \mapsto e$ indicates that e is the i 'th element in the trace, and we define the substitution function on traces by:

$$\text{subst}(t, L) \stackrel{\text{def}}{=} \{i \mapsto \text{subst}(t[i], L) \mid i \in [1 \dots \#t]\} \quad (13.19)$$

where $\#t$ and $t[i]$ denotes the length and the i 'th element of the trace t , respectively.

We then define an abstraction function $\text{abstr}(t, L, E)$, which transforms a concrete trace into an abstract trace by removing all internal events (with respect to L) that are not present in the set of abstract events E :

$$\text{abstr}(t, L, E) \stackrel{\text{def}}{=} \{e \in \mathcal{E} \mid \text{tr.e} \neq \text{re.e} \vee e \in E\} \odot (\text{subst}(t, L)) \quad (13.20)$$

where \mathcal{E} denotes the set of all events, tr.e and re.e denote the transmitter and the receiver of the event e , and $A \odot t$ is the trace t with all events not in the set A removed. We also overload abstr to trace sets in standard pointwise manner:

$$\text{abstr}(s, L, E) \stackrel{\text{def}}{=} \{\text{abstr}(t, L, E) \mid t \in s\} \quad (13.21)$$

Formally, detailing of interaction obligations is then defined by:

$$(p, n) \rightsquigarrow_c^{L, E} (p', n') \stackrel{\text{def}}{=} \text{abstr}(p, L, E) = \text{abstr}(p', L, E) \wedge \quad (13.22)$$

$$\text{abstr}(n, L, E) = \text{abstr}(n', L, E) \quad (13.23)$$

where L is a lifeline mapping as described above, and E is a set of abstract events.

Finally, detailing of interactions is defined by:

$$d \rightsquigarrow_c^{L, E} d' \stackrel{\text{def}}{=} \forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow_c^{L, E} o' \quad (13.24)$$

Detailing is compositional as defined by Definition 1. In addition, the detailing operator $\rightsquigarrow_c^{L, E}$ is monotonic with respect to assert.

The pragmatics of detailing

- Use detailing to increase the level of granularity of the specification by decomposing lifelines.
- When detailing, document the decomposition by creating a mapping L from the concrete to the abstract lifelines.
- When detailing, make sure that the refined traces are equal to the original ones when abstracting away internal communication and taking the lifeline mapping into account.

13.4.4 The Use of General Refinement

Supplementing, narrowing and detailing are all important refinement steps when developing interactions. Often, it is useful to combine two or three of these activities into a single refinement step. We therefore define a general refinement notion, of which supplementing, narrowing and detailing are all special cases. This general notion is illustrated for one interaction obligation in Fig. 13.18.

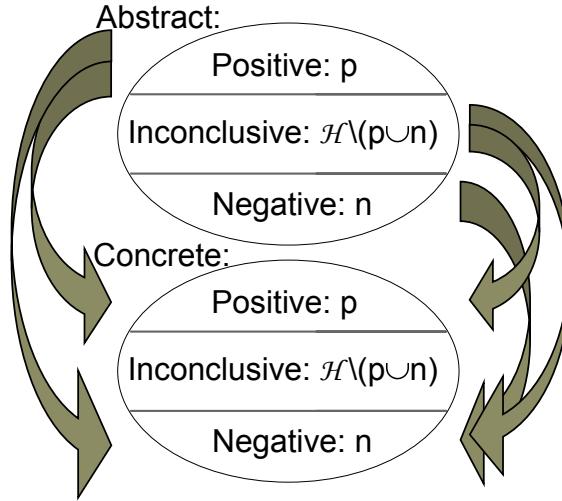


Figure 13.18: General refinement of interaction obligations

As an example of general refinement, MakeAppointment in Fig. 13.8 combines supplementing and narrowing in order to be a refinement of the interaction in Fig. 13.3. Adding an operand to the alt-construct is an example of supplementing, and is not covered by the definition of narrowing. On the other hand, adding guards is an example of narrowing, and is not covered by the definition of supplementing. For this to be a valid refinement step, we therefore need the general refinement notion, formally defined by:

$$d \rightsquigarrow_r^{L,E} d' \stackrel{\text{def}}{=} \forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow_r^{L,E} o' \quad (13.25)$$

where general refinement of interaction obligations is defined by:

$$(p, n) \rightsquigarrow_r^{L,E} (p', n') \stackrel{\text{def}}{=} abstr(n, L, E) \subseteq abstr(n', L, E) \wedge abstr(p, L, E) \subseteq abstr(p', L, E) \cup abstr(n', L, E) \quad (13.26)$$

Note that L may be the identity mapping, in which case the refinement does not include any lifeline decompositions (as in the case of MakeAppointment described above). Also, E may be the set of all events, \mathcal{E} , meaning that all events are considered when relating the traces of the refinement to the original traces. General refinement is compositional as defined by Definition 1.

Combining narrowing and supplementing may in general result in previously inconclusive traces being supplemented as positive, and the original positive traces made negative by narrowing. In order to specify that a trace *must* be present in the final implementation, and not removed

by narrowing, we need to specify an obligation with this trace as the only positive, and all other traces as negative. The only legal refinement of this operand will then be redefining the trace as negative (by narrowing), leaving an empty set of positive traces and a specification that is not implementable.

The pragmatics of general refinement

- Use general refinement to perform a combination of supplementing, narrowing and detailing in a single step.
- To define that a particular trace *must* be present in an implementation use `xalt` and `assert` to characterize an obligation with this trace as the only positive one and all other traces as negative.

13.4.5 The Use of Limited Refinement

Limited refinement is a special case of general refinement, with less possibilities for adding new interaction obligations. By definition (13.25) of general refinement, new interaction obligations may be added freely, for instance in order to increase the nondeterminism required of an interaction. One example of this is `CancelAppointment` in Fig. 13.5, which is a refinement of the interaction given in Fig. 13.1. While the original specification only gave one interaction obligation with two positive traces, the refinement gives both this interaction obligation and also two new interaction obligations that are not refinements of the original one.

At some point during the development process, it is natural to limit the possibilities for creating new interaction obligations with fundamentally new traces. This is achieved by limited refinement, which has the additional requirement that each obligation of the refined interaction must have a corresponding obligation in the original interaction.

In STAIRS, stepwise development of interactions will be performed by first using general refinement to specify the main traces of the system, before switching to limited refinement which will then be used for the rest of the development process. Typically, but not necessarily, `assert` on the complete specification will be used at the same time as switching to limited refinement. This ensures that new traces may neither be added to the existing obligations, nor be added to the specification in the form of new interaction obligations. Note that using `assert` on the complete specification is not the same as restricting further refinements to be limited, as `assert` considers each interaction obligation separately.

Note also that limited refinement allows a refinement to have more interaction obligations than the original specification, as long as each obligation is a refinement of one of the original ones. One example is given in Fig. 13.19, which is a limited refinement of `MakeAppointment` in Fig. 13.15. In Fig. 13.19, `alt` has been replaced by `xalt` in order to specify that the client *must* be offered the choice of specifying a preferred date when asking for an appointment, while `assert` has been added to specify that there should be no other alternatives. In this particular case, we have not included the referenced interaction `DecideAppTime` in the scope of the `assert`-construct, as we want the possibility of supplementing more traces here. Transforming `alt` to `xalt` means in this example that each of the interaction obligations for Fig. 13.15 (there are two due to the `xalt` in `DecideAppTime`) has two refining obligations in the semantics of Fig. 13.19. As

all obligations in Fig. 13.19 have a corresponding obligation in Fig. 13.15, this is a valid instance of limited refinement.

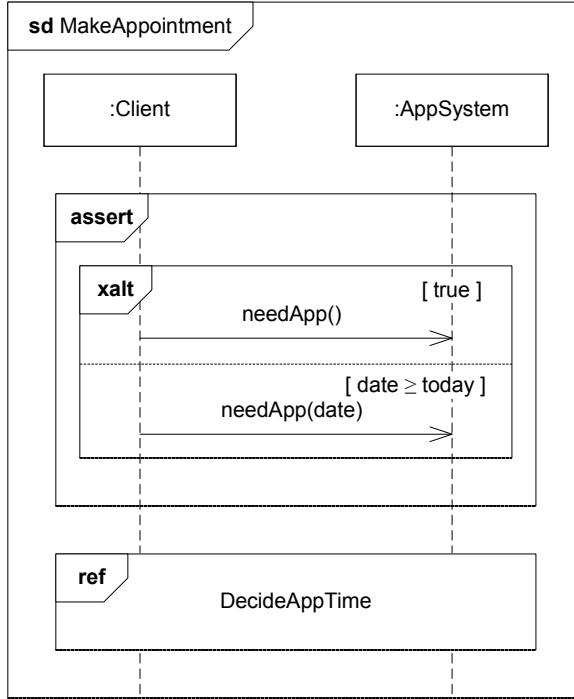


Figure 13.19: MakeAppointment revisited

Formally, limited refinement is defined by:

$$d \rightsquigarrow_l^{L,E} d' \stackrel{\text{def}}{=} d \rightsquigarrow_r^{L,E} d' \wedge \forall o' \in \llbracket d' \rrbracket : \exists o \in \llbracket d \rrbracket : o \rightsquigarrow_r^{L,E} o' \quad (13.27)$$

Limited refinement is compositional as defined by Definition 1.

The pragmatics of limited refinement

- Use assert and switch to limited refinement in order to avoid fundamentally new traces being added to the specification.
- To specify globally negative traces, define these as negative in all operands of xalt, and switch to limited refinement.

13.5 Related Work

The basis of STAIRS is interactions and sequence diagrams as defined in UML 2.0 [OMG05]. Not all of the UML 2.0 operators are defined by STAIRS, but we believe that those covered are the most useful ones in practical system development. The STAIRS operator xalt is added to this set as UML 2.0 does not distinguish between alternatives that represent underspecification and alternatives that must all be present in an implementation, but uses the alt operator in both cases.

For guarded alt, we have in our semantics chosen not to follow UML 2.0 in that the empty trace is positive if no guard is true. Instead, we recommend to make all specifications with guarded alt so that the guards are exhaustive, ensuring that this will never be a problem in practice. The UML 2.0 standard [OMG05] is vague with respect to whether the traces with a false guard should be negative or not. As we have argued, classifying these as negative is fruitful as adding guards to a specification will then be a valid refinement step.

For defining negative behaviour, UML 2.0 uses the operators neg and assert. In [RHS05a], we investigated several possible formal definitions of neg, trying to capture how it was being used on the basis of experience. However, we concluded that one operator for negation is not sufficient, which is why STAIRS defines the two operators refuse and veto to be used instead of neg.

Decomposition in UML 2.0 is the same as detailing in STAIRS, but with a more involved syntax using the concepts of interaction use and gates. With the UML 2.0 syntax, the mapping from concrete to abstract lifelines is given explicitly in the diagram.

In [GS05], Grosu and Smolka give semantics for UML sequence diagrams in the form of two Büchi automata, one for the positive and one for the negative behaviours. Refinement then corresponds to language inclusion. Their refinement notion is compositional and covers supplementing and narrowing, but not detailing. All alternatives are interpreted as underspecification, and there is no means to capture inherent nondeterminism as with xalt in STAIRS.

In [CK04], the semantics of UML interactions are defined by the notions of positive and negative satisfaction. This approach is in many ways similar to STAIRS, but does not distinguish between underspecification and inherent nondeterminism. Their definition of the UML operator neg corresponds to the STAIRS operator veto, where the empty trace is taken as positive. [CK04] defines that for alternatives specified by alt, a trace is negative only if it is negative in both operands. Also, a trace is regarded as negative if a prefix of it is described as negative, while we in STAIRS define it as inconclusive as long as the complete trace is not described by the diagram.

Another variant of sequence diagrams is Message Sequence Charts (MSCs) [ITU99]. The important distinction between different kinds of alternatives is not made for MSCs either. As in our approach, a trace is negative if its guard is false in an MSC. Refinement of MSCs is considered by Krüger in [Krü00]. Narrowing in STAIRS corresponds closely to property refinement, while detailing corresponds to structural refinement. As there is no notion of inconclusive traces in [Krü00], refinement in the form of supplementing is not considered.

Live Sequence Charts (LSCs) [DH99, HM03] is an extension of MSCs, where charts, messages, locations and conditions are specified as either universal (mandatory) or existential (optional). An existential chart specifies a behaviour (one or more traces) that must be satisfied by at least one system run, while a universal chart is a specification that must be satisfied at all times. As a universal chart specifies all allowed traces, this is not the same as inherent nondeterminism in STAIRS, which only specifies some of the traces that must be present in an implementation. In contrast to STAIRS and UML 2.0, LSC synchronizes the lifelines at the beginning of each interaction fragment. This reduces the set of possible traces, and makes it easier to implement their operational semantics.

13.6 Conclusions and Future Work

In this paper we have focused on giving practical guidelines for the use of STAIRS in the development of interactions. For each concept, these guidelines have been summarized in paragraphs entitled “The Pragmatics of...”. We have focused on situations in which STAIRS extends or expands interactions as defined in UML 2.0 [OMG05]. This includes how to define negative behaviours and how to distinguish between alternatives that represent the same behaviour and alternatives that must all be present in an implementation. STAIRS is particularly concerned with refinement, and we have given guidelines on how to refine interactions by adding behaviours (supplementing), removing underspecification (narrowing) or by decomposition (detailing).

In [RHS05b], we gave a brief explanation of what it means for an implementation to be correct with respect to a STAIRS specification. We are currently working on extending this work, leading to “the pragmatics of implementations”.

Acknowledgements. The research on which this paper reports has been partly carried out within the context of the IKT-2010 project SARDAS (15295/431). We thank the other members of the SARDAS project for useful discussions related to this work. We thank Iselin Engan for helpful comments on the final draft of this paper. We also thank the anonymous reviewers for constructive feedback.

References

- [CK04] María Victoria Cengarle and Alexander Knapp. UML 2.0 interactions: Semantics and refinement. In *Proc. 3rd Int. Wsh. Critical Systems Development with UML (CSDUML'04)*, Technical report TUM-I0415, pages 85–99. Institut für Informatik, Technische Universität München, 2004.
- [DH99] Werner Damm and David Harel. LSC’s: Breathing life into message sequence charts. In *Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, 1999.
- [GS05] Radu Grosu and Scott A. Smolka. Safety-liveness semantics for UML sequence diagrams. In *Proc. 5th Int. Conf. on Applications of Concurrency to System Design (ACSD'05)*, pages 6–14, 2005.
- [HHR05a] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. STAIRS towards formal design with sequence diagrams. *Journal of Software and Systems Modeling*, 22(4):349–458, 2005.
- [HHR05b] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. Why timed sequence diagrams require three-event semantics. In *Scenarios: Models, Transformations and Tools*, volume 3466 of *LNCS*, pages 1–25. Springer, 2005.
- [HM03] David Harel and Rami Marelly. *Come, Let’s Play.: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.
- [HS03] Øystein Haugen and Ketil Stølen. STAIRS — Steps to analyze interactions with refinement semantics. In *Proc. International Conference on UML (UML'2003)*, volume 2863 of *LNCS*, pages 388–402. Springer, 2003.
- [ITU99] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1999.

- [Krü00] Ingolf Heiko Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.
- [Kru04] Philippe Kruchten. *The Rational Unified Process*. Addison-Wesley, third edition, 2004.
- [OMG05] Object Management Group. *UML Superstructure Specification, v. 2.0*, document: formal/05-07-04 edition, 2005.
- [RHS05a] Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen. How to transform UML neg into a useful construct. In *Norsk Informatikkonferanse NIK'2005*, pages 55–66. Tapir, 2005.
- [RHS05b] Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen. Refining UML interactions with underspecification and nondeterminism. *Nordic Journal of Computing*, 12(2):157–188, 2005.
- [RRS06] Atle Refsdal, Ragnhild Kobro Runde, and Ketil Stølen. Underspecification, inherent nondeterminism and probability in sequence diagrams. In *Proc. 8th IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'06)*, volume 4037 of *LNCS*, pages 138–155. Springer, 2006.

Chapter 14

Underspecification, Inherent Nondeterminism and Probability in Sequence Diagrams

Atle Refsdal, Ragnhild Kobro Runde, and Ketil Stølen

Publication.

Published as Technical Report 335, Department of Informatics, University of Oslo, 2006. Extended version of: Underspecification, inherent nondeterminism and probability in sequence diagrams. In *Proc. Formal Methods for Open Object-Based Distributed Systems (FMOODS 2006)*, volume 4037 of *LNCS*, pages 138–155. Springer, 2006.

Abstract.

Nondeterminism in specifications may be used for at least two different purposes. One is to express underspecification, which means that the specifier for the same environment behavior allows several alternative behaviors of the specified component and leaves the choice between these to those responsible for implementing the specification. In this case a valid implementation will need to implement at least one, but not necessarily all, alternatives. The other purpose is to express inherent nondeterminism, which means that a valid implementation needs to reflect all alternatives. STAIRS is an approach to the compositional and incremental development of sequence diagrams supporting underspecification as well as inherent nondeterminism. Probabilistic STAIRS builds on STAIRS and allows probabilities to be included in the specifications. Underspecification with respect to probabilities is also allowed. This paper investigates the use of underspecification, inherent nondeterminism and probability in sequence diagrams, the relationships between these concepts, and how these are expressed in STAIRS and probabilistic STAIRS.

14.1 Introduction

Nondeterminism in specifications may be used for expressing underspecification as well as inherent nondeterminism. Underspecification means that the specifier leaves some freedom of choice to those who will implement or further refine the specification. This is useful when different design alternatives fulfill a function equally well from the specifier's point of view. For example, when specifying an automatic teller machine we need to ensure that money is delivered and the card is returned at the end of the transaction. But whether the card is returned before or after the money is not important, and we may leave the choice to those responsible for making the teller machine.

Inherent nondeterminism, on the other hand, means that all alternatives must be reflected also in the final implementation. For example, when specifying a program to simulate a coin flip it is essential that both heads and tails are possible outcomes. An inherently nondeterministic choice can be seen as an abstraction of a probabilistic choice where the probabilities are greater than 0 but otherwise unknown.

The difference between underspecification and inherent nondeterminism is related to refinement. In an implementation, which is not supposed to be refined and has no underspecification, the distinction is not relevant.

STAIRS ([HHR05b],[RHS05c]) is a method for the compositional development of interactions, such as sequence diagrams and interaction overview diagrams. STAIRS employs two different choice operators to distinguish between underspecification and inherent nondeterminism; `alt` represents underspecification and `xalt` represents inherent nondeterminism. Probabilistic STAIRS ([RHS05a]) replaces `xalt` with the `palt` operator that also allows specification of probabilities on its operands.

STAIRS includes all the main composition operators of UML 2.0 interactions, such as `seq` and `par` for specifying sequential and parallel composition respectively. As these operators are not important for the discussion in this paper, we refer to [HHR05b] for formal definitions and examples using these operators.

This paper summarizes insights gained during our work with formalization of various forms of nondeterminism in STAIRS and probabilistic STAIRS by investigating the different roles of nondeterminism in interactions. In particular we

- demonstrate the usefulness of underspecification, inherent nondeterminism and probability in specifications,
- show that these concepts are adequately expressed in STAIRS and probabilistic STAIRS by the operators `alt`, `xalt` and `palt`,
- explore the properties of these operators, in particular with respect to refinement,
- provide simple examples that give a thorough understanding of the use of these operators, both separately and combined.

The paper is organized as follows: Section 14.2 discusses underspecification and its representation in a simplified version of STAIRS. In Section 14.3 we motivate the need for inherent nondeterminism and show how this is incorporated in full STAIRS. Section 14.4 introduces

probabilistic STAIRS. We discuss related work in Section 14.5 before concluding in Section 14.6.

14.2 Underspecification

14.2.1 Motivation

Often, it is useful to write specifications where certain aspects of the behavior of the system are left open. This is known as underspecification. In many cases, underspecification will be an implicit consequence of using abstraction when describing the important features of a system. Many specification languages also include some kind of “or” operator for explicitly specifying the alternatives the implementer may choose between. In STAIRS, this is the `alt` operator.

In our setting of interactions, the `alt` operator may be used to describe scenarios that are different, but still seen as alternative means to achieve the same purpose in some sense. The `alt` operator is also called potential choice, as the alternatives represent choices that the implementation may choose between in order to satisfy the specification. As an everyday example, consider the action of making a u-turn when walking. This may be achieved by turning either 180 degrees left or 180 degrees right. Which alternative you choose is usually insignificant.

14.2.2 Semantic Representation

In STAIRS the semantics of an interaction is defined by denotational trace semantics, where a trace is a sequence of events representing a system run. We denote the semantics of an interaction d by $\llbracket d \rrbracket$. For the subset of STAIRS presented so far, containing only underspecification (and not inherent nondeterminism) the semantics of an interaction is represented by an *interaction obligation* (p, n) . Here, p is a set of positive traces, representing desired or acceptable behavior, while n is a set of negative traces, representing undesired or unacceptable behavior.

An interaction is a partial specification in the sense that it does not in general define all the behavior of the system. Traces that are neither positive nor negative are called inconclusive, meaning that these are traces that the specifier has not yet considered. Letting \mathcal{H} denote the universe of all traces, the traces $\mathcal{H} \setminus (p \cup n)$ are inconclusive in the obligation (p, n) .

From an implementation point of view, there is no distinction between inconclusive and positive traces, as they all represent possible behaviors of the system. However, conceptually there is an important difference between behaviors that are explicitly described and behaviors that are not. Also, positive and inconclusive traces are treated differently by composition operators such as `seq` (sequential composition) and `par` (parallel composition), see [HHR05b].

Underspecification by means of `alt` corresponds to taking the pairwise union of the positive and negative trace-sets of the operands. Formally:

$$\llbracket d_1 \text{ alt } d_2 \rrbracket \stackrel{\text{def}}{=} \llbracket d_1 \rrbracket \uplus \llbracket d_2 \rrbracket \quad (14.1)$$

where

$$(p_1, n_1) \uplus (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \cup p_2, n_1 \cup n_2) \quad (14.2)$$

From this definition it is clear that the `alt` operator can be used not only to introduce underspecification in the form of alternative ways of fulfilling a task (i.e. new positive traces), but

also to introduce more restrictions by adding new negative traces. By taking the union also of the negative traces, the `alt` operator can be used to merge alternatives that are considered to be similar, both at the positive and the negative level. In addition, the above definition ensures monotonicity of refinement with respect to `alt`, which will be clear from the following sections.

14.2.3 Refinement

Refinement of a specification means to reduce underspecification by adding information so that the specification becomes closer to an implementation. As in [HHR05b], we distinguish between two special cases of refinement, called narrowing and supplementing. Narrowing reduces the set of positive traces to capture new design decisions or to match the problem more accurately. Supplementing categorizes (to this point) inconclusive behavior as either positive or negative. Formally, an interaction obligation (p', n') is a refinement of an interaction obligation (p, n) , written $(p, n) \rightsquigarrow (p', n')$, if

$$n \subseteq n' \wedge p \subseteq p' \cup n' \quad (14.3)$$

Intuitively, supplementing means that it is possible to add new positive or negative traces to those already specified. Specifying more alternative traces is usually achieved by using the `alt` operator, meaning that we want $d_1 \text{ alt } d_2$ to be a valid refinement of d_1 (and of d_2). As negative traces must remain negative in a refinement, this means that $d_1 \text{ alt } d_2$ must include the negative traces of both d_1 and of d_2 , as in equation 14.2 above.

14.2.4 Simple Example

We now give a simple example of underspecification and refinement. Figure 14.1 specifies the game of tic-tac-toe between a player and the system. Either the player or the system may make the first move, and this is specified using `alt`. The player and the system then alternate making moves until the game is over. The `opt` operator is a shorthand for an `alt` with an empty second operand, while `loop(2,3)` may be interpreted as an `alt` between performing the contents of the loop two and three times. For formal definitions of `opt` and `loop`, see [RHS05c]. The game is finished after minimum five and maximum eight moves, depending on how many times the loop is executed, and whether the move inside `opt` is performed or not. (A ninth move is never really necessary, as the result of the game will be given at latest after the eighth move.) We have omitted the details describing the exact positions taken in each move.

In TicTacToe, the choice of who gets the first move is an example of underspecification. A possible refinement could be to use narrowing in order to remove this underspecification, as in TicTacToe2 where the player always moves first:

$$\text{TicTacToe2} = (\text{playerFirst}) \text{ alt } (\text{refuse systemFirst})$$

where the operator `refuse` intuitively means that all traces defined by its argument should be considered negative. (For a formal definition of `refuse`, see Section 14.3.2.) A further refinement could be to add behavior to the specification by e.g. defining that traces where the system makes a second move before the player gets to do his/her move, are negative. These behaviors were inconclusive in TicTacToe2 (and TicTacToe), making this an example of supplementing.

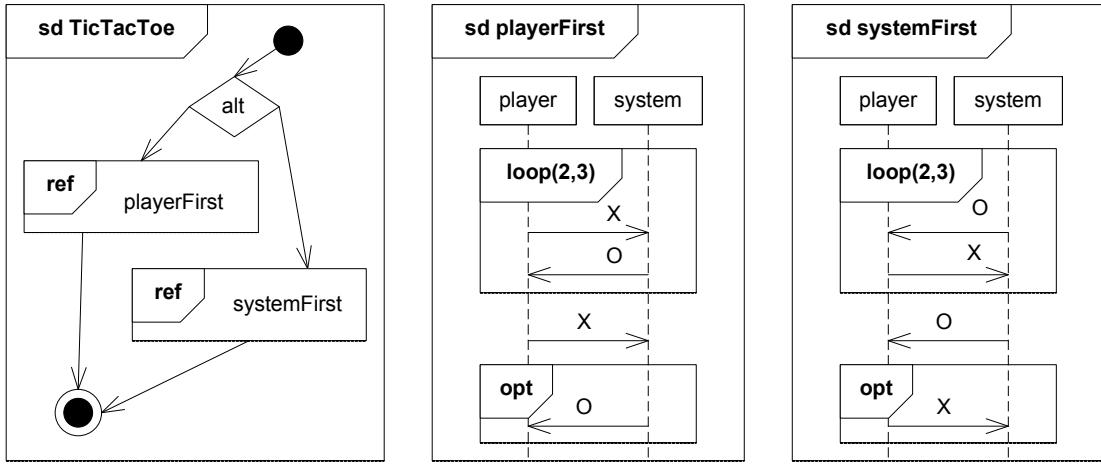


Figure 14.1: Playing tic-tac-toe

14.2.5 Properties of alt and Refinement

As can be expected, the operator **alt** is

- associative: $d_1 \text{ alt } (d_2 \text{ alt } d_3) = (d_1 \text{ alt } d_2) \text{ alt } d_3$
- commutative: $d_1 \text{ alt } d_2 = d_2 \text{ alt } d_1$

This follows trivially from the associativity and commutativity of \cup .

As proved in [HHR_{05a}], we also have that the refinement operator \rightsquigarrow is

- reflexive: $(p, n) \rightsquigarrow (p, n)$
- transitive: $(p, n) \rightsquigarrow (p', n') \wedge (p', n') \rightsquigarrow (p'', n'') \Rightarrow (p, n) \rightsquigarrow (p'', n'')$
- monotonic with respect to **alt**:
 $\llbracket d_1 \rrbracket \rightsquigarrow \llbracket d'_1 \rrbracket \wedge \llbracket d_2 \rrbracket \rightsquigarrow \llbracket d'_2 \rrbracket \Rightarrow \llbracket d_1 \text{ alt } d_2 \rrbracket \rightsquigarrow \llbracket d'_1 \text{ alt } d'_2 \rrbracket$

14.3 Inherent Nondeterminism

14.3.1 Motivation

Underspecification gives rise to nondeterminism, as the system behavior is not completely determined by the specification. Still, nondeterminism in the sense of underspecification does not require that the implementation itself should be nondeterministic. Sometimes, however, it is desirable to specify nondeterminism that *must* be present also in the implementation. We call this *inherent nondeterminism*. The throwing of a dice is an example of a process we would specify as inherently nondeterministic. Another example is a password generator, that should select passwords nondeterministically, at least from the perspective of the user (and the attacker). Inherent nondeterminism is in fact also essential in the domain of (information) security, see [Ros95].

As inherent nondeterminism and underspecification impose different requirements on an implementation, they should be described differently both in the syntax and the semantics of interactions. In STAIRS, inherent nondeterminism is specified by the use of the operator `xalt`. The `xalt` operator is also called mandatory choice, as the implementation must be able to perform (i.e. choose) any one of the given alternatives.

14.3.2 Semantic Representation

In Section 14.2.2 we represented the semantics of a STAIRS specification with underspecification as an interaction obligation (p, n) . With this simple semantics, it is not possible to express cases where *all* alternatives need to be present in an implementation, as traces could be moved from positive to negative by means of refinement. For STAIRS specifications with both underspecification and inherent nondeterminism, we therefore extend the semantics to be a *set* of interaction obligations. The interpretation is that for each interaction obligation (p_i, n_i) a valid implementation needs to be able to produce at least one trace allowed by (p_i, n_i) . Intuitively, each interaction obligation (p_i, n_i) defines an inherently nondeterministic alternative that needs to be implemented, but exactly how this should be achieved is underspecified, since $\mathcal{H} \setminus n_i$ is a set. This leads us to the following formal definition of `xalt`:

$$\llbracket d_1 \text{xalt} d_2 \rrbracket \stackrel{\text{def}}{=} \llbracket d_1 \rrbracket \cup \llbracket d_2 \rrbracket \quad (14.4)$$

We now define the operator `refuse`, informally explained in Section 14.2.4, and generalize the definition of `alt` to cover operands with several interaction obligations:

$$\llbracket \text{refuse } d \rrbracket \stackrel{\text{def}}{=} \{(\emptyset, p \cup n) \mid (p, n) \in \llbracket d \rrbracket\} \quad (14.5)$$

$$\llbracket d_1 \text{ alt } d_2 \rrbracket \stackrel{\text{def}}{=} \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in \llbracket d_1 \rrbracket \wedge (p_2, n_2) \in \llbracket d_2 \rrbracket\} \quad (14.6)$$

14.3.3 Refinement Revisited

The whole point of inherent nondeterminism in a specification is to ensure that the alternatives are preserved during refinement. Since each interaction obligation represents an inherently nondeterministic alternative, we need to ensure that each interaction obligation from the abstract specification will be represented also in the more concrete specification. Formally, a specification d' is a refinement of a specification d , written $d \rightsquigarrow d'$, if

$$\forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow o' \quad (14.7)$$

where $o \rightsquigarrow o'$ is refinement of interaction obligations as given by definition 14.3.

14.3.4 Simple Example

As an example, we consider a so-called “randomizer” that should provide nondeterministic output selected randomly. Figure 14.2 gives a specification where the randomizer simulates the flipping of a coin, where both heads and tails should be possible outcomes.

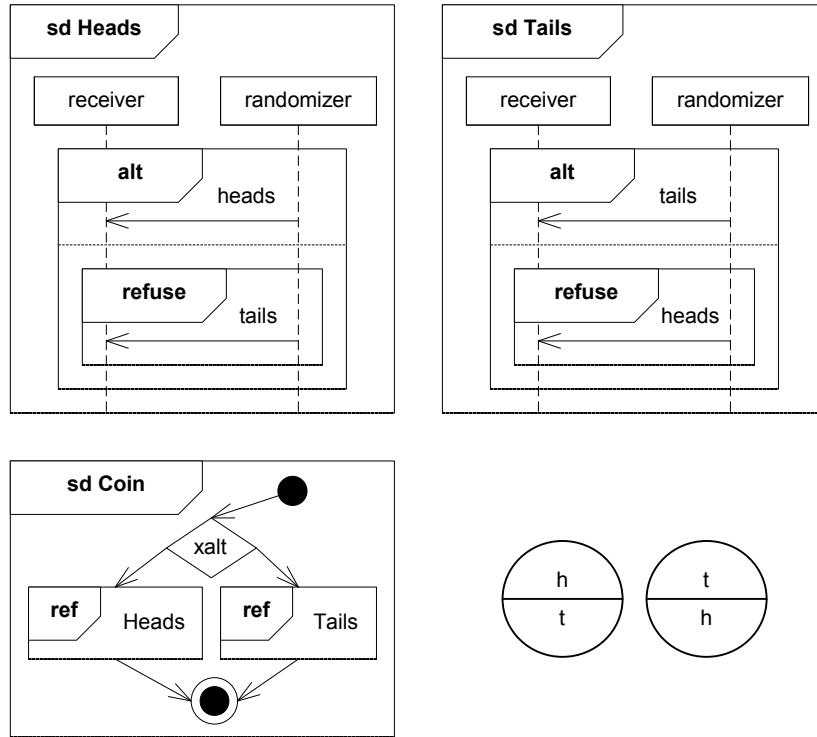


Figure 14.2: The coin specification. Semantic representation to the bottom right.

Textually, we may write the Coin specification and its semantics as:

$$\begin{aligned} \text{Coin} &= (\text{heads alt } (\text{refuse tails})) \text{xalt } (\text{tails alt } (\text{refuse heads})) \\ [\![\text{Coin}]\!] &= \{ (\{h\}, \{t\}), (\{t\}, \{h\}) \} \end{aligned}$$

where h denotes the trace(s) where the outcome is heads and t denotes the trace(s) where the outcome is tails. This semantics is illustrated in the bottom right of Figure 14.2, where each circle represents an interaction obligation with the positive traces in the upper half and the negative traces in the lower half.

As another example, we specify how throwing a dice may simulate the flipping of a coin. One way of doing this is to let odd numbers represent heads, and even numbers represent tails. This is expressed by the specification

$$\text{DiceCoin} = \text{Throw135 xalt Throw246}$$

where Throw135 specifies a throw resulting in an odd number and Throw246 specifies a throw resulting in an even number:

$$\begin{aligned} \text{Throw135} &= (1 \text{ alt } 3 \text{ alt } 5) \text{ alt } (\text{refuse } (2 \text{ alt } 4 \text{ alt } 6)) \\ \text{Throw246} &= (2 \text{ alt } 4 \text{ alt } 6) \text{ alt } (\text{refuse } (1 \text{ alt } 3 \text{ alt } 5)) \end{aligned}$$

Using the given definitions of alt and xalt, we thereby get:

$$[\![\text{DiceCoin}]\!] = \{ (\{1, 3, 5\}, \{2, 4, 6\}), (\{2, 4, 6\}, \{1, 3, 5\}) \}$$

As should be expected, this semantics tells us that when using a dice to simulate a coin, the dice should at least be able to produce one of the numbers 1, 3 and 5 (representing heads) and one of the numbers 2, 4 and 6. However, it is not significant that all numbers may be produced, and DiceCoin may be implemented by the unfair dice DiceCoin2 giving only the numbers 1 and 6:

$$\llbracket \text{DiceCoin2} \rrbracket = \{ (\{1\}, \{2, 3, 4, 5, 6\}), (\{6\}, \{1, 2, 3, 4, 5\}) \}$$

We see that DiceCoin2 is a valid refinement of DiceCoin, as each obligation of DiceCoin is refined into an obligation of DiceCoin2 where some of the positive behaviors have been redefined as negative (i.e. narrowed).

14.3.5 Relating xalt to alt

It is interesting to investigate what kinds of specifications we get by combining the operators for underspecification (i.e. alt) and inherent nondeterminism (i.e. xalt). We have already seen examples of alt within xalt in DiceCoin and DiceCoin2 in the previous section. It remains to investigate the use of xalt within one or both of the operands of alt.

A possible refinement of the Coin specification in Figure 14.2, is to strengthen the specification by stating that the coin should never land on the side. As landing on the side is negative both in the heads and the tails alternative, this behavior may be added by using alt as the top-level operator as in Coin2:

$$\begin{aligned} \text{Coin2} &= \text{Coin alt (refuse side)} \\ \llbracket \text{Coin2} \rrbracket &= \{ (\{h\}, \{t, s\}), (\{t\}, \{h, s\}) \} \end{aligned}$$

where s denotes the trace(s) where the coin lands on the side. As the example demonstrates, alt may in general be used to add (i.e. supplement) the same positive and/or negative traces to *all* interaction obligations specified by xalt.

It remains to consider the case where we have xalt in both operands of alt. Consider again the flipping of a coin as given in Figure 14.2. Another specification where the randomizer simulates the rolling of a three-sided dice is given by:

$$\begin{aligned} \text{Dice} &= (1 \text{ alt (refuse (2 alt 3))) xalt (2 alt (refuse (1 alt 3))) xalt} \\ &\quad (3 \text{ alt (refuse (1 alt 2)))}) \end{aligned}$$

In Figure 14.3 the specifications Coin and Dice are merged by the alt operator. Observe that Coin/Dice is a refinement of both the Coin and the Dice specifications. Each interaction obligation defined by Coin has three refined obligations in Coin/Dice (one would have been sufficient), as the earlier inconclusive traces related to Dice have been supplemented as positive or negative. Similarly, each of the three interaction obligations defined by Dice is refined by two interaction obligations in Coin/Dice. In this sense we may say that the specification of Coin/Dice represents both the Coin and the Dice specifications.

On the other hand, neither Coin nor Dice are valid refinements of Coin/Dice, since the traces 1, 2, 3 are inconclusive in the interaction obligations of Coin and the traces h and t are inconclusive in the interaction obligations of Dice. However, the specifications (Coin alt (refuse Dice)) and ((refuse Coin) alt Dice) are both valid refinements of Coin/Dice, since these specifications ensure that none of the traces from the Coin/Dice specification are inconclusive. Intuitively,

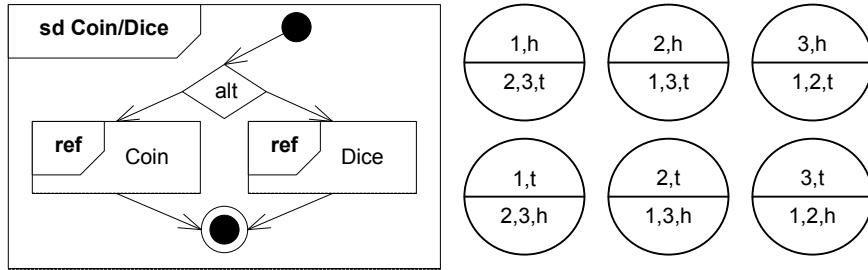


Figure 14.3: The Coin and Dice specifications combined by alt. Semantic representation to the right.

these specifications mean that traces from the Dice (or Coin) alternative should not be produced, which means that the opposite alternative is chosen. In general, for any specifications d_1 and d_2 the set of valid refinements (and therefore implementations) of $d_1 \text{ alt } d_2$ will be both a subset of the valid refinements of d_1 and a subset of the valid refinements of d_2 .

A valid refinement of the specification in Figure 14.3 would be to move the trace h to the negative sets in all interaction obligations, without doing the same with the trace t . The possible outcomes of a single run would then be 1, 2, 3 or t – so we know that if a coin trace is produced, it will be t (assuming 1, 2, 3, h and t are the only relevant traces).

The alt operator should be interpreted as underspecification w.r.t. traces and not w.r.t. interaction obligations. As demonstrated by the examples in this section it is not sufficient for an implementation to consider only one of the alt operands. In general, the alt characterizes the intersection of its operands, meaning that $d_1 \text{ alt } d_2$ is a refinement of both d_1 and d_2 . If we restrict refinement to narrowing, using alt between two specifications with xalt may be interpreted as “the implementation must include the inherent nondeterminism specified by at least one of the alternatives”.

14.3.6 Properties of xalt and Refinement

As for alt, xalt is

- associative: $d_1 \text{ xalt } (d_2 \text{ xalt } d_3) = (d_1 \text{ xalt } d_2) \text{ xalt } d_3$
- commutative: $d_1 \text{ xalt } d_2 = d_2 \text{ xalt } d_1$

This follows trivially from the associativity and commutativity of \cup .

With respect to xalt, alt is

- right distributive: $(d_1 \text{ xalt } d_2) \text{ alt } d_3 = (d_1 \text{ alt } d_3) \text{ xalt } (d_2 \text{ alt } d_3)$
- left distributive: $d_1 \text{ alt } (d_2 \text{ xalt } d_3) = (d_1 \text{ alt } d_2) \text{ xalt } (d_1 \text{ alt } d_3)$

meaning that a specification with arbitrary nesting of alt and xalt may always be rewritten as a specification with xalt as the top-level operator. This is proved by theorems 1 and 2 in Appendix 14.A.

As in the simple case, we have that the refinement operator \rightsquigarrow is:

- reflexive: $d \rightsquigarrow d$
- transitive: $d \rightsquigarrow d' \wedge d' \rightsquigarrow d'' \Rightarrow d \rightsquigarrow d''$
- monotonic with respect to alt and xalt:

$$d_1 \rightsquigarrow d'_1 \wedge d_2 \rightsquigarrow d'_2 \Rightarrow d_1 \text{ alt } d_2 \rightsquigarrow d'_1 \text{ alt } d'_2 \wedge d_1 \text{ xalt } d_2 \rightsquigarrow d'_1 \text{ xalt } d'_2$$

These results are proved in [HHR05a].

14.4 Probability

14.4.1 Motivation

Being able to specify probabilities add useful expressiveness to the specifications. One typical example is in the specification of a coin or a dice, where the alternatives must occur with the same probability. Another example is a gambling machine, where the winning alternatives should occur, but less often than the losing ones.

Interactions are mainly used for specifying communication scenarios. Probabilities are equally relevant in this setting, for instance to specify the probability that a message will never be received when sent over an unreliable communication channel. Another example is when specifying soft real-time constraints such as “the user of the system will receive an answer within 10 seconds at least 90% of the time” (for more details, see [RHS05a]). As this example demonstrates, we are not only interested in assigning exact probabilities to all alternatives specified by an xalt, but also to specify a possible range for the probabilities, i.e. to allow underspecification with respect to probabilities as well as behaviors. In STAIRS, this is achieved by generalizing xalt to the palt operator.

14.4.2 Semantic Representation

Semantically, a probabilistic STAIRS specification is represented by a set of *probability obligations* (also called *p-obligations*). A p-obligation $((p, n), Q)$ consists of an interaction obligation (p, n) and a set of probabilities Q . In any valid implementation the p-obligation $((p, n), Q)$ should be selected with a probability in Q . The fact that Q is a set and not a single probability allows us to represent underspecification w.r.t. probabilities.

If a specification includes the p-obligation $((\{t_1, t_2\}, \mathcal{H} \setminus \{t_1, t_2\}), \{0.6\})$, this does not necessarily mean that the probability of getting either t_1 or t_2 is 0.6; it may be greater if there is another p-obligation $((p, n), Q)$ such that $\{t_1, t_2\} \not\subseteq n$. On the other hand, if a specification contains a p-obligation $((p, n), \{0.6\})$ such that $\{t_3, t_4\} \subseteq n$, then we know that the probability of getting a trace in $\{t_3, t_4\}$ is at most 0.4.

The palt construct expresses probabilistic choice. Use of the palt operator is the only way to assign probabilities different from 1. Before defining the semantics of the palt, we introduce the notion of probability decoration, used to specify the probabilities associated with the operands of a palt. It is defined by

$$\llbracket d; Q' \rrbracket \stackrel{\text{def}}{=} \{(o, Q * Q') \mid (o, Q) \in \llbracket d \rrbracket\} \quad (14.8)$$

where multiplication of probability sets is defined by

$$Q_1 * Q_2 \stackrel{\text{def}}{=} \{q_1 * q_2 \mid q_1 \in Q_1 \wedge q_2 \in Q_2\} \quad (14.9)$$

We also define the summation of n probability sets:

$$\sum_{i=1}^n Q_i \stackrel{\text{def}}{=} \{\min(\sum_{i=1}^n q_i, 1) \mid \forall i \leq n : q_i \in Q_i\} \quad (14.10)$$

The palt operator describes the probabilistic choice between two or more alternative operands whose joint probability should add up to one. Formally, the palt is defined by

$$[\![\text{palt}(d_1; Q_1, \dots, d_n; Q_n)]\!] \stackrel{\text{def}}{=} \quad (14.11)$$

$$\begin{aligned} & \{(\oplus \bigcup_{i \in N} \{po_i\}, \sum_{i \in N} \pi_{2,po_i}) \mid \\ & \quad N \subseteq \{1, \dots, n\} \wedge N \neq \emptyset \wedge \forall i \in N : po_i \in [\![d_i; Q_i]\!] \} \end{aligned} \quad (\text{a})$$

$$\cup \{(\oplus \bigcup_{i=1}^n [\![d_i; Q_i]\!], \{1\} \cap \sum_{i=1}^n Q_i)\} \quad (\text{b})$$

where $\pi_{2,po}$ returns the probability set of the p-obligation po and \oplus is an operator for combining the interaction obligations of a set S of p-obligations into a single interaction obligation, defined as

$$\oplus S \stackrel{\text{def}}{=} ((\bigcup_{((p,n),Q) \in S} p) \cap (\bigcap_{((p,n),Q) \in S} p \cup n), \bigcap_{((p,n),Q) \in S} n) \quad (14.12)$$

We now explain definition 14.11 in detail. We first look at 14.11a. If we restricted each N to be a singleton set then this part of the definition could be written equivalently as $\bigcup_{i=1}^n [\![d_i; Q_i]\!]$. This would correspond to the definition of xalt and means simply that each probabilistic alternative should be reflected in a valid implementation.

By including also the cases where N is any non-empty subset of $\{1, \dots, n\}$ we are able to define the semantics as a set of p-obligations instead of as a multiset. The operator \oplus characterizes the traces allowed by all the p-obligations in its argument set: A trace t is positive if it is positive according to at least one p-obligation and not inconclusive according to any; t is negative only if it is negative according to all p-obligations; traces that are inconclusive according to at least one p-obligation remain inconclusive. So if a p-obligation $((p, n), Q)$ occurs for example in two operands of the palt, then the resulting semantics will contain a p-obligation $((p, n), Q + Q)$.

The single p-obligation in 14.11b requires the probabilities of the operands to add up to one. If it is impossible to choose one probability from each Q_i so that the sum is 1, then the probability set will be empty and the specification is not implementable.

We also redefine the refuse and alt operators to take probabilities into account. Redefining positive traces as negative does not influence probabilities, so refuse is defined simply by

$$[\![\text{refuse } d]\!] \stackrel{\text{def}}{=} \{((\emptyset, p \cup n), Q) \mid ((p, n), Q) \in [\![d]\!] \} \quad (14.13)$$

The `alt` construct captures underspecification with respect to traces. Two sets of p-obligations are combined by taking the pairwise combination of p-obligations from each set. As before, interaction obligations are combined by taking the union of the positive traces and the union of the negative traces. In Section 14.3.5 we showed that the resulting interaction obligation is a refinement of both the original ones, and therefore represents both of these interaction obligations. Since the two p-obligations from the different operands are chosen independently from each other, probabilities are multiplied. Formally:

$$\llbracket d_1 \text{ alt } d_2 \rrbracket \stackrel{\text{def}}{=} \{(o_1 \uplus o_2, Q_1 * Q_2) \mid (o_1, Q_1) \in \llbracket d_1 \rrbracket \wedge (o_2, Q_2) \in \llbracket d_2 \rrbracket\} \quad (14.14)$$

14.4.3 Refinement Revisited

A p-obligation is refined by either refining its interaction obligation, or by reducing its set of probabilities. Formally, a p-obligation $((p', n'), Q')$ is a refinement of a p-obligation $((p, n), Q)$, written $((p, n), Q) \rightsquigarrow ((p', n'), Q')$, if

$$(p, n) \rightsquigarrow (p', n') \wedge Q' \subseteq Q \quad (14.15)$$

All abstract p-obligations must be represented by a p-obligation also at the refined level, unless it has 0 as an acceptable probability, which means that it does not need to be implemented. Formally, a specification d' is a refinement of a specification d , written $d \rightsquigarrow d'$, if

$$\forall po \in \llbracket d \rrbracket : (0 \notin \pi_2.po \Rightarrow \exists po' \in \llbracket d' \rrbracket : po \rightsquigarrow po') \quad (14.16)$$

We now explain further why also the cases where N is any non-singular subset of $\{1, \dots, n\}$ is included in definition 14.11a. Firstly, we want to avoid a situation where two p-obligations (o_1, Q_1) and (o_2, Q_2) coming from different operands of a `palt` are represented *only* by a single p-obligation (o, Q) that is a refinement of both (o_1, Q_1) and (o_2, Q_2) at the concrete level. We avoid this since also the p-obligation $(\oplus\{(o_1, Q_1), (o_2, Q_2)\}, Q_1 + Q_2)$ is included in the semantics and hence needs to be represented at the concrete level.

Secondly, it should be possible to let a single p-obligation at the abstract level be represented by a combination of p-obligations at the concrete level, as long as each of these p-obligations are valid refinements of the original p-obligation w.r.t. interaction obligations and their probability sets add up to a subset of the original probability set. The inclusion of the combined p-obligations (resulting from N sets with more than one element) in the `palt` semantics makes this possible.

Our definition of refinement also explains why we have chosen to assign sets of acceptable probabilities to the operands, and not simply lower bounds. Consider the following specifications:

$$\begin{aligned} d_a &= \text{palt}(d_1; [\frac{1}{5} \dots 1], d_2; [\frac{1}{5} \dots 1], d_3; [\frac{1}{5} \dots 1]) \\ d_b &= \text{palt}(d_1; [\frac{1}{5} \dots \frac{1}{2}], d_2; [\frac{1}{5} \dots \frac{1}{2}], d_3; [\frac{1}{5} \dots \frac{1}{2}]) \\ d_c &= \text{palt}(d_1; \{\frac{1}{5}\}, d_2; \{\frac{1}{5}\}, d_3; \{\frac{3}{5}\}) \end{aligned}$$

Then d_c is a refinement of d_a , but not of d_b . So by using only lower bounds we would have less expressive power.

14.4.4 Simple Example

We now demonstrate a simple refinement in probabilistic STAIRS, building on the DiceCoin/DiceCoin2 example from Section 14.3.4. Let pDiceCoin be a probabilistic version of DiceCoin where the probabilities of odd and even numbers are the same, represented syntactically and semantically by

$$\begin{aligned} \text{pDiceCoin} &= \text{palt}(\text{Throw135};\{\frac{1}{2}\}, \text{Throw246};\{\frac{1}{2}\}) \\ \llbracket \text{pDiceCoin} \rrbracket &= \{ ((\{1, 3, 5\}, \{2, 4, 6\}), \{\frac{1}{2}\}), ((\{2, 4, 6\}, \{1, 3, 5\}), \{\frac{1}{2}\}), \\ &\quad ((\{1, 2, 3, 4, 5, 6\}, \emptyset), \{1\}) \} \end{aligned}$$

The semantic representation tells us that the dice should be able to produce at least one number in $\{1, 3, 5\}$, and the probability for this alternative should be $\frac{1}{2}$. Similarly, the dice should be able to produce at least one number in $\{2, 4, 6\}$, with probability $\frac{1}{2}$. Obviously, the probability of producing a number in $\{1, 2, 3, 4, 5, 6\}$ should then be 1.

Suppose now that we require that the dice should be fair w.r.t. the odd numbers, give equal chances of odd and even number, and not produce any even number different from 6. We first let $\llbracket \text{Throw1} \rrbracket = \{ ((\{1\}, \{2, 3, 4, 5, 6\}), \{1\}) \}$ and similarly for the other numbers. We then refine Throw135 by Throw135Fairly:

$$\begin{aligned} \text{Throw135Fairly} &= \text{palt}(\text{Throw1};\{\frac{1}{3}\}, \text{Throw3};\{\frac{1}{3}\}, \text{Throw5};\{\frac{1}{3}\}) \\ \llbracket \text{Throw135Fairly} \rrbracket &= \{ ((\{1\}, \{2, 3, 4, 5, 6\}), \{\frac{1}{3}\}), \\ &\quad ((\{3\}, \{1, 2, 4, 5, 6\}), \{\frac{1}{3}\}), ((\{5\}, \{1, 2, 3, 4, 6\}), \{\frac{1}{3}\}), \\ &\quad ((\{1, 3\}, \{2, 4, 5, 6\}), \{\frac{2}{3}\}), ((\{1, 5\}, \{2, 3, 4, 6\}), \{\frac{2}{3}\}), \\ &\quad ((\{3, 5\}, \{1, 2, 4, 6\}), \{\frac{2}{3}\}), ((\{1, 3, 5\}, \{2, 4, 6\}), \{1\}) \} \end{aligned}$$

As Throw135 has the semantics $\{((\{1, 3, 5\}, \{2, 4, 6\}), \{1\})\}$, we see that this is indeed a valid refinement, since the only p-obligation in $\llbracket \text{Throw135} \rrbracket$ is identical to one of the p-obligations in $\llbracket \text{Throw135Fairly} \rrbracket$. A dice that is fair w.r.t. the odd numbers, has equal chances of odd and even numbers, and does not produce any even number different from 6 can now be expressed by

$$\begin{aligned} \text{pDiceCoin2} &= \text{palt}(\text{Throw135Fairly};\{\frac{1}{2}\}, \text{Throw6};\{\frac{1}{2}\}) \\ \llbracket \text{pDiceCoin2} \rrbracket &= \{ ((\{1\}, \{2, 3, 4, 5, 6\}), \{\frac{1}{6}\}), \\ &\quad ((\{3\}, \{1, 2, 4, 5, 6\}), \{\frac{1}{6}\}), ((\{5\}, \{1, 2, 3, 4, 6\}), \{\frac{1}{6}\}), \\ &\quad ((\{1, 3\}, \{2, 4, 5, 6\}), \{\frac{1}{3}\}), ((\{1, 5\}, \{2, 3, 4, 6\}), \{\frac{1}{3}\}), \\ &\quad ((\{3, 5\}, \{1, 2, 4, 6\}), \{\frac{1}{3}\}), ((\{1, 6\}, \{2, 3, 4, 5\}), \{\frac{2}{3}\}), \\ &\quad ((\{3, 6\}, \{1, 2, 4, 5\}), \{\frac{2}{3}\}), ((\{5, 6\}, \{1, 2, 3, 4\}), \{\frac{2}{3}\}), \\ &\quad ((\{1, 3, 6\}, \{2, 4, 5\}), \{\frac{5}{6}\}), ((\{1, 5, 6\}, \{2, 3, 4\}), \{\frac{5}{6}\}), \\ &\quad ((\{3, 5, 6\}, \{1, 2, 4\}), \{\frac{5}{6}\}), ((\{1, 3, 5\}, \{2, 4, 6\}), \{\frac{1}{2}\}), \\ &\quad ((\{6\}, \{1, 2, 3, 4, 5\}), \{\frac{1}{2}\}), ((\{1, 3, 5, 6\}, \{2, 4\}), \{1\}) \} \end{aligned}$$

Each p-obligation in $\llbracket \text{pDiceCoin} \rrbracket$ has a refining p-obligation in $\llbracket \text{pDiceCoin2} \rrbracket$, so $\text{pDiceCoin} \rightsquigarrow \text{pDiceCoin2}$ holds.

14.4.5 Relating palt to xalt and alt

In STAIRS, every xalt-operand represents an alternative that must be reflected in the implementation. Its probability should be greater than 0, but is otherwise unknown. In probabilistic

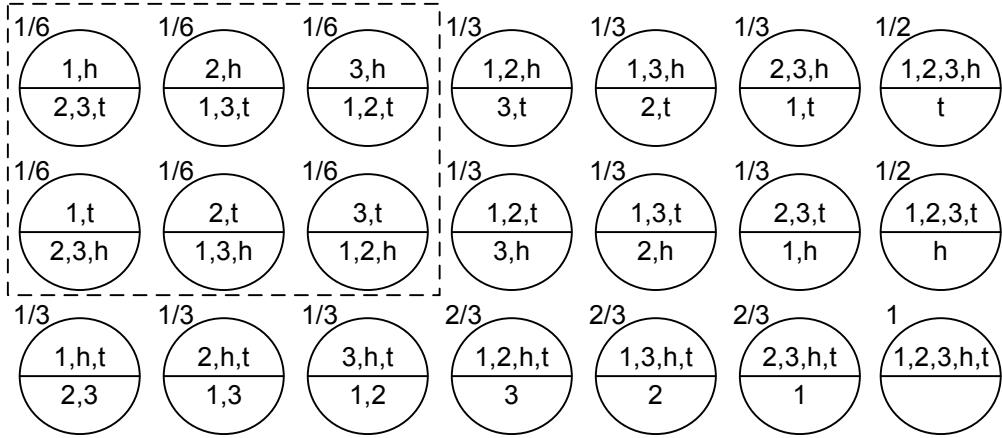


Figure 14.4: The semantics of (pCoin alt pDice) in probabilistic STAIRS.

STAIRS, a specification $\text{xalt}(d_1, \dots, d_n)$ is therefore interpreted as $\text{palt}(d_1; Q, \dots, d_n; Q)$ where $Q = \langle 0, \dots, 1 \rangle$.

We now discuss what it means to have probabilistic STAIRS specifications that combine the use of the alt and palt operators. We hope the meaning of underspecification within probabilistic alternative is intuitively clear, and do not go further into this. Instead we show a probabilistic version of the previous examples. pCoin specifies a coin, while pDice specifies a 3-sided dice:

$$\begin{aligned}
 \text{pCoin} &= \text{palt}(\text{Heads}; \{\frac{1}{2}\}, \text{Tails}; \{\frac{1}{2}\}) \\
 \text{pDice} &= \text{palt}(\text{One}; \{\frac{1}{3}\}, \text{Two}; \{\frac{1}{3}\}, \text{Three}; \{\frac{1}{3}\}) \\
 [\![\text{pCoin}]\!] &= \{(\{\{h\}, \{t\}\}, \{\frac{1}{2}\}), (\{\{t\}, \{h\}\}, \{\frac{1}{2}\}), (\{\{h, t\}, \emptyset\}, \{1\})\} \\
 [\![\text{pDice}]\!] &= \{(\{\{1\}, \{2, 3\}\}, \{\frac{1}{3}\}), (\{\{2\}, \{1, 3\}\}, \{\frac{1}{3}\}), (\{\{3\}, \{1, 2\}\}, \{\frac{1}{3}\}), \\
 &\quad (\{\{1, 2\}, \{3\}\}, \{\frac{2}{3}\}), (\{\{1, 3\}, \{2\}\}, \{\frac{2}{3}\}), (\{\{2, 3\}, \{1\}\}, \{\frac{2}{3}\}), \\
 &\quad (\{\{1, 2, 3\}, \emptyset\}, \{1\})\}
 \end{aligned}$$

These examples use only a single probability in each probability set (there is no underspecification w.r.t. probabilities). Figure 14.4 shows the semantics of

$$\text{pCoin/Dice} = \text{pCoin alt pDice}$$

We see that the interaction obligation of each p-obligation in pCoin/Dice refines the interaction obligation of a p-obligation for both pCoin and pDice. For example, the interaction obligation of the leftmost, uppermost p-obligation in Figure 14.4 represent the first p-obligation of both $[\![\text{pCoin}]\!]$ and $[\![\text{pDice}]\!]$. Since these represent two independent probabilistic choices it is reasonable to multiply their probabilities. This also gives the nice result that if we consider only “pure” p-obligations (those we get from definition 14.11a in the cases where N is a singleton set), then their probabilities add up to 1. In Figure 14.4 these “pure” p-obligations are enclosed by the dotted line.

14.4.6 Properties of alt, palt and Refinement

For alt, the revised definition 14.14 is still associative and commutative.

In contrast to `xalt`, `palt` is *not* associative. The order in which obligations are combined according to 14.11b is significant, since this determines which probabilities must add up to 1. Remember that the requirement that probabilities for the operands add up to 1 applies to each occurrence of a `palt` operator, independently of the nesting level. For similar reasons, `alt` is not distributive with respect to `palt`. Consider the following specifications:

$$\begin{aligned} d_a &= (\text{palt}(d_1; Q_1, d_2; Q_2)) \text{ alt } (\text{palt}(d_3; Q_3, d_4; Q_4)) \\ d_b &= \text{palt}((\text{palt}(d_1; Q_1, d_2; Q_2) \text{ alt } d_3); Q_3, (\text{palt}(d_1; Q_1, d_2; Q_2) \text{ alt } d_4); Q_4) \end{aligned}$$

In d_b we are free to choose different probabilities from the sets Q_1 and Q_2 in the two operands of the outermost `palt`. In d_a there is no such freedom, so in this respect d_a is more restrictive than d_b .

However, we do have commutativity of `palt`:

$$\forall i, j \in [1, n] : \text{palt}(\dots, d_i; Q_i, \dots, d_j; Q_j, \dots) = \text{palt}(\dots, d_j; Q_j, \dots, d_i; Q_i, \dots)$$

This follows trivially from the commutativity of \cup .

For probabilistic STAIRS, the refinement operator \rightsquigarrow is:

- reflexive, transitive, and monotonic with respect to `alt`
- restricted monotonic with respect to `palt`:
 $(\forall i \in [1 : n] : d_i \rightsquigarrow d'_i \wedge Q'_i \subseteq Q_i \wedge \bigoplus \llbracket d_i \rrbracket \rightsquigarrow \bigoplus \llbracket d'_i \rrbracket) \Rightarrow \text{palt}(d_1; Q_1, \dots, d_n; Q_n) \rightsquigarrow \text{palt}(d'_1; Q'_1, \dots, d'_n; Q'_n)$

This is proved in [RHS05b], which also motivates the last requirement in the monotonicity for `palt`.

The interpretation given for `xalt` in probabilistic STAIRS is reasonable, as `xalt`(d_1, \dots, d_n) and `palt`($d_1; \langle 0 \dots 1 \rangle, \dots, d_n; \langle 0 \dots 1 \rangle$) are refinements of each other when abstracting away the probabilities. This is proved by theorem 3 in Appendix 14.A.

14.5 Related Work

Most specification languages do not distinguish between underspecification and inherent nondeterminism the way it is done in STAIRS. The most well known dialects of interactions are UML [OMG04] and MSC [ITU99]. Neither of these have two different operators corresponding to `alt` and `xalt`. In practice, the `alt` operator of UML is probably used by different groups to describe both inherent nondeterminism and underspecification.

Live Sequence Charts [DH01] and [HM03] is a dialect of MSC where a (part) of an interaction may be designated as either universal (mandatory) or existential (optional). Explicit criteria in the form of precharts decide when the chart applies; whenever the communication behavior described by the prechart occurs, behavior described by the chart *must* follow (in the case of universal locations) or *may* follow (in the case of existential locations). Universal charts specify all allowed traces. This is therefore not the same as inherently nondeterministic alternatives in STAIRS, since the latter only specifies some of the traces that must be present in an implementation.

CSP [Hoa85] defines two different operators for nondeterministic choice. Their difference, however, is explained in terms of internal versus external choice. This is not the same distinction as the one between underspecification and inherent nondeterminism. As an example, let $?$ denote an input event, $!$ denote an output event, and seq be the operator for sequential composition in the STAIRS specification $(?a \text{ seq } (!b \text{ xalt } !c)) \text{ alt } ((?b \text{ seq } !d))$. Here, the environment may choose between the two alt-operands, corresponding to external choice in CSP. However, the choice between $!b$ and $!c$ should be inherently nondeterministic, a requirement that may not be expressed using the CSP operators, while replacing xalt with alt , would correspond to internal choice in CSP.

[SBDB97] extends the process algebraic language LOTOS [ISO89] with a disjunction operator for specifying implementation freedom (i.e. underspecification), leaving the LOTOS choice operator to be used for inherent nondeterminism. The disjunction operator is similar to our alt operator, and the choice operator corresponds to xalt . An important difference between disjunction and alt is that an implementation will have to select exactly one of the disjunction operands, while it may include several of the traces specified by alt .

Probabilistic automata [Seg95] includes both nondeterminism and probabilistic choice. Underspecification with respect to probabilities is represented by nondeterministic choices between distributions. As for automata in general, specifications are complete in the sense that there is no notion of inconclusive behavior.

In [MM99] a probabilistic extension of Dijkstra's Guarded Command Language *GCL* [Dij76] called *pGCL* is presented. The language includes both an operator \sqcap for “demonic” choice and an operator $p\oplus$ for probabilistic choice. The following intuitive explanation is given for the meaning of the construct *this* \sqcap *that*: “The customer will be happy with either *this* or *that*; and the implementer may choose between them according to his own concerns.” This indicates that the role of the \sqcap operator in a *pGCL* specification is to express underspecification, similar to the role of the alt operator in (probabilistic) STAIRS. By specifying probabilistic choices the role of the $p\oplus$ operator in *pGCL* corresponds to the role of palt in probabilistic STAIRS. There is no notion of inconclusive behavior in *pGCL*.

[Heh04] shows how probabilistic reasoning can be applied to predicative programs and specifications. Nondeterminism is disjunction, and equivalent to a deterministic choice in which the determining expression is a variable of unknown value (probability). Nondeterminism gives freedom to the implementer; it can be refined by a deterministic or a probabilistic choice. Since the implementer is not forced to produce both alternatives, the nondeterminism in [Heh04] corresponds to underspecification in STAIRS. Cases where both alternatives need to be possible are expressed by a probabilistic choice, as in probabilistic STAIRS.

14.6 Conclusion

This article has shown the need for underspecification, inherent nondeterminism and probability in specifications. We have demonstrated that these phenomena are adequately expressed in STAIRS and probabilistic STAIRS by the operators alt , xalt and palt . New insight has been gained into the interplay between these operators through studies of simple examples. The focus of this paper has been on the theoretical understanding of how underspecification and inherent nondeterminism is expressed in specifications and represented semantically. The simplicity of

the specifications has allowed us to properly explain their semantic representations. For more examples related to communication see [HHR05b], [RHS05c] and [RHS05a]. We firmly believe that STAIRS and probabilistic STAIRS offer a suitable expressiveness for practical specifications, and intend to show this in the future through studies of real-life specifications.

Acknowledgements. The research on which this paper reports has been partly carried out within the context of the IKT-2010 project SARDAS (15295/431) and the IKT SOS project ENFORCE (164382/V30), both funded by the Research Council of Norway. We thank Roberto Segala and the other members of the SARDAS project for useful discussions related to this work. We also thank the anonymous reviewers for constructive feedback.

References

- [DH01] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.
- [Dij76] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
- [Heh04] E. C. R. Hehner. Probabilistic predicative programming. In Dexter Kozen and Caron Shankland, editors, *Mathematics of Program Construction, 7th International Conference*, number 3125 in Lecture Notes in Computer Science, pages 169–185. Springer, 2004.
- [HHR05a] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309, Department of Informatics, University of Oslo, 2005.
- [HHR05b] Ø. Haugen, K.E. Husa, R.K. Runde, and K. Stølen. STAIRS towards formal design with sequence diagrams. *Software and System Modeling*, 4(4):349–458, 2005.
- [HM03] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer, 2003.
- [Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [ISO89] International Standards Organization. *Information Processing Systems – Open Systems Interconnection - Lotos – a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour – ISO 8807*, 1989.
- [ITU99] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1999.
- [MM99] C. Morgan and A. McIver. pGCL: Formal reasoning for random algorithms. *South African Computer Journal*, 22:14–27, 1999.
- [OMG04] Object Management Group. *UML 2.0 Superstructure Specification*, ptc/04-10-02 edition, 2004.
- [RHS05a] A. Refsdal, K. E. Husa, and K. Stølen. Specification and refinement of soft real-time requirements using sequence diagrams. In P. Pettersson and W. Yi, editors, *Proc. Formal Modeling and Analysis of Timed Systems: Third International Conference, FORMATS, 2005*, number 3829 in Lecture Notes in Computer Science, pages 32–48. Springer, 2005.
- [RHS05b] A. Refsdal, K. E. Husa, and K. Stølen. Specification and refinement of soft real-time requirements using sequence diagrams. Technical Report 323, Department of Informatics, University of Oslo, 2005.
- [RHS05c] R.K. Runde, Ø. Haugen, and K. Stølen. Refining UML interactions with underspecification and nondeterminism. *Nordic Journal of Computing*, 12(2):157–188, 2005.

- [Ros95] A. W. Roscoe. CSP and determinism in security modelling. In *Proc. IEEE Symposium on Security and Privacy*, pages 114–127. IEEE Press, 1995.
- [SBDB97] M.W.A. Steen, H. Bowman, J. Derrick, and E.A. Boiten. Disjunction of LOTOS specifications. In T. Mizuno, N. Shiratori, T. Higashino, and A. Togashi, editors, *Formal Description Techniques and Protocol Specification, Testing and Verification: FORTE X / PSTV XVII '97*, pages 177–192. Chapman & Hall, 1997.
- [Seg95] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.

14.A Proofs

First, we introduce some additional notation:

- $\llbracket d \rrbracket$ is the semantic function in ordinary STAIRS, as defined in Section 14.3.
- $\llbracket d \rrbracket^p$ is the semantics function in probabilistic STAIRS, as defined in Section 14.4.

For the proofs, we need the formal definitions of the semantics of an interaction consisting of a single event e in STAIRS and probabilistic STAIRS.

$$\llbracket e \rrbracket \stackrel{\text{def}}{=} \{ (\{\langle e \rangle\}, \emptyset) \} \quad (14.17)$$

$$\llbracket e \rrbracket^p \stackrel{\text{def}}{=} \{ ((\{\langle e \rangle\}, \emptyset), \{1\}) \} \quad (14.18)$$

These definitions are taken from [HHR05b] and [RHS05a].

We also rewrite the refinement definition 14.7 to a form more suitable for the proofs. Refinement of interactions is defined by

$$d \rightsquigarrow d' \stackrel{\text{def}}{=} \llbracket d \rrbracket \rightsquigarrow \llbracket d' \rrbracket \quad (14.19)$$

where refinement of sets of interaction obligations is defined by

$$O \rightsquigarrow O' \stackrel{\text{def}}{=} \forall o \in O : \exists o' \in O' : o \rightsquigarrow o' \quad (14.20)$$

It is straightforward to see that definitions 14.19 and 14.20 together are equivalent to the original definition 14.7.

Theorem 1 *Right distributivity of alt over xalt*

$$P : (d_1 \text{xalt } d_2) \text{ alt } d_3 = (d_1 \text{ alt } d_3) \text{xalt } (d_2 \text{ alt } d_3)$$

$$\begin{aligned} P &: \\ &\llbracket (d_1 \text{xalt } d_2) \text{ alt } d_3 \rrbracket \\ &= \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in \llbracket d_1 \text{xalt } d_2 \rrbracket \wedge (p_2, n_2) \in \llbracket d_3 \rrbracket\} \quad \text{by definition 14.6} \\ &= \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in \llbracket d_1 \rrbracket \cup \llbracket d_2 \rrbracket \wedge (p_2, n_2) \in \llbracket d_3 \rrbracket\} \quad \text{by definition 14.4} \\ &= \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in \llbracket d_1 \rrbracket \wedge (p_2, n_2) \in \llbracket d_3 \rrbracket\} \\ &\quad \cup \\ &\quad \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in \llbracket d_2 \rrbracket \wedge (p_2, n_2) \in \llbracket d_3 \rrbracket\} \\ &= \llbracket d_1 \text{ alt } d_3 \rrbracket \cup \llbracket d_2 \text{ alt } d_3 \rrbracket \quad \text{by definition 14.6} \\ &= \llbracket (d_1 \text{ alt } d_3) \text{xalt } (d_2 \text{ alt } d_3) \rrbracket \quad \text{by definition 14.4} \end{aligned}$$

Theorem 2 *Left distributivity of alt over xalt*

$$P : d_1 \text{ alt } (d_2 \text{ xalt } d_3) = (d_1 \text{ alt } d_2) \text{ xalt } (d_1 \text{ alt } d_3)$$

$$P :$$

$$\begin{aligned} & \llbracket d_1 \text{ alt } (d_2 \text{ xalt } d_3) \rrbracket \\ &= \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in \llbracket d_1 \rrbracket \wedge (p_2, n_2) \in \llbracket d_2 \text{ xalt } d_3 \rrbracket\} \quad \text{by definition 14.6} \\ &= \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in \llbracket d_1 \rrbracket \wedge (p_2, n_2) \in \llbracket d_2 \rrbracket \cup \llbracket d_3 \rrbracket\} \quad \text{by definition 14.4} \\ &= \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in \llbracket d_1 \rrbracket \wedge (p_2, n_2) \in \llbracket d_2 \rrbracket\} \\ &\quad \cup \\ &\quad \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in \llbracket d_1 \rrbracket \wedge (p_2, n_2) \in \llbracket d_3 \rrbracket\} \\ &= \llbracket d_1 \text{ alt } d_2 \rrbracket \cup \llbracket d_1 \text{ alt } d_3 \rrbracket \quad \text{by definition 14.6} \\ &= \llbracket (d_1 \text{ alt } d_2) \text{ xalt } (d_1 \text{ alt } d_3) \rrbracket \quad \text{by definition 14.4} \end{aligned}$$

Lemma 1 *Given an arbitrary, non-empty set of p-obligations S, every interaction obligation of a p-obligation in S will be a refinement of $\oplus S$.*

$$P : \forall((p, n), Q) \in S : \oplus S \rightsquigarrow (p, n)$$

P : The negative traces in $\oplus S$ are traces that are negative in *every* p-obligation in S , meaning that they are included in n . Similarly, every positive trace in $\oplus S$ must be either positive or negative in *every* p-obligation in S , meaning that it is included in $p \cup n$.

$\langle 1 \rangle 1.$ Choose arbitrary $((p, n), Q) \in S$

P : S is non-empty.

$\langle 1 \rangle 2.$ Requirement 1: $(\bigcap_{((p_i, n_i), Q_i) \in S} n_i) \subseteq n$

P : $\langle 1 \rangle 1$ and $A \cap B \subseteq B$ for arbitrary sets A and B .

$\langle 1 \rangle 3.$ Requirement 2: $((\bigcup_{((p_i, n_i), Q_i) \in S} p_i) \cap (\bigcap_{((p_i, n_i), Q_i) \in S} p_i \cup n_i)) \subseteq p \cup n$

$\langle 2 \rangle 1.$ $(\bigcap_{((p_i, n_i), Q_i) \in S} p_i \cup n_i) \subseteq p \cup n$

P : $\langle 1 \rangle 1$ and $A \cap (B \cup C) \subseteq (B \cup C)$ for arbitrary sets A , B and C .

$\langle 2 \rangle 2.$ $((\bigcup_{((p_i, n_i), Q_i) \in S} p_i) \cap (\bigcap_{((p_i, n_i), Q_i) \in S} p_i \cup n_i)) \subseteq p \cup n$

P : $\langle 2 \rangle 1$ and $A \cap (B \cup C) \subseteq (B \cup C)$ for arbitrary sets A , B and C .

$\langle 2 \rangle 3.$ Q.E.D.

$\langle 1 \rangle 4.$ Q.E.D.

P : \forall -rule, definition 14.12 of \oplus and definition 14.3 of \rightsquigarrow .

□

Theorem 3 *For a STAIRS specification d, we define d^p as the corresponding probabilistic STAIRS specification, in which every occurrence of xalt is replaced by palt with probability set $Q = \langle 0 \dots 1 \rangle$ for each of the operands.*

We then have

$$a. \llbracket d \rrbracket \rightsquigarrow \pi_1. \llbracket d^p \rrbracket^p$$

$$b. \pi_1. \llbracket d^p \rrbracket^p \rightsquigarrow \llbracket d \rrbracket$$

where π_1 is an operator returning the interaction obligations from a set of p-obligations.

a. $P : \llbracket d \rrbracket \rightsquigarrow \pi_1. \llbracket d^p \rrbracket^p$

P : By induction on the number of nested operators in d .

(1)1. Base case: d contains no operators, i.e. d is a single event e .

(2)1. $d = d^p$

P : By definition of d^p .

(2)2. $\llbracket d \rrbracket = \{ (\{\langle e \rangle\}, \emptyset) \}$

P : Definition 14.17 of $\llbracket e \rrbracket$.

(2)3. $\llbracket d^p \rrbracket^p = \{ ((\{\langle e \rangle\}, \emptyset), \{1\}) \}$

P : (2)1 and definition 14.18 of $\llbracket e \rrbracket^p$.

(2)4. $\llbracket d \rrbracket = \pi_1. \llbracket d^p \rrbracket^p$

P : (2)2, (2)3 and definition of π_1 .

(2)5. $\llbracket d \rrbracket \rightsquigarrow \pi_1. \llbracket d^p \rrbracket^p$

P : (2)4 and reflexivity of \rightsquigarrow (proved in [HHR05a]).

(2)6. Q.E.D.

(1)2. Induction step:

L : op be the main operator of d , and d_1, \dots, d_n its operands.

A : $\forall i \in [1 \dots n] : \llbracket d_i \rrbracket \rightsquigarrow \pi_1. \llbracket d_i^p \rrbracket^p$ (induction hypothesis)

P : $\llbracket d \rrbracket \rightsquigarrow \pi_1. \llbracket d^p \rrbracket^p$

(2)1. C : $\text{op} \neq \text{xalt}$

P : By the induction hypothesis and monotonicity of \rightsquigarrow with respect to op (proved in [HHR05a]).

(2)2. C : $\text{op} = \text{xalt}$

(3)1. Choose arbitrary $o \in \llbracket d \rrbracket$

P : $\llbracket d \rrbracket$ is non-empty for all interactions d .

(3)2. Choose $i \in [1 \dots n]$ such that $o \in \llbracket d_i \rrbracket$

P : (3)1 and definition 14.4 of xalt.

(3)3. Choose $o' \in \pi_1. \llbracket d_i^p \rrbracket^p$ such that $o \rightsquigarrow o'$

P : (3)2, the induction hypothesis and definition 14.20 of \rightsquigarrow .

(3)4. $o' \in \pi_1. \llbracket d_i^p; Q \rrbracket^p$

P : (3)3, definition of π_1 and definition of probability decoration.

(3)5. $o' \in \pi_1. \llbracket d^p \rrbracket^p$

(4)1. $o' \in \pi_1. \{(o'', Q'') \mid (o'', Q'') \in \llbracket d_i^p; Q \rrbracket^p\}$

P : (3)4 and definition of $\llbracket \cdot \rrbracket^p$.

(4)2. $o' \in \pi_1. \{(\oplus\{(o'', Q'')\}, Q'') \mid (o'', Q'') \in \llbracket d_i^p; Q \rrbracket^p\}$

P : (4)1 and definition 14.12 which gives $\oplus\{(p, n), Q\} = (p, n)$.

(4)3. Q.E.D.

P : By the assumption (the structure of d), the definition of d^p , (4)2 and definition 14.11 part a, choosing $N = i$.

(3)6. $\forall o \in \llbracket d \rrbracket : \exists o' \in \pi_1. \llbracket d^p \rrbracket^p : o \rightsquigarrow o'$

P : (3)1, (3)3 and (3)5 and \forall -rule.

(3)7. Q.E.D.

P : (3)6 and definition 14.20 of \rightsquigarrow .

(2)3. Q.E.D.

P : The cases are exhaustive.

$\langle 1 \rangle 3.$ Q.E.D.

P : By induction on the structure of d .

□

b. P : $\pi_1. [\![d^p]\!]^p \rightsquigarrow [\![d]\!]$

P : By induction on the number of nested operators in d .

$\langle 1 \rangle 1.$ Base case: d^p contains no operators, i.e. d and d^p is a single event e .

P : Equal to the proof of $\langle 1 \rangle 1$ in the proof of theorem 3, part 1.

$\langle 1 \rangle 2.$ Induction step:

L : op be the main operator of d^p , and d_1^p, \dots, d_n^p its operands

A : $\forall i \in [1 \dots n] : \pi_1. [\![d_i^p]\!]^p \rightsquigarrow [\![d_i]\!]$ (induction hypothesis)

P : $\pi_1. [\![d^p]\!]^p \rightsquigarrow [\![d]\!]$

$\langle 2 \rangle 1.$ C : $op \neq palt$

P : By the induction hypothesis and monotonicity of \rightsquigarrow with respect to op (proved in [HHR05a]).

$\langle 2 \rangle 2.$ C : $op = palt$

P : The proof is split in two cases, one for each part of the palt-definition. In each case we use lemma 1 and the induction hypothesis to prove the existence of a refining obligation.

$\langle 3 \rangle 1.$ Choose arbitrary $o \in [\![d^p]\!]^p$.

P : $[\![d]\!]^p$ is non-empty for all interactions d .

$\langle 3 \rangle 2.$ C : $o \in \pi_1. \{ (\bigoplus_{i \in N} \{ po_i \}, \sum_{i \in N} \pi_2. po_i) \mid N \subseteq \{1, \dots, n\} \wedge N \neq \emptyset \wedge \forall i \in N : po_i \in [\![d_i^p; Q]\!]^p \}$

$\langle 4 \rangle 1.$ $o \in \{ (\bigoplus_{i \in N} \{ po_i \}) \mid$

$N \subseteq \{1, \dots, n\} \wedge N \neq \emptyset \wedge \forall i \in N : po_i \in [\![d_i^p; Q]\!]^p \}$

P : Definition of π_1 .

$\langle 4 \rangle 2.$ Choose $N \subseteq \{1, \dots, n\}$ such that

$o \in \{ (\bigoplus_{i \in N} \{ po_i \}) \mid \forall i \in N : po_i \in [\![d_i^p; Q]\!]^p \}$

P : $\langle 4 \rangle 1.$

$\langle 4 \rangle 3.$ For all $i \in N$ choose $(o_i, Q_i) \in [\![d_i^p; Q]\!]^p$ such that

$o = \bigoplus \bigcup_{i \in N} \{ (o_i, Q_i) \}$

P : $\langle 4 \rangle 2.$

$\langle 4 \rangle 4.$ $\forall i \in N : o_i \in \pi_1. [\![d_i^p]\!]^p$

P : $\langle 4 \rangle 3$ and definition of π_1 .

$\langle 4 \rangle 5.$ Choose arbitrary $j \in N$

P : N is non-empty by $\langle 4 \rangle 1.$

$\langle 4 \rangle 6.$ $o \rightsquigarrow o_j$

P : $\langle 4 \rangle 3, \langle 4 \rangle 5$ and lemma 1.

$\langle 4 \rangle 7.$ Choose $o'_j \in [\![d_j]\!]$ such that $o_j \rightsquigarrow o'_j$

P : $\langle 4 \rangle 4, \langle 4 \rangle 5$ and the induction hypothesis.

$\langle 4 \rangle 8.$ $o \rightsquigarrow o'_j$

P : $\langle 4 \rangle 6, \langle 4 \rangle 7$ and transitivity of \rightsquigarrow (proved in [HHR05a]).

- $\langle 4 \rangle 9. o'_j \in \llbracket d \rrbracket$
 P : $\langle 4 \rangle 7$ and definition 14.4 of xalt .
- $\langle 4 \rangle 10. \forall o \in \pi_1. \llbracket d^p \rrbracket^p : \exists o' \in \llbracket d \rrbracket : o \rightsquigarrow o'$
 P : $\langle 3 \rangle 1, \langle 4 \rangle 8, \langle 4 \rangle 9$ and \forall -rule.
- $\langle 4 \rangle 11. \text{Q.E.D.}$
 P : $\langle 4 \rangle 10$ and definition 14.20 of \rightsquigarrow .
- $\langle 3 \rangle 3. C : o \in \pi_1. \{(\oplus \bigcup_{i=1}^n \llbracket d_i^p; Q \rrbracket^p, \{1\} \cap \sum_{i=1}^n Q)\}$
- $\langle 4 \rangle 1. o = \oplus \bigcup_{i=1}^n \llbracket d_i^p; Q \rrbracket^p$
 P : Definition of π_1 .
- $\langle 4 \rangle 2. \text{Choose arbitrary } i \in [1 \dots n] \text{ and arbitrary } (o', Q') \in \llbracket d_i^p; Q \rrbracket^p$
 P : $\llbracket d \rrbracket^p$ is non-empty for all d .
- $\langle 4 \rangle 3. o' \in \pi_1. \llbracket d_i^p \rrbracket^p$
 P : $\langle 4 \rangle 2$, definition of π_1 and definition of probability decoration.
- $\langle 4 \rangle 4. o \rightsquigarrow o'$
 P : $\langle 4 \rangle 1, \langle 4 \rangle 2$ and lemma 1.
- $\langle 4 \rangle 5. \text{Choose } o'' \in \llbracket d_i \rrbracket \text{ such that } o' \rightsquigarrow o''$
 P : $\langle 4 \rangle 3$ and the induction hypothesis.
- $\langle 4 \rangle 6. o \rightsquigarrow o''$
 P : $\langle 4 \rangle 4, \langle 4 \rangle 5$ and transitivity of \rightsquigarrow (proved in [HHR05a]).
- $\langle 4 \rangle 7. o'' \in \llbracket d \rrbracket$
 P : $\langle 4 \rangle 5$ and definition 14.4 of xalt .
- $\langle 4 \rangle 8. \forall o \in \pi_1. \llbracket d^p \rrbracket^p : \exists o' \in \llbracket d \rrbracket : o \rightsquigarrow o'$
 P : $\langle 3 \rangle 1, \langle 4 \rangle 6, \langle 4 \rangle 7$ and \forall -rule.
- $\langle 4 \rangle 9. \text{Q.E.D.}$
 P : $\langle 4 \rangle 8$ and definition 14.20 of \rightsquigarrow .
- $\langle 3 \rangle 4. \text{Q.E.D.}$
 P : By definition 14.11 of palt , the cases are exhaustive.
- $\langle 2 \rangle 3. \text{Q.E.D.}$
 P : The cases are exhaustive.
- $\langle 1 \rangle 3. \text{Q.E.D.}$
 P : By induction on the structure of d .

□

Chapter 15

Relating Computer Systems to Sequence Diagrams with Underspecification, Inherent Nondeterminism and Probabilistic Choice – Part 1: Underspecification and Inherent Nondeterminism

Ragnhild Kobro Runde, Atle Refsdal, and Ketil Stølen

Publication.

Published as Technical Report 346, Department of Informatics, University of Oslo, 2007.

Abstract.

Having a sequence diagram specification and a computer system, we need to answer the question: *Is the system compliant with the sequence diagram specification in the desired way?* We present a procedure for answering this question for three variations of sequence diagrams. The procedure is independent of the choice of programming language used for the system. The semantics of sequence diagrams is denotational and based on traces. In order to answer the initial question, the procedure starts by obtaining the trace-set of the system by e.g. testing, and then transforming this into the same semantic model as that used for the sequence diagram. In addition to extending our earlier work on refinement relations for sequence diagrams, we define conformance relations relating systems to sequence diagrams.

The work is split in two parts. This paper presents part 1, in which we introduce the necessary definitions for using the compliance checking procedure on sequence diagrams with underspecification and sequence diagrams with inherent nondeterminism. In part 2 [RRS07], we present the definitions for using the procedure on sequence diagrams with probabilistic choice.

15.1 Introduction

Having a sequence diagram specification and a computer system, we need to answer the question: *Is the system compliant with the specification in the desired way?*

Sequence diagrams are widely used for specifying computer systems within a broad range of application domains. They are used for different methodological purposes including requirements capture, illustrating example runs, test scenario specification and risk scenario documentation. Although sequence diagrams are widely used in practice, their relationship to real computer systems is nevertheless surprisingly unclear. This is partly caused by the fact that sequence diagrams are used for different purposes, but even more so because in contrast to most other techniques for specifying dynamic behaviour they give only a partial view.

Answering the initial question above requires an understanding of what is meant by a computer system and to what extent such a system is different from a sequence diagram. Obviously, we need a formal model for computer systems. Also, the answer clearly depends on the expressiveness of the sequence diagram dialect we are using. In this paper we study the problem with respect to two different variations of sequence diagrams, as formally defined in the denotational trace semantics of STAIRS [HHRS05]. The two variations are sequence diagrams with underspecification and sequence diagrams with inherent nondeterminism.

The notion of compliance is closely related to that of refinement. Refinement is a way of relating different specifications of the same system, where the idea is that a refinement should be a more detailed description containing all the constraints given by the original specification, in addition to some new ones.¹ Different development stages may require different notions of refinement. The final specification used when implementing the system, is the result of several successive refinement steps. The system should be compliant not only with the final specification, but also with all specifications in the chain of refinements. Consequently, we may need several notions of compliance corresponding to the various notions of refinement.

In this paper we only consider compliance for sequence diagrams without external input and output. For such sequence diagrams, we propose the following compliance checking procedure:

- a. Given a computer system I and a sequence diagram d , use e.g. testing on I to obtain the trace-set describing its behaviour.
- b. Transform this trace-set into the same semantic model as that used for d .
- c. Depending on the kind of compliance desired, select the appropriate compliance relation.
- d. I is compliant with d if this compliance relation holds between the semantics of d and the representation of I obtained in step b.

This paper is organized as follows: In Section 15.2 we state the requirements that a step-wise procedure for checking computer systems against sequence diagrams needs to fulfil. Section 15.3 gives a general introduction to sequence diagrams and their denotational trace semantics. Sections 15.4 and 15.5 introduce sequence diagrams with underspecification and inherent nondeterminism, respectively, and define what it means for a system to be compliant with such

¹Note that we use the term “constraint” rather loosely. For instance, the addition of a new constraint may result in the specification requiring more behaviours of the system.

sequence diagrams. In Section 15.6 we present theoretical results related to the definitions of refinement and compliance. We discuss related work in Section 15.7, before concluding in Section 15.8. Appendices 15.A and 15.B give a detailed overview of the theoretical results, together with the necessary proofs.

15.2 Requirements

In order to motivate the following discussion and formal definitions, we formulate a number of requirements that our procedure has been designed to fulfil. That these requirements are met, are demonstrated throughout the discussion and summed up in Section 15.8.

- a. The procedure should be independent of the choice of programming language in which the system is implemented. A sequence diagram does not prescribe any particular programming language, and the procedure should be sufficiently general to capture all possible choices. In general, we cannot assume that we have access to the source code of the system. This means that the only knowledge about the system that may be used by the procedure, is what can be obtained by testing. Although not feasible in practice, we assume that we are able to observe infinite runs. Otherwise, we would have to restrict ourselves to safety properties.
- b. The notion of compliance should be a special case of refinement. Given a sequence diagram and its refinement, the procedure should give that a system is compliant with the refinement only if the system is also compliant with the original sequence diagram.
- c. There should be a natural correspondence between the compliance relations for the two variations of sequence diagrams. The language of sequence diagrams without inherent nondeterminism is a subclass of the language that also allows inherent nondeterminism. This means that the general compliance relation for sequence diagrams with inherent nondeterminism should at least capture everything allowed by the general compliance relation for sequence diagrams containing only underspecification.
- d. The procedure should be faithful to the underlying ideas and principles of UML 2.1 [OMG06] sequence diagrams. UML is the leading specification language within the software industry of today, and our goal is that our approach should be of help for UML practitioners.

15.3 Sequence Diagrams and Trace Semantics

This section gives necessary background for the following sections. It provides a general introduction to sequence diagrams and their denotational trace semantics. In this section we consider only three operators on sequence diagrams, namely the operators for refusal, sequential and parallel composition. The following two sections extend this basic set of operators with operators for underspecification and inherent nondeterminism, in each case focusing on how to determine whether a given system is in compliance with such a sequence diagram.

We use the simple sequence diagram in Figure 15.1 to introduce some terminology. S is the name of the sequence diagram, A and B are lifelines (corresponding to e.g. components or objects), while m is a message from A to B. We say that the diagram includes two events, the sending of m (denoted $!m$) and the reception of m (denoted $?m$).

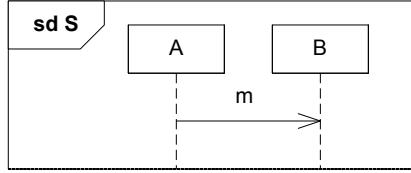


Figure 15.1: Simple sequence diagram

To assign precise meaning to a sequence diagram, we use denotational trace semantics as defined in STAIRS [HHR05]. This formal semantics is compliant with the semi-formal descriptions given in the UML 2.1 standard [OMG06]. A trace is a sequence of events representing a system run. An event is a pair (k, m) consisting of a kind k (either $!$ or $?$) and a message m . A message is a triple (s, tr, re) consisting of a signal s , a transmitter lifeline tr and a receiver lifeline re .

For a trace to be well-formed, we require that for all messages:

- if both the sender and receiver lifeline are present in the diagram, then both the send and the receive event are present in the trace;
- the send event is ordered before the corresponding receive event if both events are present in the trace.

The semantics of a sequence diagram d is denoted $\llbracket d \rrbracket$. In the basic case, the semantics of a sequence diagram is an *interaction obligation* (p, n) where p is a set of positive (i.e. valid) traces and n is a set of negative (i.e. invalid) traces. Traces not in the diagram are called inconclusive, and may be introduced as positive or negative by later refinement steps. Letting \mathcal{H} denote the universe of all well-formed traces, the traces $\mathcal{H} \setminus (p \cup n)$ are inconclusive in the interaction obligation (p, n) .

Parallel composition (\parallel) of two trace sets corresponds to point-wise interleaving of their individual traces. The ordering of events within each trace is maintained in the result. For sequential composition (\succsim) we require in addition that for events on the same lifeline, all events from the first trace is ordered before the events from the second trace. Formally:

$$\begin{aligned} s_1 \parallel s_2 &\stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists p \in \{1, 2\}^\infty : \\ &\quad \pi_2((\{1\} \times \mathcal{E}) \oplus (p, h)) \in s_1 \wedge \\ &\quad \pi_2((\{2\} \times \mathcal{E}) \oplus (p, h)) \in s_2\} \end{aligned} \tag{15.1}$$

$$\begin{aligned} s_1 \succsim s_2 &\stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \forall l \in \mathcal{L} : \\ &\quad e.l \odot h = e.l \odot h_1 \cap e.l \odot h_2\} \end{aligned} \tag{15.2}$$

where \mathcal{E} and \mathcal{L} are the sets of all events and lifelines, respectively; $e.l$ is the set of events that may take place on the lifeline l ; π_2 is a projection operator returning the second element of a pair;

and \smallfrown is the concatenation operator for sequences. \circledcirc and \circledast are filtering operators for traces and pairs of traces, respectively. $E \circledcirc h$ is the trace obtained from the trace h by removing from h all events that is not in the set of events E . For instance, we have that

$$\{e_1, e_3\} \circledast \langle e_1, e_1, e_2, e_1, e_3, e_2 \rangle = \langle e_1, e_1, e_1, e_3 \rangle$$

The operator \circledast is a generalization of \circledcirc filtering pairs of traces with respect to pairs of elements such that for instance

$$\begin{aligned} \{(1, e_1), (1, e_2)\} \circledcirc (\langle 1, 1, 2, 1, 2 \rangle, \langle e_1, e_1, e_1, e_2, e_2 \rangle) \\ = (\langle 1, 1, 1 \rangle, \langle e_1, e_1, e_2 \rangle) \end{aligned}$$

For formal definitions of \circledcirc and \circledast , see [BS01].

For interaction obligations, parallel composition (\parallel), sequential composition (\succsim) and refusal (\dagger) are defined by:

$$(p_1, n_1) \parallel (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \parallel p_2, (n_1 \parallel p_2) \cup (n_1 \parallel n_2) \cup (p_1 \parallel n_2)) \quad (15.3)$$

$$(p_1, n_1) \succsim (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \succsim p_2, (n_1 \succsim p_2) \cup (n_1 \succsim n_2) \cup (p_1 \succsim n_2)) \quad (15.4)$$

$$\dagger(p_1, n_1) \stackrel{\text{def}}{=} (\emptyset, p_1 \cup n_1) \quad (15.5)$$

Notice that composing a positive and a negative trace always yields a negative trace, while the result of composing an inconclusive trace with a positive or negative trace is always inconclusive.

Finally, the sequence diagram operators for parallel composition (par), sequential composition (seq) and negative behaviour (refuse) are defined by:

$$[\![d_1 \text{ par } d_2]\!] \stackrel{\text{def}}{=} [\![d_1]\!] \parallel [\![d_2]\!] \quad (15.6)$$

$$[\![d_1 \text{ seq } d_2]\!] \stackrel{\text{def}}{=} [\![d_1]\!] \succsim [\![d_2]\!] \quad (15.7)$$

$$[\![\text{refuse } d_1]\!] \stackrel{\text{def}}{=} \dagger[\![d_1]\!] \quad (15.8)$$

15.4 Relating Computer Systems to Sequence Diagrams with Underspecification

When writing specifications, it is often useful to leave certain aspects of the system behaviour open. This is known as underspecification. Typically, underspecification is a consequence of abstraction and a desire to focus on the essential behaviour of the system. In sequence diagrams, underspecification may be the result of weak sequencing or specified using the operator alt, describing alternative behaviours that the system *may* exhibit. Underspecification may be removed either by later development steps (refinements) or during the implementation process.

Underspecification in the sense of alt corresponds to taking the pair-wise union of the positive and negative trace-sets of the operands. Formally:

$$[\![d_1 \text{ alt } d_2]\!] \stackrel{\text{def}}{=} [\![d_1]\!] \uplus [\![d_2]\!] \quad (15.9)$$

where inner union (\uplus) on interaction obligations is defined by:

$$(p_1, n_1) \uplus (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \cup p_2, n_1 \cup n_2) \quad (15.10)$$

15.4.1 Refinement

Refinement means to add more information to the specification in order to bring it closer to a real system. An important requirement is that any valid system that is compliant with the refinement should also be compliant with the original specification. In addition to the basic refinement relation defined in [HHR05], we define another relation called restricted refinement. The idea is that the two refinement relations will be used in different phases of the development process.

Refinement As sequence diagrams are incomplete specifications describing only parts of the system behaviour, a refinement step may add more positive and/or negative behaviours to the specification, hence reducing the set of inconclusive traces. Also, a refinement step may reduce underspecification, i.e. redefine positive traces as negative. Negative traces always remain negative. Formally, refinement of interaction obligations is defined by:

$$(p, n) \rightsquigarrow_r (p', n') \stackrel{\text{def}}{=} n \subseteq n' \wedge p \subseteq p' \cup n' \quad (15.11)$$

As can be seen from the definition, a refinement may legally redefine all of the original positive traces as negative. Having this possibility may be important in an early development phase focusing on exploring the desired system requirements.

Restricted refinement At some stage during the development process it may be natural to fix the set of positive traces, with the intention that at least one of these should be present in any system compliant with this sequence diagram. After that, valid refinement steps may only redefine positive and inconclusive traces as negative, in order to remove underspecification and decide on the exact set of traces that may be produced by the final system. Extending the set of positive traces is no longer allowed:

$$(p, n) \rightsquigarrow_{rr} (p', n') \stackrel{\text{def}}{=} (p, n) \rightsquigarrow_r (p', n') \wedge p' \subseteq p \quad (15.12)$$

15.4.2 Compliance

As explained in Section 15.2, we assume that all we know about a computer system is its set of traces. The traces are assumed to be well-formed in the sense explained in Section 15.3. This allows us to reason independently of any particular programming language, and also to handle applications where different components may be written in different languages.

In order to check a system I represented by its set of traces against a sequence diagram specification d using the semantic model outlined in Section 15.3, we first transform I into an interaction obligation $\langle I \rangle_d$:

$$\langle I \rangle_d \stackrel{\text{def}}{=} (\text{traces}(I), \mathcal{H}^{ll(d)} \setminus \text{traces}(I)) \quad (15.13)$$

where $\mathcal{H}^{ll(d)}$ is the set of all well-formed traces consisting only of events taking place on the lifelines in the sequence diagram d , denoted $ll(d)$.

A general refinement principle in STAIRS is that traces described as positive or negative in the original specification cannot become inconclusive by a refinement step, as this would mean

deviation from earlier given constraints. Since compliance should be a special case of refinement, $\langle I \rangle_d$ must include (as positive or negative) at least all traces described by d . Definition (15.13) ensures this by the use of $\mathcal{H}^{ll(d)}$. Employing $\mathcal{H}^{ll(d)}$ and not \mathcal{H} , guarantees consistency when performing parallel composition of two systems with disjoint sets of lifelines.

Corresponding to the two refinement relations in Section 15.4.1, we then define two different compliance relations.

Compliance relation A system I complies to a sequence diagram d if the semantics we get by using definition (15.13) is a refinement of $\llbracket d \rrbracket$. Formally:

$$\llbracket d \rrbracket \mapsto_r \langle I \rangle_d \stackrel{\text{def}}{=} \llbracket d \rrbracket \rightsquigarrow_r \langle I \rangle_d \quad (15.14)$$

Restricted compliance relation A sequence diagram specification gives a global view of the system behaviour, but is often implemented as a number of components. As each of these components only has a local view of the overall system behaviour, their independent behaviours may combine in unexpected ways resulting in so-called implied scenarios [AEY00], i.e. traces that are inconclusive in the sequence diagram specification.

With restricted compliance, the system should contain at least one of the positive traces from the sequence diagram. Because of implied scenarios, we do not require that all possible system traces are explicitly described as positive in the sequence diagram. For a system I , we instead require that $\text{traces}(I)$ and the positive traces of the sequence diagram have at least one trace in common. In addition, $\text{traces}(I)$ may contain arbitrary many of the positive and inconclusive traces from the sequence diagram. Formally:

$$\llbracket d \rrbracket \mapsto_{rr} \langle I \rangle_d \stackrel{\text{def}}{=} \llbracket d \rrbracket \mapsto_r \langle I \rangle_d \wedge \pi_1(\llbracket d \rrbracket) \cap \pi_1(\langle I \rangle_d) \neq \emptyset \quad (15.15)$$

where π_1 is a projection operator returning the first element of a pair, in this case the positive traces of d and $\langle I \rangle_d$ (i.e. $\text{traces}(I)$), respectively.

15.4.3 Example

As a simple example, consider the specification of a gambling machine in Figure 15.2. First, the machine receives either a dime or a quarter. As a result, the machine either sends the message “You won” together with a dollar, or the message “You lost”.² The veto operator is a high-level operator defined by:

$$\text{veto } d \stackrel{\text{def}}{=} \text{skip alt refuse } d \quad (15.16)$$

where skip is the empty diagram defined by:

$$\llbracket \text{skip} \rrbracket \stackrel{\text{def}}{=} (\{\langle \rangle\}, \emptyset) \quad (15.17)$$

²As we will come back to in Section 15.5, alt is not the best operator to use between these two last alternatives.

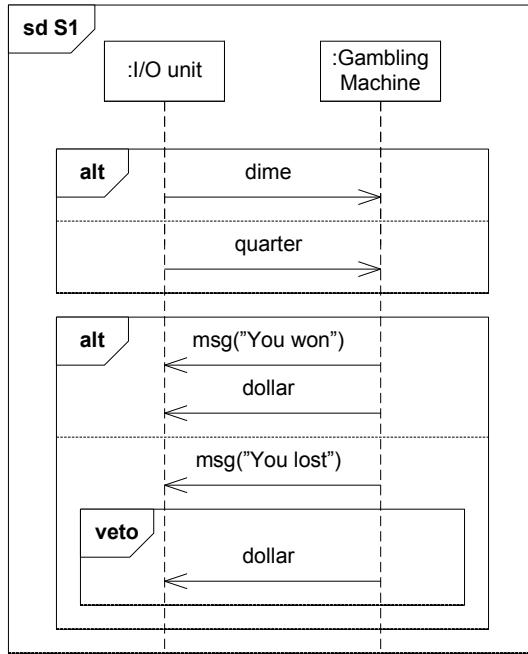


Figure 15.2: Sequence diagram with underspecification (alt)

In this example, **veto** is used to specify that the message “You lost” should *not* be followed by a dollar. Shortening each message, the semantics $\llbracket S_1 \rrbracket$ of this sequence diagram is:

$$\begin{aligned}
& (\{ \langle !di, ?di, !m(yw), ?m(yw), !do, ?do \rangle, \langle !qu, ?qu, !m(yw), ?m(yw), !do, ?do \rangle, \\
& \quad \langle !di, ?di, !m(yw), !do, ?m(yw), ?do \rangle, \langle !qu, ?qu, !m(yw), !do, ?m(yw), ?do \rangle, \\
& \quad \langle !di, ?di, !m(yl), ?m(yl) \rangle, \langle !qu, ?qu, !m(yl), ?m(yl) \rangle \}, \\
& \{ \langle !di, ?di, !m(yl), ?m(yl), !do, ?do \rangle, \langle !qu, ?qu, !m(yl), ?m(yl), !do, ?do \rangle, \\
& \quad \langle !di, ?di, !m(yl), !do, ?m(yl), ?do \rangle, \langle !qu, ?qu, !m(yl), !do, ?m(yl), ?do \rangle \})
\end{aligned}$$

A possible way to implement this sequence diagram would be to build a system I_1 where the gambling machine receives a dime, after which it responds with a “You lost” message and then nothing more happens. This may be represented by the trace-set

$$traces(I_1) = \{ \langle !di, ?di, !m(yl), ?m(yl) \rangle \}$$

which gives

$$\langle I_1 \rangle_{S_1} = (\{ \langle !di, ?di, !m(yl), ?m(yl) \rangle \}, \mathcal{H}^{ll(S_1)} \setminus \{ \langle !di, ?di, !m(yl), ?m(yl) \rangle \})$$

It is straightforward to see that I_1 is in compliance with the sequence diagram S_1 according to both definitions (15.14) and (15.15), as the trace $\langle !di, ?di, !m(yl), ?m(yl) \rangle$ is positive in $\llbracket S_1 \rrbracket$, and the negative traces of $\llbracket S_1 \rrbracket$ are also negative in $\langle I_1 \rangle_{S_1}$.

15.5 Relating Computer Systems to Sequence Diagrams with Inherent Nondeterminism

Using only underspecification, a system may be in compliance with the sequence diagram even if it contains one only of the positive traces and nothing else. In many cases this is not sufficient. One example is the gambling machine from Section 15.4.3, where the sequence diagram allowed a system where the only possible outcome was the user losing his money. A more realistic specification would be to require that both winning and losing should be possible outcomes. Also, the choice between winning and losing should be performed nondeterministically (or at least appear so to the user of the gambling machine).

For specifying inherent nondeterminism, or alternatives that must all be reflected in the specified system, the operator xalt (first introduced in [HS03]) may be used. To distinguish between underspecification and inherent nondeterminism also at the semantic level, the semantics of a sequence diagram d is no longer a single interaction obligation as in Sections 15.3 and 15.4, but instead a *set* of m interaction obligations $\llbracket d \rrbracket = \{(p_1, n_1), (p_2, n_2), \dots, (p_m, n_m)\}$ for some natural number m . The idea is that each interaction obligation gives a requirement that must be fulfilled by any system in compliance with it. Formally, the xalt operator is defined by:

$$\llbracket d_1 \text{xalt} d_2 \rrbracket \stackrel{\text{def}}{=} \llbracket d_1 \rrbracket \cup \llbracket d_2 \rrbracket \quad (15.18)$$

Hence, the composition of d_1 and d_2 by xalt requires all the inherent nondeterminism specified by d_1 in addition to all the inherent nondeterminism specified by d_2 .

With this extended semantic model, we also need definitions for parallel composition (\parallel), sequential composition (\succsim), underspecification (\uplus) and refusal (\dagger) on sets of interaction obligations:

$$O_1 \text{ op } O_2 \stackrel{\text{def}}{=} \{o_1 \text{ op } o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\} \quad (15.19)$$

$$\dagger O_1 \stackrel{\text{def}}{=} \{\dagger o_1 \mid o_1 \in O_1\} \quad (15.20)$$

where op is one of \parallel , \succsim or \uplus .

15.5.1 Refinement

For a specification with both inherent nondeterminism and underspecification, we distinguish between four different refinement relations. The most general notion is general refinement, which is typically used initially, while the most specific notion is restricted limited refinement, which is more likely to be used near the end of the development process.

General and restricted general refinement For a specification having several interaction obligations as its semantics, we require that each one should be refined by at least one interaction obligation in any valid refinement. The only difference between general (\rightsquigarrow_g) and restricted general (\rightsquigarrow_{rg}) refinement, is with respect to the refinement definition used between the two interaction obligations:

$$\llbracket d \rrbracket \rightsquigarrow_{(r)g} \llbracket d' \rrbracket \stackrel{\text{def}}{=} \forall o \in \llbracket d \rrbracket : \exists o' \in \llbracket d' \rrbracket : o \rightsquigarrow_{(r)r} o' \quad (15.21)$$

where \rightsquigarrow_r and \rightsquigarrow_{rr} are refinement of interaction obligations as defined by definitions (15.11) and (15.12), respectively.

Limited and restricted limited refinement Both versions of definition (15.21) allow a refinement to introduce new interaction obligations that are not refinements of any interaction obligations in the original specification, possibly increasing the inherent nondeterminism required of the final system. A trace may be positive in one of these new interaction obligations even if it is negative in all other interaction obligations. In limited (\rightsquigarrow_l) and restricted limited (\rightsquigarrow_{rl}) refinement, adding new interaction obligations like this is not allowed:

$$\begin{aligned} \llbracket d \rrbracket \rightsquigarrow_{(r)l} \llbracket d' \rrbracket &\stackrel{\text{def}}{=} \llbracket d \rrbracket \rightsquigarrow_{(r)g} \llbracket d' \rrbracket \\ &\wedge \forall o' \in \llbracket d' \rrbracket : \exists o \in \llbracket d \rrbracket : o \rightsquigarrow_{(r)r} o' \end{aligned} \quad (15.22)$$

15.5.2 Compliance

In order to characterize compliance between a system I and a sequence diagram d with inherent nondeterminism (as well as underspecification), we redefine $\langle I \rangle_d$ to consist of one interaction obligation for each trace in $\text{traces}(I)$:

$$\langle I \rangle_d \stackrel{\text{def}}{=} \{(\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\}) \mid h \in \text{traces}(I)\} \quad (15.23)$$

Corresponding to the four refinement relations in Section 15.5.1, we then define four different compliance relations.

General and restricted general compliance Similar to (restricted) general refinement, a system I is in (restricted) general compliance with a sequence diagram d if every interaction obligation in $\llbracket d \rrbracket$ is reflected in at least one of the interaction obligations we get by using definition (15.23). The only difference between general (\mapsto_g) and restricted general (\mapsto_{rg}) compliance is with respect to the compliance relation used for each single interaction obligation.

$$\llbracket d \rrbracket \mapsto_{(r)g} \langle I \rangle_d \stackrel{\text{def}}{=} \forall o \in \llbracket d \rrbracket : \exists o' \in \langle I \rangle_d : o \mapsto_{(r)r} o' \quad (15.24)$$

Limited and restricted limited compliance Similar to (restricted) limited refinement, (restricted) limited compliance requires that every interaction obligation obtained by definition (15.23) complies with at least one interaction obligation in $\llbracket d \rrbracket$:

$$\begin{aligned} \llbracket d \rrbracket \mapsto_{(r)l} \langle I \rangle_d &\stackrel{\text{def}}{=} \llbracket d \rrbracket \mapsto_{(r)g} \langle I \rangle_d \\ &\wedge \forall o' \in \langle I \rangle_d : \exists o \in \llbracket d \rrbracket : o \mapsto_{(r)r} o' \end{aligned} \quad (15.25)$$

15.5.3 Example

Figure 15.3 is a revised specification of the gambling machine, replacing the second alt operator with xalt and adding some more negative behaviours. The ref-construct may be understood as a syntactical shorthand for the contents of the referenced sequence diagram.

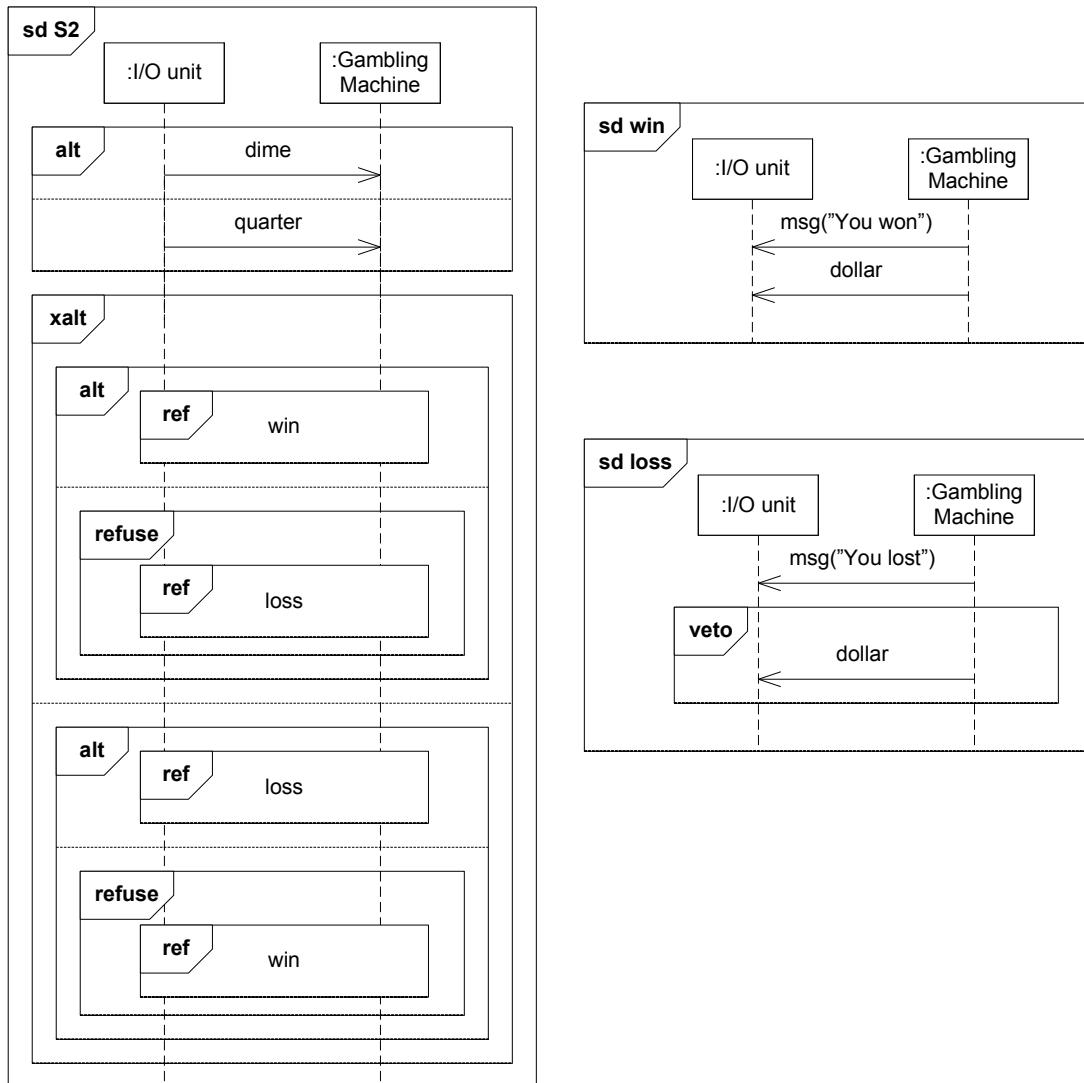


Figure 15.3: Sequence diagram with inherent nondeterminism (xalt)

The semantics $\llbracket S_2 \rrbracket$ of this sequence diagram is a set of two interaction obligations:

$$\begin{aligned} & \{ (\{ \langle !di, ?di, !m(yw), ?m(yw), !do, ?do \rangle, \langle !qu, ?qu, !m(yw), ?m(yw), !do, ?do \rangle, \\ & \quad \langle !di, ?di, !m(yw), !do, ?m(yw), ?do \rangle, \langle !qu, ?qu, !m(yw), !do, ?m(yw), ?do \rangle \}, \\ & \quad \{ \langle !di, ?di, !m(yl), ?m(yl) \rangle, \langle !qu, ?qu, !m(yl), ?m(yl) \rangle, \\ & \quad \langle !di, ?di, !m(yl), ?m(yl), !do, ?do \rangle, \langle !qu, ?qu, !m(yl), ?m(yl), !do, ?do \rangle, \\ & \quad \langle !di, ?di, !m(yl), !do, ?m(yl), ?do \rangle, \langle !qu, ?qu, !m(yl), !do, ?m(yl), ?do \rangle \}), \\ & (\{ \langle !di, ?di, !m(yl), ?m(yl) \rangle, \langle !qu, ?qu, !m(yl), ?m(yl) \rangle \}, \\ & \quad \{ \langle !di, ?di, !m(yw), ?m(yw), !do, ?do \rangle, \langle !qu, ?qu, !m(yw), ?m(yw), !do, ?do \rangle, \\ & \quad \langle !di, ?di, !m(yw), !do, ?m(yw), ?do \rangle, \langle !qu, ?qu, !m(yw), !do, ?m(yw), ?do \rangle, \\ & \quad \langle !di, ?di, !m(yl), ?m(yl), !do, ?do \rangle, \langle !qu, ?qu, !m(yl), ?m(yl), !do, ?do \rangle, \\ & \quad \langle !di, ?di, !m(yl), !do, ?m(yl), ?do \rangle, \langle !qu, ?qu, !m(yl), !do, ?m(yl), ?do \rangle \}) \} \end{aligned}$$

The system I_1 given in section 15.4.3 is not in compliance with S_2 , as the only trace $\langle !di, ?di, !m(yl), ?m(yl) \rangle$ in $\langle I_1 \rangle_{S_2}$ is negative in the first interaction obligation in $\llbracket S_2 \rrbracket$, meaning that this interaction obligation is not reflected in the system as required by both definitions (15.24) and (15.25).

However, a system I_2 with trace-set as given by $\text{traces}(I_2) = \{t_1, t_2, t_3\}$ where $t_1 = \langle !di, ?di, !m(yw), ?m(yw), !do, ?do \rangle$, $t_2 = \langle !di, ?di, !m(yw), !do, ?m(yw), ?do \rangle$ and $t_3 = \langle !di, ?di, !m(yl), ?m(yl) \rangle$, is in compliance with S_2 according to all of the compliance relations \mapsto_g , \mapsto_{rg} , \mapsto_l and \mapsto_{rl} . To realize this notice that the three traces in $\text{traces}(I_2)$ will give rise to three different interaction obligations when using definition (15.23) of $\langle I_2 \rangle_{S_2}$. The interaction obligation $(\{t_1\}, \mathcal{H}^{ll(S_2)} \setminus \{t_1\})$ is in compliance with the first interaction obligation in $\llbracket S_2 \rrbracket$ according to both \mapsto_r and \mapsto_{rr} . The same is the case for the interaction obligation $(\{t_2\}, \mathcal{H}^{ll(S_2)} \setminus \{t_2\})$. Similarly, the interaction obligation $(\{t_3\}, \mathcal{H}^{ll(S_2)} \setminus \{t_3\})$ is in compliance with the second interaction obligation in $\llbracket S_2 \rrbracket$ according to both \mapsto_r and \mapsto_{rr} . Hence, both interaction obligations in $\llbracket S_2 \rrbracket$ are reflected in an interaction obligation in $\langle I_2 \rangle_{S_2}$, and each of the three interaction obligations in $\langle I_2 \rangle_{S_2}$ is in compliance with an interaction obligation in $\llbracket S_2 \rrbracket$.

15.6 Results

In this section we present a number of essential properties that have been established, such as transitivity, monotonicity and the relationship between refinement and compliance. For proofs, we refer to the appendices.

Theorem 1 (Transitivity of refinement.) *For every refinement relation \rightsquigarrow :*

$$\llbracket d \rrbracket \rightsquigarrow \llbracket d' \rrbracket \wedge \llbracket d' \rrbracket \rightsquigarrow \llbracket d'' \rrbracket \Rightarrow \llbracket d \rrbracket \rightsquigarrow \llbracket d'' \rrbracket$$

Transitivity is important, as it ensures that the result of successive refinement steps is a valid refinement of the original sequence diagram. As all other refinement relations are special cases of general refinement, the use of different refinement relations in the various steps ensures that the resulting sequence diagram is at least a general refinement of the original sequence diagram.

Theorem 2 (Monotonicity of refinement.) *For every refinement relation \rightsquigarrow :*

$$\begin{aligned} \llbracket d_1 \rrbracket \rightsquigarrow \llbracket d'_1 \rrbracket \wedge \llbracket d_2 \rrbracket \rightsquigarrow \llbracket d'_2 \rrbracket \Rightarrow \\ \llbracket \text{refuse } d_1 \rrbracket \rightsquigarrow \llbracket \text{refuse } d'_1 \rrbracket \\ \wedge \llbracket d_1 \text{ seq } d_2 \rrbracket \rightsquigarrow \llbracket d'_1 \text{ seq } d'_2 \rrbracket \\ \wedge \llbracket d_1 \text{ par } d_2 \rrbracket \rightsquigarrow \llbracket d'_1 \text{ par } d'_2 \rrbracket \\ \wedge \llbracket d_1 \text{ alt } d_2 \rrbracket \rightsquigarrow \llbracket d'_1 \text{ alt } d'_2 \rrbracket \\ \wedge \llbracket d_1 \text{ xalt } d_2 \rrbracket \rightsquigarrow \llbracket d'_1 \text{ xalt } d'_2 \rrbracket \end{aligned}$$

Monotonicity ensures that the different parts of a sequence diagram may be refined separately. Again, using different refinement relations means that the resulting sequence diagram will at least be a general refinement of the original one.

Theorem 3 (Transitivity between refinement and compliance.) *For every refinement relation \rightsquigarrow and compliance relation \mapsto with the same subscript:*

$$\llbracket d_1 \rrbracket \rightsquigarrow \llbracket d_2 \rrbracket \wedge \llbracket d_2 \rrbracket \mapsto \langle I \rangle_{d_2} \Rightarrow \llbracket d_1 \rrbracket \mapsto \langle I \rangle_{d_1}$$

In general, the compliance relation used for relating I to d_2 may be more restrictive (i.e. allowing only a subset of the implementations) than the one that must be used for relating I to d_1 . Theorem 3 is important as it tells us that in this case, the compliance relation to be used for relating I to d_1 is the one corresponding to the refinement relation used when going from d_1 to d_2 .

A sequence diagram with no `xalt` operator can be viewed either as a diagram with underspecification or as a diagram with inherent nondeterminism, since it contains only operators that are legal in both of these variations. The following Theorem 4 characterizes the relationships between the different interpretations with respect to refinement and compliance.

Until now we have overloaded the notation for the semantic representation of diagrams and computer systems in order to enhance readability. We now need to introduce the full notation. Let $\llbracket d \rrbracket^u$ and $\llbracket d \rrbracket^i$ denote the semantics of the sequence diagram d when viewed as a sequence diagram with underspecification (a single interaction obligation) or inherent nondeterminism (a set of interaction obligations), respectively. Similarly, for a system I we use $\langle I \rangle_d^u$ and $\langle I \rangle_d^i$ to denote its semantic representation with respect to d according to definition (15.13) or definition (15.23), respectively.

Theorem 4 (Correspondence.) *For a sequence diagram d without inherent nondeterminism:*

$$\llbracket d \rrbracket^u \mapsto_r \langle I \rangle_d^u \Rightarrow \llbracket d \rrbracket^i \mapsto_g \langle I \rangle_d^i$$

$$\llbracket d \rrbracket^u \mapsto_{rr} \langle I \rangle_d^u \Rightarrow \llbracket d \rrbracket^i \mapsto_{rg} \langle I \rangle_d^i$$

$$\llbracket d \rrbracket^u \mapsto_r \langle I \rangle_d^u \Leftrightarrow \llbracket d \rrbracket^i \mapsto_l \langle I \rangle_d^i$$

$$\llbracket d \rrbracket^u \mapsto_{rr} \langle I \rangle_d^u \Leftrightarrow \llbracket d \rrbracket^i \mapsto_{rl} \langle I \rangle_d^i$$

From Theorem 4, we see that (restricted) general compliance $\mapsto_{(r)g}$ allows at least all of the implementations allowed by (restricted) compliance $\mapsto_{(r)r}$. We now explain why (restricted) general compliance $\mapsto_{(r)g}$ may allow other implementations as well. As an example, assume that d is a sequence diagram where the trace t is negative in every interaction obligation in $\llbracket d \rrbracket^i$. According to (restricted) general refinement $\rightsquigarrow_{(r)g}$, a valid refinement of d is the sequence diagram d' with semantics $\llbracket d' \rrbracket^i = \llbracket d \rrbracket^i \cup \{(\{t\}, \emptyset)\}$. In other words, (restricted) general refinement $\rightsquigarrow_{(r)g}$ allows the addition of new interaction obligations where a trace may be positive even if it is negative in all of the original interaction obligations. This is also true for (restricted) general compliance $\mapsto_{(r)g}$, which is meant to reflect this refinement relation. However, implementing a negative trace is not allowed by (restricted) compliance $\mapsto_{(r)r}$, where a single interaction obligation is the semantic model used for representing both the sequence diagram and the system.

Implementing negative traces is also not allowed by limited compliance \mapsto_l , and we see from Theorem 4 that for sequence diagrams without inherent nondeterminism, compliance \mapsto_r and limited compliance \mapsto_l are always in accordance with each other.

On the other hand, restricted compliance \mapsto_{rr} allows implementations not allowed by restricted limited compliance \mapsto_{rl} . As an example, assume that d is a sequence diagram with semantics $\llbracket d \rrbracket^u = (\{t_1\}, \emptyset)$ (i.e. $\llbracket d \rrbracket^i = \{(\{t_1\}, \emptyset)\}$) and that I is a system such that $\text{traces}(I) = \{t_1, t_2\}$. Using definition (15.13) we get that the single interaction obligation of $\langle I \rangle_d^u$ is $(\{t_1, t_2\}, \mathcal{H}^{ll(d)} \setminus \{t_1, t_2\})$, meaning that I is in restricted compliance \mapsto_{rr} to d . However, using definition (15.23), we get $\langle I \rangle_d^i = \{(\{t_1\}, \mathcal{H}^{ll(d)} \setminus \{t_1\}), (\{t_2\}, \mathcal{H}^{ll(d)} \setminus \{t_2\})\}$, which is not in restricted limited compliance \mapsto_{rl} with d as d does not contain any interaction obligation where t_2 is positive. In other words, restricted compliance \mapsto_{rr} allows a system to perform traces that are inconclusive in the sequence diagram, while this is not allowed by restricted limited compliance \mapsto_{rl} .

15.7 Related Work

To the best of our knowledge, there is no other paper treating the relationship between computer systems and sequence diagrams as thoroughly as we have done in this paper. The closest is the work by Cengarle and Knapp in [CK04], which defines an implementation notion which has inspired our notion of restricted compliance. However, inherent nondeterminism is not treated.

The basis of this paper is sequence diagrams as defined in e.g. UML 2.1 [OMG06]. As the focus of this paper is on compliance relations and not sequence diagrams as such, we have covered only the most essential of the UML 2.1 operators. In addition, we have considered an operator for specifying inherent nondeterminism. This operator is not found in UML 2.1, and neither in most other variants of sequence diagrams such as Message Sequence Charts (MSCs) [ITU99].

Live Sequence Charts (LSCs) [DH99, HM03] is an extension of MSCs, where elements in the diagram may be specified as either universal (mandatory) or existential (optional). An existential chart specifies a behaviour (one or more traces) that must be satisfied by at least one system run. A universal chart specifies all allowed traces, but does not require that more than one is implemented. [HM03] defines an operational semantics for LSCs, but as the focus is on simulating the specifications, neither refinement nor compliance relations are defined.

Related to implementations, [Krü00] defines four possible interpretations of a single MSC: negated, exact, existential and universal. The negated interpretation means that the MSC describes behaviour that should not happen, and corresponds to the UML use of negation operators within a sequence diagram. For the exact interpretation, all behaviours that are not in the MSC are forbidden. An existential MSC describes behaviour that cannot be prohibited by the system in all executions. We obtain the same result by letting an interaction obligation contain the existential behaviour as the only positive, while all other behaviours are negative in that interaction obligation. Finally, the universal interpretation describes behaviour that must occur as part of all executions of the system. We have no similar notion to this. As for most work on MSCs, [Krü00] does not include a notion of inherent nondeterminism.

Looking at other specification languages than sequence diagrams, more work has been done on nondeterminism. As the main focus of this paper is on computer systems in relation to sequence diagrams, we refer to [RRS06] for a more general treatment of related work with respect to nondeterminism.

15.8 Conclusions

For sequence diagrams with underspecification and inherent nondeterminism, we have in this paper defined different refinement relations, their corresponding compliance relations, and investigated the mathematical properties of these relations. Which of the refinement and compliance relations will turn out to be most useful in practice, is an open question that should be subject to future research.

Our general compliance checking procedure for relating systems and sequence diagrams was given in section 15.1. Together with the defined refinement and compliance relations, the procedure meets the requirements of Section 15.2 in the following sense:

- a. The procedure is independent of any particular programming language or paradigm. All we require, is that there exists some means to obtain the traces of the system.
- b. The notion of compliance is a special case of refinement. With the exception of restricted compliance, all compliance relations are special cases of the corresponding refinement relations. Whatever refinement relation is used between two sequence diagrams, any system compliant with the refinement is also compliant with the original diagram.
- c. If a system is in compliance with a sequence diagram using the most general compliance relation for sequence diagrams with underspecification, \hookrightarrow_r , the most general compliance relation for sequence diagrams with inherent nondeterminism, \hookrightarrow_g , may be used with the same result. This means that \hookrightarrow_g allows all systems that are allowed by \hookrightarrow_r . For correspondences between the other compliance relations, we refer to Theorem 4 in Section 15.6.
- d. The approach is faithful to the UML 2.1 standard, both with respect to the underlying semantic model using sets of positive and negative traces, and with respect to the semantics given for each concrete operator. In particular, all of our definitions take into account the partial nature of sequence diagrams.

In this paper we have only considered sequence diagrams without external input and output. Our results may be generalized to handle also sequence diagrams with such external communication by in each case defining an adversary representing the environment of the system, and then checking compliance under the assumption of this adversary.

References

- [AEY00] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of message sequence charts. In *Proc. 22nd International Conference on Software Engineering (ICSE 2000)*, pages 304–313. ACM, 2000.
- [BS01] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems : Foundations, Streams, Interfaces, and Refinement*. Springer, 2001.
- [CK04] María Victoria Cengarle and Alexander Knapp. UML 2.0 interactions: Semantics and refinement. In *Proc. 3rd Int. Wsh. Critical Systems Development with UML (CSDUML'04)*, Technical report TUM-I0415, pages 85–99. Institut für Informatik, Technische Universität München, 2004.

- [DH99] Werner Damm and David Harel. LSC's: Breathing life into message sequence charts. In *Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*. Kluwer, 1999.
- [HHR05] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. STAIRS towards formal design with sequence diagrams. *Journal of Software and Systems Modeling*, 22(4):349–458, 2005.
- [HHR06] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309, Department of Informatics, University of Oslo, 2006.
- [HM03] David Harel and Rami Marelly. *Come, Let's Play.: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.
- [HS03] Øystein Haugen and Ketil Stølen. STAIRS — Steps to analyze interactions with refinement semantics. In *Proc. International Conference on UML (UML'2003)*, volume 2863 of *LNCS*, pages 388–402. Springer, 2003.
- [ITU99] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1999.
- [Krü00] Ingolf Heiko Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.
- [OMG06] Object Management Group. *UML 2.1 Superstructure Specification*, document: ptc/06-04-02 edition, 2006.
- [RHS07] Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen. Refining UML interactions with underspecification and nondeterminism. Technical Report 325, Department of Informatics, University of Oslo, 2007.
- [RRS06] Atle Refsdal, Ragnhild Kobro Runde, and Ketil Stølen. Underspecification, inherent nondeterminism and probability in sequence diagrams. In *Proc. 8th IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'06)*, volume 4037 of *LNCS*, pages 138–155. Springer, 2006.
- [RRS07] Atle Refsdal, Ragnhild Kobro Runde, and Ketil Stølen. Relating computer systems to sequence diagrams with underspecification, inherent nondeterminism and probabilistic choice. Part 2: probabilistic choice. Technical Report 347 (to appear), Department of Informatics, University of Oslo, 2007.

15.A Summary of Results

The following tables summarize results with reference to the relevant theorems.

Property	\rightsquigarrow_r	\rightsquigarrow_{rr}
Trans.	Lemma 26 in [HHR06]	Theorem 5
Trans. ref./impl.	Theorem 6	Theorem 7
Mon. w.r.t refuse	Lemma 4 in [RHS07]	Theorem 8
Mon. w.r.t seq	Lemma 30 in [HHR06]	Theorem 9
Mon. w.r.t par	Lemma 31 in [HHR06]	Theorem 10
Mon. w.r.t alt	Theorem 11	Theorem 12

Property	\rightsquigarrow_g	\rightsquigarrow_{rg}
Trans.	Theorem 9 in [HHR06]	Theorem 13
Trans. ref./impl.	Theorem 15	Theorem 15
Mon. w.r.t refuse	Theorem 7 in [RHS07]	Theorem 17
Mon. w.r.t seq	Theorem 13 in [HHR06]	Theorem 18
Mon. w.r.t par	Theorem 14 in [HHR06]	Theorem 19
Mon. w.r.t alt	Theorem 11 in [HHR06]	Theorem 20
Mon. w.r.t xalt	Theorem 12 in [HHR06]	Theorem 21

Property	\rightsquigarrow_l	\rightsquigarrow_{rl}
Trans.	Theorem 6 in [RHS07]	Theorem 14
Trans. ref./impl.	Theorem 16	Theorem 16
Mon. w.r.t refuse	Theorem 8 in [RHS07]	Theorem 22
Mon. w.r.t seq	Theorem 10 in [RHS07]	Theorem 23
Mon. w.r.t par	Theorem 24	Theorem 24
Mon. w.r.t alt	Theorem 11 in [RHS07]	Theorem 25
Mon. w.r.t xalt	Theorem 12 in [RHS07]	Theorem 26

	\mapsto_g	\mapsto_l
\mapsto_r	$\Rightarrow:$ Theorem 27	$\Rightarrow:$ Theorem 29 $\Leftarrow:$ Theorem 30
Correspondence properties:	\mapsto_{rg}	\mapsto_{rl}
\mapsto_{rr}	$\Rightarrow:$ Theorem 28	$\Leftarrow:$ Theorem 31

15.B Proofs

In this section we state and prove each individual theorem. Theorems that are proved in other technical reports are not included.

15.B.1 Specifications with Underspecification

Transitivity

Theorem 5 (Transitivity of \rightsquigarrow_{rr} .) Let d , d' and d'' be sequence diagrams without xalt. Then

$$\llbracket d \rrbracket^u \rightsquigarrow_{rr} \llbracket d' \rrbracket^u \wedge \llbracket d' \rrbracket^u \rightsquigarrow_{rr} \llbracket d'' \rrbracket^u \Rightarrow \llbracket d \rrbracket^u \rightsquigarrow_{rr} \llbracket d'' \rrbracket^u$$

P .

- L : $\llbracket d \rrbracket^u = (p, n)$
- $\llbracket d' \rrbracket^u = (p', n')$
- $\llbracket d'' \rrbracket^u = (p'', n'')$
- $\langle 1 \rangle$ 1. A : 1. $(p, n) \rightsquigarrow_{rr} (p', n')$
2. $(p', n') \rightsquigarrow_{rr} (p'', n'')$
- P : $(p, n) \rightsquigarrow_{rr} (p'', n'')$

$\langle 2 \rangle 1.$ Requirement 1: $(p, n) \rightsquigarrow_r (p'', n'')$
 $\langle 3 \rangle 1.$ $(p, n) \rightsquigarrow_r (p', n')$
 P : $\langle 1 \rangle 1:1$ and definition 15.12 of \rightsquigarrow_{rr} .
 $\langle 3 \rangle 2.$ $(p', n') \rightsquigarrow_r (p'', n'')$
 P : $\langle 1 \rangle 1:2$ and definition 15.12 of \rightsquigarrow_{rr} .
 $\langle 3 \rangle 3.$ Q.E.D.
 P : $\langle 2 \rangle 1, \langle 3 \rangle 2$ and lemma 26 in [HHR06] (transitivity of \rightsquigarrow_r).
 $\langle 2 \rangle 2.$ Requirement 2: $p'' \subseteq p$
 $\langle 3 \rangle 1.$ $p' \subseteq p$
 P : $\langle 1 \rangle 1:1$ and definition 15.12 of \rightsquigarrow_{rr} .
 $\langle 3 \rangle 2.$ $p'' \subseteq p'$
 P : $\langle 1 \rangle 1:2$ and definition 15.12 of \rightsquigarrow_{rr} .
 $\langle 3 \rangle 3.$ Q.E.D.
 P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and transitivity of \subseteq .
 $\langle 2 \rangle 3.$ Q.E.D.
 P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition 15.12 of \rightsquigarrow_{rr} .
 $\langle 1 \rangle 2.$ Q.E.D.
 P : \Rightarrow -rule.

□

Transitivity between refinement and implementation

Theorem 6 (Transitivity between refinement and implementation for \rightsquigarrow_r .) Let d_1 and d_2 be sequence diagrams without xalt . Then

$$\llbracket d_1 \rrbracket^u \rightsquigarrow_r \llbracket d_2 \rrbracket^u \wedge \llbracket d_2 \rrbracket^u \mapsto_r \langle I \rangle_{d_2}^u \Rightarrow \llbracket d_1 \rrbracket^u \mapsto_r \langle I \rangle_{d_1}^u$$

P .

L : $\llbracket d_1 \rrbracket^u = (p_1, n_1)$
 $\llbracket d_2 \rrbracket^u = (p_2, n_2)$

$\langle 1 \rangle 1.$ $\langle I \rangle_{d_1}^u = (\text{traces}(I), \mathcal{H}^{ll(d_1)} \setminus \text{traces}(I))$
 P : Definition 15.13 of $\langle I \rangle_d^u$.
 $\langle 1 \rangle 2.$ $\langle I \rangle_{d_2}^u = (\text{traces}(I), \mathcal{H}^{ll(d_2)} \setminus \text{traces}(I))$
 P : Definition 15.13 of $\langle I \rangle_d^u$.
 $\langle 1 \rangle 3.$ A : 1. $(p_1, n_1) \rightsquigarrow_r (p_2, n_2)$
 2. $(p_2, n_2) \mapsto_r (\text{traces}(I), \mathcal{H}^{ll(d_2)} \setminus \text{traces}(I))$
 i.e. $(p_2, n_2) \rightsquigarrow_r (\text{traces}(I), \mathcal{H}^{ll(d_2)} \setminus \text{traces}(I))$ by definition 15.14 of \mapsto_r .
 P : $(p_1, n_1) \mapsto_r (\text{traces}(I), \mathcal{H}^{ll(d_1)} \setminus \text{traces}(I))$

$\langle 2 \rangle 1.$ Requirement 1: $n_1 \subseteq \mathcal{H}^{ll(d_1)} \setminus \text{traces}(I)$
 $\langle 3 \rangle 1.$ $n_1 \subseteq \mathcal{H}^{ll(d_2)} \setminus \text{traces}(I)$
 P : $\langle 1 \rangle 3:1, \langle 1 \rangle 3:2$, lemma 26 in [HHR06] (transitivity of \rightsquigarrow_r) and definition 15.11
 of \rightsquigarrow_r .
 $\langle 3 \rangle 2.$ $n_1 \subseteq \mathcal{H}^{ll(d_1)}$
 P : $\llbracket d_1 \rrbracket^u = (p_1, n_1)$, definition of $ll(d)$ and definition of \mathcal{H}^L .
 $\langle 3 \rangle 3.$ Q.E.D.

- P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and $A \subseteq B \wedge A \subseteq C \setminus X \Rightarrow A \subseteq B \setminus X$ for arbitrary sets A, B, C and X .
- $\langle 2 \rangle 2.$ Requirement 2: $p_1 \subseteq \text{traces}(I) \cup (\mathcal{H}^{ll(d_1)} \setminus \text{traces}(I))$,
i.e. $p_1 \subseteq \text{traces}(I) \cup \mathcal{H}^{ll(d_1)}$
- P : $p_1 \subseteq \mathcal{H}^{ll(d)}$ by $\llbracket d_1 \rrbracket^u = (p_1, n_1)$, definition of $ll(d)$ and definition of \mathcal{H}^L .
- $\langle 2 \rangle 3.$ $(p_1, n_1) \rightsquigarrow_r (\text{traces}(I), \mathcal{H}^{ll(d_1)} \setminus \text{traces}(I))$
- P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition 15.11 of \rightsquigarrow_r .
- $\langle 2 \rangle 4.$ Q.E.D.
- P : $\langle 2 \rangle 3$ and definition 15.14 of \mapsto_r .
- $\langle 1 \rangle 4.$ Q.E.D.
- P : $\langle 1 \rangle 3$ and \Rightarrow -rule.

□

Theorem 7 (Transitivity between refinement and implementation for \rightsquigarrow_{rr} .) Let d_1 and d_2 be sequence diagrams without xalt. Then

$$\llbracket d_1 \rrbracket^u \rightsquigarrow_{rr} \llbracket d_2 \rrbracket^u \wedge \llbracket d_2 \rrbracket^u \mapsto_{rr} \langle I \rangle_{d_2}^u \Rightarrow \llbracket d_1 \rrbracket^u \mapsto_{rr} \langle I \rangle_{d_1}^u$$

- P .
- L : $\llbracket d_1 \rrbracket^u = (p_1, n_1)$
 $\llbracket d_2 \rrbracket^u = (p_2, n_2)$
- $\langle 1 \rangle 1.$ A : 1. $\llbracket d_1 \rrbracket^u \rightsquigarrow_{rr} \llbracket d_2 \rrbracket^u$
2. $\llbracket d_2 \rrbracket^u \mapsto_{rr} \langle I \rangle_{d_2}^u$
- P : $\llbracket d_1 \rrbracket^u \mapsto_{rr} \langle I \rangle_{d_1}^u$
- $\langle 2 \rangle 1.$ Requirement 1: $\llbracket d_1 \rrbracket^u \mapsto_r \langle I \rangle_{d_1}^u$
- $\langle 3 \rangle 1.$ $\llbracket d_1 \rrbracket^u \rightsquigarrow_r \llbracket d_2 \rrbracket^u$
- P : $\langle 1 \rangle 1:1$ and definition 15.12 of \rightsquigarrow_{rr} .
- $\langle 3 \rangle 2.$ $\llbracket d_2 \rrbracket^u \mapsto_r \langle I \rangle_{d_2}^u$
- P : $\langle 1 \rangle 1:2$ and definition 15.15 of \mapsto_{rr} .
- $\langle 3 \rangle 3.$ Q.E.D.
- P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and theorem 6 (transitivity between refinement and implementation for \rightsquigarrow_r).
- $\langle 2 \rangle 2.$ Requirement 2: $\pi_1(\llbracket d_1 \rrbracket^u) \cap \pi_1(\langle I \rangle_{d_1}^u) \neq \emptyset$
- $\langle 3 \rangle 1.$ $\pi_1(\llbracket d_1 \rrbracket^u) = p_1$
- P : $\llbracket d_1 \rrbracket^u = (p_1, n_1)$ and definition of π_1 .
- $\langle 3 \rangle 2.$ $\pi_1(\langle I \rangle_{d_1}^u) = \text{traces}(I)$
- P : Definition 15.13 of $\langle I \rangle_d$ and definition of π_1 .
- $\langle 3 \rangle 3.$ $p_2 \subseteq p_1$
- P : $\langle 1 \rangle 1:1$ and definition 15.12 of \rightsquigarrow_{rr} .
- $\langle 3 \rangle 4.$ $p_2 \cap \text{traces}(I) \neq \emptyset$
- P : $\langle 1 \rangle 1:2$ and definition 15.15 of \mapsto_{rr} .
- $\langle 3 \rangle 5.$ $p_1 \cap \text{traces}(I) \neq \emptyset$
- P : $\langle 3 \rangle 3, \langle 3 \rangle 4$ and $A \subseteq B \wedge A \cap X \neq \emptyset \Rightarrow B \cap X \neq \emptyset$ for arbitrary sets A, B and X .
- $\langle 3 \rangle 6.$ Q.E.D.

P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and $\langle 3 \rangle 5$.
 $\langle 2 \rangle 3$. Q.E.D.
P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition 15.15 of \mapsto_{rr} .
 $\langle 1 \rangle 2$. Q.E.D.
P : \Rightarrow -rule.

□

Monotonicity

Lemma 1 (*To be used when proving monotonicity with respect to seq.*)

A : a. $s_1 \subseteq s'_1$
b. $s_2 \subseteq s'_2$
P : $s_1 \succsim s_2 \subseteq s'_1 \succsim s'_2$
P .

This is lemma 27 in [HHR06].

□

Lemma 2 (*To be used when proving monotonicity with respect to par.*)

A : a. $s_1 \subseteq s'_1$
b. $s_2 \subseteq s'_2$
P : $s_1 \parallel s_2 \subseteq s'_1 \parallel s'_2$
P .

This is lemma 28 in [HHR06].

□

Theorem 8 (Monotonicity of \rightsquigarrow_{rr} w.r.t refuse.) Let d be a sequence diagram without xalt. Then

$$\llbracket d \rrbracket^u \rightsquigarrow_{rr} \llbracket d' \rrbracket^u \Rightarrow \llbracket \text{refuse } d \rrbracket^u \rightsquigarrow_{rr} \llbracket \text{refuse } d' \rrbracket^u$$

P .

L : $\llbracket d \rrbracket^u = (p, n)$

$\llbracket d' \rrbracket^u = (p', n')$

$\langle 1 \rangle 1$. A : $\llbracket d \rrbracket^u \rightsquigarrow_{rr} \llbracket d' \rrbracket^u$,
i.e. $(p, n) \rightsquigarrow_{rr} (p', n')$

P : $\llbracket \text{refuse } d \rrbracket^u \rightsquigarrow_{rr} \llbracket \text{refuse } d' \rrbracket^u$,

i.e. $(\emptyset, p \cup n) \rightsquigarrow_{rr} (\emptyset, p' \cup n')$ by definition 15.8 of refuse.

$\langle 2 \rangle 1$. Requirement 1: $(\emptyset, p \cup n) \rightsquigarrow_r (\emptyset, p' \cup n')$

$\langle 3 \rangle 1$. $(p, n) \rightsquigarrow_r (p', n')$

P : $\langle 1 \rangle 1$ and definition 15.12 of \rightsquigarrow_{rr} .

$\langle 3 \rangle 2$. Q.E.D.

P : $\langle 3 \rangle 1$ and lemma 4 in [RHS07] (monotonicity of \rightsquigarrow_r w.r.t refuse).

$\langle 2 \rangle 2$. Requirement 2: $\emptyset \subseteq \emptyset$

P : Trivial.

$\langle 2 \rangle 3$. Q.E.D.

P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition 15.12 of \rightsquigarrow_{rr} .

$\langle 1 \rangle 2.$ Q.E.D.

P : \Rightarrow -rule.

□

Theorem 9 (Monotonicity of \rightsquigarrow_{rr} w.r.t seq.) Let d_1, d_2, d'_1 and d'_2 be sequence diagrams without $\times\text{alt}$. Then

$$\llbracket d_1 \rrbracket^u \rightsquigarrow_{rr} \llbracket d'_1 \rrbracket^u \wedge \llbracket d_2 \rrbracket^u \rightsquigarrow_{rr} \llbracket d'_2 \rrbracket^u \Rightarrow \llbracket d_1 \text{ seq } d_2 \rrbracket^u \rightsquigarrow_{rr} \llbracket d'_1 \text{ seq } d'_2 \rrbracket^u$$

P .

$$\begin{aligned} L &: \llbracket d_1 \rrbracket^u = (p_1, n_1) \\ &\quad \llbracket d'_1 \rrbracket^u = (p'_1, n'_1) \\ &\quad \llbracket d_2 \rrbracket^u = (p_2, n_2) \\ &\quad \llbracket d'_2 \rrbracket^u = (p'_2, n'_2) \end{aligned}$$

- $\langle 1 \rangle 1.$ A : 1. $\llbracket d_1 \rrbracket^u \rightsquigarrow_{rr} \llbracket d'_1 \rrbracket^u$,
i.e. $(p_1, n_1) \rightsquigarrow_{rr} (p'_1, n'_1)$
2. $\llbracket d_2 \rrbracket^u \rightsquigarrow_{rr} \llbracket d'_2 \rrbracket^u$,
i.e. $(p_2, n_2) \rightsquigarrow_{rr} (p'_2, n'_2)$

P : $\llbracket d_1 \text{ seq } d_2 \rrbracket^u \rightsquigarrow_{rr} \llbracket d'_1 \text{ seq } d'_2 \rrbracket^u$,
i.e. $(p_1, n_1) \rightsquigarrow_{rr} (p_2, n_2) \rightsquigarrow_{rr} (p'_1, n'_1) \rightsquigarrow_{rr} (p'_2, n'_2)$ by definition 15.7
i.e. $(p_1 \rightsquigarrow_{rr} p_2, n_1 \rightsquigarrow_{rr} p_2 \cup n_1 \rightsquigarrow_{rr} n_2 \cup p_1 \rightsquigarrow_{rr} n_2)$
 $\rightsquigarrow_{rr} (p'_1 \rightsquigarrow_{rr} p'_2, n'_1 \rightsquigarrow_{rr} p'_2 \cup n'_1 \rightsquigarrow_{rr} n'_2 \cup p'_1 \rightsquigarrow_{rr} n'_2)$ by definition 15.4.

$\langle 2 \rangle 1.$ Requirement 1: $(p_1, n_1) \rightsquigarrow_r (p_2, n_2) \rightsquigarrow_r (p'_1, n'_1) \rightsquigarrow_r (p'_2, n'_2)$

$\langle 3 \rangle 1.$ $(p_1, n_1) \rightsquigarrow_r (p'_1, n'_1)$

P : $\langle 1 \rangle 1:1$ and definition 15.12 of \rightsquigarrow_{rr} .

$\langle 3 \rangle 2.$ $(p_2, n_2) \rightsquigarrow_r (p'_2, n'_2)$

P : $\langle 1 \rangle 1:2$ and definition 15.12 of \rightsquigarrow_{rr} .

$\langle 3 \rangle 3.$ Q.E.D.

P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and lemma 30 in [HHR06] (monotonicity of \rightsquigarrow_r w.r.t seq).

$\langle 2 \rangle 2.$ Requirement 2: $(p'_1 \rightsquigarrow_{rr} p'_2) \subseteq (p_1 \rightsquigarrow_{rr} p_2)$

$\langle 3 \rangle 1.$ $p'_1 \subseteq p_1$

P : $\langle 1 \rangle 1:1$ and definition 15.12 of \rightsquigarrow_{rr} .

$\langle 3 \rangle 2.$ $p'_2 \subseteq p_2$

P : $\langle 1 \rangle 1:2$ and definition 15.12 of \rightsquigarrow_{rr} .

$\langle 3 \rangle 3.$ Q.E.D.

P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and lemma 1.

$\langle 2 \rangle 3.$ Q.E.D.

P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition 15.12 of \rightsquigarrow_{rr} .

$\langle 1 \rangle 2.$ Q.E.D.

P : \Rightarrow -rule.

□

Theorem 10 (Monotonicity of \rightsquigarrow_{rr} w.r.t par.) Let d_1, d_2, d'_1 and d'_2 be sequence diagrams without xalt . Then

$$\llbracket d_1 \rrbracket^u \rightsquigarrow_{rr} \llbracket d'_1 \rrbracket^u \wedge \llbracket d_2 \rrbracket^u \rightsquigarrow_{rr} \llbracket d'_2 \rrbracket^u \Rightarrow \llbracket d_1 \text{ par } d_2 \rrbracket^u \rightsquigarrow_{rr} \llbracket d'_1 \text{ par } d'_2 \rrbracket^u$$

P .

$$\begin{aligned} L : \llbracket d_1 \rrbracket^u &= (p_1, n_1) \\ \llbracket d'_1 \rrbracket^u &= (p'_1, n'_1) \\ \llbracket d_2 \rrbracket^u &= (p_2, n_2) \\ \llbracket d'_2 \rrbracket^u &= (p'_2, n'_2) \end{aligned}$$

$$\langle 1 \rangle 1. A : \begin{aligned} 1. \llbracket d_1 \rrbracket^u &\rightsquigarrow_{rr} \llbracket d'_1 \rrbracket^u, \\ &\text{i.e. } (p_1, n_1) \rightsquigarrow_{rr} (p'_1, n'_1) \\ 2. \llbracket d_2 \rrbracket^u &\rightsquigarrow_{rr} \llbracket d'_2 \rrbracket^u, \\ &\text{i.e. } (p_2, n_2) \rightsquigarrow_{rr} (p'_2, n'_2) \end{aligned}$$

$$\begin{aligned} P : \llbracket d_1 \text{ par } d_2 \rrbracket^u &\rightsquigarrow_{rr} \llbracket d'_1 \text{ par } d'_2 \rrbracket^u, \\ &\text{i.e. } (p_1, n_1) \parallel (p_2, n_2) \rightsquigarrow_{rr} (p'_1, n'_1) \parallel (p'_2, n'_2) \text{ by definition 15.6,} \\ &\text{i.e. } (p_1 \parallel p_2, n_1 \parallel p_2 \cup n_1 \parallel n_2 \cup p_1 \parallel n_2) \\ &\rightsquigarrow_{rr} (p'_1 \parallel p'_2, n'_1 \parallel p'_2 \cup n'_1 \parallel n'_2 \cup p'_1 \parallel n'_2) \text{ by definition 15.3.} \end{aligned}$$

$$\langle 2 \rangle 1. \text{ Requirement 1: } (p_1, n_1) \parallel (p_2, n_2) \rightsquigarrow_r (p'_1, n'_1) \parallel (p'_2, n'_2)$$

$$\langle 3 \rangle 1. (p_1, n_1) \rightsquigarrow_r (p'_1, n'_1)$$

P : $\langle 1 \rangle 1:1$ and definition 15.12 of \rightsquigarrow_{rr} .

$$\langle 3 \rangle 2. (p_2, n_2) \rightsquigarrow_r (p'_2, n'_2)$$

P : $\langle 1 \rangle 1:2$ and definition 15.12 of \rightsquigarrow_{rr} .

$$\langle 3 \rangle 3. \text{ Q.E.D.}$$

P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and lemma 31 in [HHR06] (monotonicity of \rightsquigarrow_r w.r.t par).

$$\langle 2 \rangle 2. \text{ Requirement 2: } (p'_1 \parallel p'_2) \subseteq (p_1 \parallel p_2)$$

$$\langle 3 \rangle 1. p'_1 \subseteq p_1$$

P : $\langle 1 \rangle 1:1$ and definition 15.12 of \rightsquigarrow_{rr} .

$$\langle 3 \rangle 2. p'_2 \subseteq p_2$$

P : $\langle 1 \rangle 1:2$ and definition 15.12 of \rightsquigarrow_{rr} .

$$\langle 3 \rangle 3. \text{ Q.E.D.}$$

P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and lemma 2.

$$\langle 2 \rangle 3. \text{ Q.E.D.}$$

P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition 15.12 of \rightsquigarrow_{rr} .

$$\langle 1 \rangle 2. \text{ Q.E.D.}$$

P : \Rightarrow -rule.

□

Theorem 11 (Monotonicity of \rightsquigarrow_r w.r.t alt.) Let d_1, d_2, d'_1 and d'_2 be sequence diagrams without xalt . Then

$$\llbracket d_1 \rrbracket^u \rightsquigarrow_r \llbracket d'_1 \rrbracket^u \wedge \llbracket d_2 \rrbracket^u \rightsquigarrow_r \llbracket d'_2 \rrbracket^u \Rightarrow \llbracket d_1 \text{ alt } d_2 \rrbracket^u \rightsquigarrow_r \llbracket d'_1 \text{ alt } d'_2 \rrbracket^u$$

P .

$$\begin{aligned} L : \llbracket d_1 \rrbracket^u &= (p_1, n_1) \\ \llbracket d'_1 \rrbracket^u &= (p'_1, n'_1) \\ \llbracket d_2 \rrbracket^u &= (p_2, n_2) \\ \llbracket d'_2 \rrbracket^u &= (p'_2, n'_2) \end{aligned}$$

$$\langle 1 \rangle 1. A : \begin{aligned} 1. \llbracket d_1 \rrbracket^u &\rightsquigarrow_r \llbracket d'_1 \rrbracket^u, \\ &\text{i.e. } (p_1, n_1) \rightsquigarrow_r (p'_1, n'_1) \\ 2. \llbracket d_2 \rrbracket^u &\rightsquigarrow_r \llbracket d'_2 \rrbracket^u, \\ &\text{i.e. } (p_2, n_2) \rightsquigarrow_r (p'_2, n'_2) \end{aligned}$$

$$P : \llbracket d_1 \text{ alt } d_2 \rrbracket^u \rightsquigarrow_r \llbracket d'_1 \text{ alt } d'_2 \rrbracket^u, \\ \text{i.e. } (p_1, n_1) \uplus (p_2, n_2) \rightsquigarrow_r (p'_1, n'_1) \uplus (p'_2, n'_2) \text{ by definition 15.9,} \\ \text{i.e. } (p_1 \cup p_2, n_1 \cup n_2) \rightsquigarrow_r (p'_1 \cup p'_2, n'_1 \cup n'_2) \text{ by definition 15.10.}$$

$$\langle 2 \rangle 1. \text{ Requirement 1: } (n_1 \cup n_2) \subseteq (n'_1 \cup n'_2)$$

$$\langle 3 \rangle 1. n_1 \subseteq n'_1 \\ P : \langle 1 \rangle 1:1 \text{ and definition 15.11 of } \rightsquigarrow_r.$$

$$\langle 3 \rangle 2. n_2 \subseteq n'_2 \\ P : \langle 1 \rangle 1:2 \text{ and definition 15.11 of } \rightsquigarrow_r.$$

$$\langle 3 \rangle 3. Q.E.D. \\ P : \langle 3 \rangle 1 \text{ and } \langle 3 \rangle 2.$$

$$\langle 2 \rangle 2. \text{ Requirement 2: } (p_1 \cup p_2) \subseteq ((p'_1 \cup p'_2) \cup (n'_1 \cup n'_2))$$

$$\langle 3 \rangle 1. p_1 \subseteq p'_1 \cup n'_1 \\ P : \langle 1 \rangle 1:1 \text{ and definition 15.11 of } \rightsquigarrow_r.$$

$$\langle 3 \rangle 2. p_2 \subseteq p'_2 \cup n'_2 \\ P : \langle 1 \rangle 1:2 \text{ and definition 15.11 of } \rightsquigarrow_r.$$

$$\langle 3 \rangle 3. Q.E.D. \\ P : \langle 3 \rangle 1, \langle 3 \rangle 2 \text{ and associativity and commutativity of } \cup.$$

$$\langle 2 \rangle 3. Q.E.D.$$

$$P : \langle 2 \rangle 1, \langle 2 \rangle 2 \text{ and definition 15.11 of } \rightsquigarrow_r.$$

$$\langle 1 \rangle 2. Q.E.D.$$

$$P : \Rightarrow\text{-rule.}$$

□

Theorem 12 (Monotonicity of \rightsquigarrow_{rr} w.r.t alt.) Let d_1 , d_2 , d'_1 and d'_2 be sequence diagrams without xalt . Then

$$\llbracket d_1 \rrbracket^u \rightsquigarrow_{rr} \llbracket d'_1 \rrbracket^u \wedge \llbracket d_2 \rrbracket^u \rightsquigarrow_{rr} \llbracket d'_2 \rrbracket^u \Rightarrow \llbracket d_1 \text{ alt } d_2 \rrbracket^u \rightsquigarrow_{rr} \llbracket d'_1 \text{ alt } d'_2 \rrbracket^u$$

P .

$$\begin{aligned} L : \llbracket d_1 \rrbracket^u &= (p_1, n_1) \\ \llbracket d'_1 \rrbracket^u &= (p'_1, n'_1) \\ \llbracket d_2 \rrbracket^u &= (p_2, n_2) \\ \llbracket d'_2 \rrbracket^u &= (p'_2, n'_2) \end{aligned}$$

$$\langle 1 \rangle 1. A : \begin{aligned} 1. \llbracket d_1 \rrbracket^u &\rightsquigarrow_{rr} \llbracket d'_1 \rrbracket^u, \\ &\text{i.e. } (p_1, n_1) \rightsquigarrow_{rr} (p'_1, n'_1) \\ 2. \llbracket d_2 \rrbracket^u &\rightsquigarrow_{rr} \llbracket d'_2 \rrbracket^u, \\ &\text{i.e. } (p_2, n_2) \rightsquigarrow_{rr} (p'_2, n'_2) \end{aligned}$$

$$\begin{aligned} P : \llbracket d_1 \text{ alt } d_2 \rrbracket^u &\rightsquigarrow_{rr} \llbracket d'_1 \text{ alt } d'_2 \rrbracket^u, \\ &\text{i.e. } (p_1, n_1) \uplus (p_2, n_2) \rightsquigarrow_{rr} (p'_1, n'_1) \uplus (p'_2, n'_2) \text{ by definition 15.9,} \\ &\text{i.e. } (p_1 \cup p_2, n_1 \cup n_2) \rightsquigarrow_r (p'_1 \cup p'_2, n'_1 \cup n'_2) \text{ by definition 15.10.} \end{aligned}$$

$$\langle 2 \rangle 1. \text{ Requirement 1: } (p_1, n_1) \uplus (p_2, n_2) \rightsquigarrow_r (p'_1, n'_1) \uplus (p'_2, n'_2)$$

$$\begin{aligned} \langle 3 \rangle 1. (p_1, n_1) &\rightsquigarrow_r (p'_1, n'_1) \\ P : \langle 1 \rangle 1:1 &\text{ and definition 15.12 of } \rightsquigarrow_{rr}. \end{aligned}$$

$$\begin{aligned} \langle 3 \rangle 2. (p_2, n_2) &\rightsquigarrow_r (p'_2, n'_2) \\ P : \langle 1 \rangle 1:2 &\text{ and definition 15.12 of } \rightsquigarrow_{rr}. \end{aligned}$$

$\langle 3 \rangle 3.$ Q.E.D.

P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and theorem 11 (monotonicity of \rightsquigarrow_r w.r.t alt).

$$\langle 2 \rangle 2. \text{ Requirement 2: } (p'_1 \cup p'_2) \subseteq (p_1 \cup p_2)$$

$$\begin{aligned} \langle 3 \rangle 1. p'_1 &\subseteq p_1 \\ P : \langle 1 \rangle 1:1 &\text{ and definition 15.12 of } \rightsquigarrow_{rr}. \end{aligned}$$

$$\begin{aligned} \langle 3 \rangle 2. p'_2 &\subseteq p_2 \\ P : \langle 1 \rangle 1:2 &\text{ and definition 15.12 of } \rightsquigarrow_{rr}. \end{aligned}$$

$\langle 3 \rangle 3.$ Q.E.D.

P : $\langle 3 \rangle 1$ and $\langle 3 \rangle 2$.

$$\langle 2 \rangle 3. \text{ Q.E.D.}$$

P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition 15.12 of \rightsquigarrow_{rr} .

$$\langle 1 \rangle 2. \text{ Q.E.D.}$$

P : \Rightarrow -rule.

□

15.B.2 Specifications with Inherent Nondeterminism

Transitivity

Theorem 13 (Transitivity of \rightsquigarrow_{rg} .) Let d , d' and d'' be sequence diagrams that may contain xalt. Then

$$\llbracket d \rrbracket^i \rightsquigarrow_{rg} \llbracket d' \rrbracket^i \wedge \llbracket d' \rrbracket^i \rightsquigarrow_{rg} \llbracket d'' \rrbracket^i \Rightarrow \llbracket d \rrbracket^i \rightsquigarrow_{rg} \llbracket d'' \rrbracket^i$$

P .

$$\langle 1 \rangle 1. A : 1. \llbracket d \rrbracket^i \rightsquigarrow_{rg} \llbracket d' \rrbracket^i \\ 2. \llbracket d' \rrbracket^i \rightsquigarrow_{rg} \llbracket d'' \rrbracket^i$$

$$P : \llbracket d \rrbracket^i \rightsquigarrow_{rg} \llbracket d'' \rrbracket^i$$

$$\langle 2 \rangle 1. \text{Choose arbitrary } o \in \llbracket d \rrbracket^i$$

P : $\llbracket d \rrbracket^i$ is non-empty for all interactions d .

$$\langle 2 \rangle 2. \text{Choose } o' \in \llbracket d' \rrbracket^i \text{ such that } o \rightsquigarrow_{rr} o'$$

P : $\langle 2 \rangle 1, \langle 1 \rangle 1:1$ and definition 15.21 of \rightsquigarrow_{rg} .

$$\langle 2 \rangle 3. \text{Choose } o'' \in \llbracket d'' \rrbracket^i \text{ such that } o' \rightsquigarrow_{rr} o''$$

P : $\langle 2 \rangle 2, \langle 1 \rangle 1:2$ and definition 15.21 of \rightsquigarrow_{rg} .

$$\langle 2 \rangle 4. o \rightsquigarrow_{rr} o''$$

P : $\langle 2 \rangle 2, \langle 2 \rangle 3$ and theorem 5 (transitivity of \rightsquigarrow_{rr}).

$$\langle 2 \rangle 5. \forall o \in \llbracket d \rrbracket^i : \exists o'' \in \llbracket d'' \rrbracket^i : o \rightsquigarrow_{rr} o''$$

P : $\langle 2 \rangle 1, \langle 2 \rangle 3, \langle 2 \rangle 4$ and \forall -rule.

$$\langle 2 \rangle 6. \text{Q.E.D.}$$

P : $\langle 2 \rangle 5$ and definition 15.21 of \rightsquigarrow_{rg} .

$$\langle 1 \rangle 2. \text{Q.E.D.}$$

P : \Rightarrow -rule.

□

Theorem 14 (Transitivity of \rightsquigarrow_{rl} .) Let d , d' and d'' be sequence diagrams that may contain xalt. Then

$$\llbracket d \rrbracket^i \rightsquigarrow_{rl} \llbracket d' \rrbracket^i \wedge \llbracket d' \rrbracket^i \rightsquigarrow_{rl} \llbracket d'' \rrbracket^i \Rightarrow \llbracket d \rrbracket^i \rightsquigarrow_{rl} \llbracket d'' \rrbracket^i$$

P .

$$\langle 1 \rangle 1. A : 1. \llbracket d \rrbracket^i \rightsquigarrow_{rl} \llbracket d' \rrbracket^i \\ 2. \llbracket d' \rrbracket^i \rightsquigarrow_{rl} \llbracket d'' \rrbracket^i$$

$$P : \llbracket d \rrbracket^i \rightsquigarrow_{rl} \llbracket d'' \rrbracket^i$$

$$\langle 2 \rangle 1. \llbracket d \rrbracket^i \rightsquigarrow_{rg} \llbracket d'' \rrbracket^i$$

P : $\langle 1 \rangle 1:1$ and definition 15.22 of \rightsquigarrow_{rl} .

$$\langle 3 \rangle 2. \llbracket d' \rrbracket^i \rightsquigarrow_{rg} \llbracket d'' \rrbracket^i$$

P : $\langle 1 \rangle 1:2$ and definition 15.22 of \rightsquigarrow_{rl} .

$$\langle 3 \rangle 3. \text{Q.E.D.}$$

P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and theorem 13 (transitivity of \rightsquigarrow_{rg}).

$$\langle 2 \rangle 2. \forall o'' \in \llbracket d'' \rrbracket^i : \exists o \in \llbracket d \rrbracket^i : o \rightsquigarrow_{rr} o'$$

$$\langle 3 \rangle 1. \text{Choose arbitrary } o'' \in \llbracket d'' \rrbracket^i$$

- P : $\llbracket d \rrbracket^i$ is non-empty for all interactions d .
- $\langle 3 \rangle 2.$ Choose $o' \in \llbracket d' \rrbracket^i$ such that $o' \rightsquigarrow_{rr} o''$
- P : $\langle 3 \rangle 1, \langle 1 \rangle 1:1$ and definition 15.22 of \rightsquigarrow_{rl} .
- $\langle 3 \rangle 3.$ Choose $o \in \llbracket d \rrbracket^i$ such that $o \rightsquigarrow_{rr} o'$
- P : $\langle 3 \rangle 2, \langle 1 \rangle 1:2$ and definition 15.22 of \rightsquigarrow_{rl} .
- $\langle 3 \rangle 4.$ $o \rightsquigarrow_{rr} o''$
- P : $\langle 3 \rangle 2, \langle 3 \rangle 3$ and theorem 5 (transitivity of \rightsquigarrow_{rr}).
- $\langle 3 \rangle 5.$ Q.E.D.
- P : $\langle 3 \rangle 1, \langle 3 \rangle 3, \langle 3 \rangle 4$ and \forall -rule.
- $\langle 2 \rangle 3.$ Q.E.D.
- P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition 15.22 of \rightsquigarrow_{rl} .
- $\langle 1 \rangle 2.$ Q.E.D.
- P : \Rightarrow -rule.

□

Transitivity between refinement and implementation

Lemma 3 Let d_1 and d_2 be sequence diagrams that may contain xalt , and h a well-formed trace.

- A : a. $(p, n) \in \llbracket d_1 \rrbracket^i$
- b. $(p', n') \in \llbracket d_2 \rrbracket^i$
- c. $(p, n) \rightsquigarrow_r (p', n')$
- d. $(p', n') \mapsto_r (\{h\}, \mathcal{H}^{ll(d_2)} \setminus \{h\})$
- P : $(p, n) \mapsto_r (\{h\}, \mathcal{H}^{ll(d_1)} \setminus \{h\})$
- $\langle 1 \rangle 1.$ $(p, n) \rightsquigarrow_r (\{h\}, \mathcal{H}^{ll(d_1)} \setminus \{h\})$
- $\langle 2 \rangle 1.$ Requirement 1: $n \subseteq \mathcal{H}^{ll(d_1)} \setminus \{h\}$
- $\langle 3 \rangle 1.$ $n \subseteq \mathcal{H}^{ll(d_2)} \setminus \{h\}$
- $\langle 4 \rangle 1.$ $n \subseteq n'$
- P : Assumption c and definition 15.11 of \rightsquigarrow_r .
- $\langle 4 \rangle 2.$ $n' \subseteq \mathcal{H}^{ll(d_2)} \setminus \{h\}$
- P : Assumption d and definitions 15.11 and 15.14 of \mapsto_r .
- $\langle 4 \rangle 3.$ Q.E.D.
- P : $\langle 4 \rangle 1, \langle 4 \rangle 2$ and transitivity of \subseteq .
- $\langle 3 \rangle 2.$ $n \subseteq \mathcal{H}^{ll(d_1)}$
- P : Assumption a, definition of $ll(d)$ and definition of \mathcal{H}^L .
- $\langle 3 \rangle 3.$ Q.E.D.
- P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and $A \subseteq B \wedge A \subseteq C \setminus X \Rightarrow A \subseteq B \setminus X$ for arbitrary sets A, B, C and X .
- $\langle 2 \rangle 2.$ Requirement 2: $p \subseteq \{h\} \cup (\mathcal{H}^{ll(d_1)} \setminus \{h\})$, i.e. $p \subseteq \{h\} \cup \mathcal{H}^{ll(d_1)}$
- P : $p \subseteq \mathcal{H}^{ll(d_1)}$ by assumption a, definition of $ll(d)$ and definition of \mathcal{H}^L .
- $\langle 2 \rangle 3.$ Q.E.D.
- P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition 15.11 of \rightsquigarrow_r .

$\langle 1 \rangle 2.$ Q.E.D.

P : Definition 15.14 of \mapsto_r .

□

Lemma 4 Let d_1 and d_2 be sequence diagrams that may contain `xalt`, and h a well-formed trace.

- A : a. $(p, n) \in [\![d_1]\!]^i$
 - b. $(p', n') \in [\![d_2]\!]^i$
 - c. $(p, n) \rightsquigarrow_{rr} (p', n')$
 - d. $(p', n') \mapsto_{rr} (\{h\}, \mathcal{H}^{ll(d_2)} \setminus \{h\})$
- P : $(p, n) \mapsto_{rr} (\{h\}, \mathcal{H}^{ll(d_1)} \setminus \{h\})$
- $\langle 1 \rangle 1.$ $(p, n) \rightsquigarrow_r (\{h\}, \mathcal{H}^{ll(d_1)} \setminus \{h\})$
- P : Assumptions a-d and lemma 3.
- $\langle 1 \rangle 2.$ $p \cap \{h\} \neq \emptyset$
- $\langle 2 \rangle 1.$ $p' \subseteq p$
- P : Assumption c and definition 15.12 of \rightsquigarrow_{rr} .
- $\langle 2 \rangle 2.$ $p' \cap \{h\} \neq \emptyset$
- P : Assumption d and definition 15.15 of \mapsto_{rr} .
- $\langle 2 \rangle 3.$ Q.E.D.
- P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and $A \subseteq B \wedge A \cap X \neq \emptyset \Rightarrow B \cap X \neq \emptyset$ for arbitrary sets A, B and X .
- $\langle 1 \rangle 3.$ Q.E.D.
- P : $\langle 1 \rangle 1, \langle 1 \rangle 2$ and definitions 15.14 and 15.15 of \mapsto_{rr} .

□

Theorem 15 (Transitivity between refinement and implementation for $\rightsquigarrow_{(r)g}$) Let d_1 and d_2 be sequence diagrams that may contain `xalt`. Then

$$[\![d_1]\!]^i \rightsquigarrow_{(r)g} [\![d_2]\!]^i \wedge [\![d_2]\!]^i \mapsto_{(r)g} \langle I \rangle_{d_2}^i \Rightarrow [\![d_1]\!]^i \mapsto_{(r)g} \langle I \rangle_{d_1}^i$$

P .

$$\langle 1 \rangle 1. \langle I \rangle_{d_1}^i = \{(\{h\}, \mathcal{H}^{ll(d_1)} \setminus \{h\}) \mid h \in \text{traces}(I)\}$$

P : Definition 15.23 of $\langle I \rangle_d^i$.

$$\langle 1 \rangle 2. \langle I \rangle_{d_2}^i = \{(\{h\}, \mathcal{H}^{ll(d_2)} \setminus \{h\}) \mid h \in \text{traces}(I)\}$$

P : Definition 15.23 of $\langle I \rangle_d^i$.

$$\langle 1 \rangle 3. A : 1. [\![d_1]\!]^i \rightsquigarrow_{(r)g} [\![d_2]\!]^i$$

$$2. [\![d_2]\!]^i \mapsto_{(r)g} \langle I \rangle_{d_2}^i,$$

i.e. $\forall o \in [\![d_2]\!]^i : \exists o' \in \langle I \rangle_{d_2}^i : o \mapsto_{(r)r} o'$ by definition 15.24.

$$P : [\![d_1]\!]^i \mapsto_{(r)g} \langle I \rangle_{d_1}^i,$$

i.e. $\forall o \in [\![d_1]\!]^i : \exists o' \in \langle I \rangle_{d_1}^i : o \mapsto_{(r)r} o'$ by definition 15.24.

$$\langle 2 \rangle 1. \text{ Choose arbitrary } (p, n) \in [\![d_1]\!]^i$$

P : $[\![d]\!]^i$ is non-empty for all interactions d .

$$\langle 2 \rangle 2. \text{ Choose } (p', n') \in [\![d_2]\!]^i \text{ such that } (p, n) \rightsquigarrow_{(r)r} (p', n')$$

- P : $\langle 2 \rangle 1, \langle 1 \rangle 3:1$ and definition 15.21 of $\rightsquigarrow_{(r)g}$.
- $\langle 2 \rangle 3.$ Choose $h \in \text{traces}(I)$ such that $(p', n') \mapsto_{(r)r} (\{h\}, \mathcal{H}^{ll(d_2)} \setminus \{h\})$
- P : $\langle 2 \rangle 2, \langle 1 \rangle 3:2$ and $\langle 1 \rangle 2.$
- $\langle 2 \rangle 4.$ $(p, n) \mapsto_{(r)r} (\{h\}, \mathcal{H}^{ll(d_1)} \setminus \{h\})$
- P : $\langle 2 \rangle 1, \langle 2 \rangle 2, \langle 2 \rangle 3$ and lemma 3 (4).
- $\langle 2 \rangle 5.$ $(\{h\}, \mathcal{H}^{ll(d_1)} \setminus \{h\}) \in \langle I \rangle_{d_1}^i$
- P : $\langle 1 \rangle 1$ and $\langle 2 \rangle 3.$
- $\langle 2 \rangle 6.$ Q.E.D.
- P : $\langle 2 \rangle 1, \langle 2 \rangle 4, \langle 2 \rangle 5$ and \forall -rule.
- $\langle 1 \rangle 4.$ Q.E.D.
- P : $\langle 1 \rangle 3$ and \Rightarrow -rule.

□

Theorem 16 (Transitivity between refinement and implementation for $\rightsquigarrow_{(r)l}$.) Let d_1 and d_2 be sequence diagrams that may contain $xalt$. Then

$$\llbracket d_1 \rrbracket^i \rightsquigarrow_{(r)l} \llbracket d_2 \rrbracket^i \wedge \llbracket d_2 \rrbracket^i \mapsto_{(r)l} \langle I \rangle_{d_2}^i \Rightarrow \llbracket d_1 \rrbracket^i \mapsto_{(r)l} \langle I \rangle_{d_1}^i$$

- P .
- $\langle 1 \rangle 1.$ $\langle I \rangle_{d_1}^i = \{(\{h\}, \mathcal{H}^{ll(d_1)} \setminus \{h\}) \mid h \in \text{traces}(I)\}$
- P : Definition 15.23 of $\langle I \rangle_{d_1}^i.$
- $\langle 1 \rangle 2.$ $\langle I \rangle_{d_2}^i = \{(\{h\}, \mathcal{H}^{ll(d_2)} \setminus \{h\}) \mid h \in \text{traces}(I)\}$
- P : Definition 15.23 of $\langle I \rangle_{d_2}^i.$
- $\langle 1 \rangle 3.$ A : 1. $\llbracket d_1 \rrbracket^i \rightsquigarrow_{(r)l} \llbracket d_2 \rrbracket^i$
 2. $\llbracket d_2 \rrbracket^i \mapsto_{(r)l} \langle I \rangle_{d_2}^i,$
 i.e. $\llbracket d_2 \rrbracket^i \mapsto_{(r)g} \langle I \rangle_{d_2}^i \wedge \forall o' \in \langle I \rangle_{d_2}^i : \exists o \in \llbracket d_2 \rrbracket^i : o \mapsto_{(r)r} o'$ by definition 15.25.
- P : $\llbracket d_1 \rrbracket^i \mapsto_{(r)l} \langle I \rangle_{d_1}^i$
- $\langle 2 \rangle 1.$ $\llbracket d_1 \rrbracket^i \mapsto_{(r)g} \langle I \rangle_{d_1}^i$
- $\langle 3 \rangle 1.$ $\llbracket d_1 \rrbracket^i \rightsquigarrow_{(r)g} \llbracket d_2 \rrbracket^i$
- P : $\langle 1 \rangle 3:1$ and definition 15.22 of $\rightsquigarrow_{(r)l}.$
- $\langle 3 \rangle 2.$ $\llbracket d_2 \rrbracket^i \mapsto_{(r)g} \langle I \rangle_{d_2}^i$
- P : $\langle 1 \rangle 3:2.$
- $\langle 3 \rangle 3.$ Q.E.D.
- P : $\langle 3 \rangle 1, \langle 3 \rangle 2$ and theorem 15 (transitivity between refinement and implementation for $\rightsquigarrow_{(r)g}$).
- $\langle 2 \rangle 2.$ $\forall o' \in \langle I \rangle_{d_1}^i : \exists o \in \llbracket d_1 \rrbracket^i : o \mapsto_{(r)r} o'$
- $\langle 3 \rangle 1.$ Choose arbitrary $o' \in \langle I \rangle_{d_1}^i$
- P : $\langle I \rangle_{d_1}^i$ is non-empty for all real systems $I.$
- $\langle 3 \rangle 2.$ Choose $h \in \text{traces}(I)$ such that $o' = (\{h\}, \mathcal{H}^{ll(d_1)} \setminus \{h\})$
- P : $\langle 3 \rangle 1$ and $\langle 1 \rangle 1.$
- $\langle 3 \rangle 3.$ $(\{h\}, \mathcal{H}^{ll(d_2)} \setminus \{h\}) \in \langle I \rangle_{d_2}^i$
- P : $\langle 3 \rangle 2$ and $\langle 1 \rangle 2.$
- $\langle 3 \rangle 4.$ Choose $(p', n') \in \llbracket d_2 \rrbracket^i$ such that $(p', n') \mapsto_{(r)r} (\{h\}, \mathcal{H}^{ll(d_2)} \setminus \{h\})$

- P : $\langle 3 \rangle 3$ and $\langle 1 \rangle 3:2$.
- $\langle 3 \rangle 5$. Choose $(p, n) \in [\![d_1]\!]^i$ such that $(p, n) \rightsquigarrow_{(r)r} (p', n')$
- P : $\langle 3 \rangle 4, \langle 1 \rangle 3:1$ and definition 15.22 of $\rightsquigarrow_{(r)l}$.
- $\langle 3 \rangle 6$. $(p, n) \mapsto_{(r)r} (\{h\}, \mathcal{H}^{ll(d_1)} \setminus \{h\})$
- P : $\langle 3 \rangle 5, \langle 3 \rangle 4$ and lemma 3 (4).
- $\langle 3 \rangle 7$. Q.E.D.
- P : $\langle 3 \rangle 2, \langle 3 \rangle 5, \langle 3 \rangle 6$ and \forall -rule.
- $\langle 2 \rangle 3$. Q.E.D.
- P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and definition 15.25 of $\mapsto_{(r)l}$.
- $\langle 1 \rangle 4$. Q.E.D.
- P : $\langle 1 \rangle 3$ and \Rightarrow -rule.

□

Monotonicity

Theorem 17 (Monotonicity of \rightsquigarrow_{rg} w.r.t refuse.) Let d be a sequence diagram that may contain xalt. Then

$$[\![d]\!]^i \rightsquigarrow_{rg} [\![d']\!]^i \Rightarrow [\![\text{refuse } d]\!]^i \rightsquigarrow_{rg} [\![\text{refuse } d']\!]^i$$

- P .
- $\langle 1 \rangle 1$. A : $[\![d]\!]^i \rightsquigarrow_{rg} [\![d']\!]^i$
- P : $[\![\text{refuse } d]\!]^i \rightsquigarrow_{rg} [\![\text{refuse } d']\!]^i$
- $\langle 2 \rangle 1$. Choose arbitrary $o = (p, n) \in [\![\text{refuse } d]\!]^i$
- P : $[\![d]\!]^i$ is non-empty for all interactions d .
- $\langle 2 \rangle 2$. Choose $(p_1, n_1) \in [\![d]\!]$ such that $p = \emptyset$ and $n = p_1 \cup n_1$
- P : $\langle 2 \rangle 1$ and definitions 15.5 and 15.8 of refuse.
- $\langle 2 \rangle 3$. Choose $(p'_1, n'_1) \in [\![d']\!]$ such that $(p_1, n_1) \rightsquigarrow_{rr} (p'_1, n'_1)$
- P : $\langle 2 \rangle 2, \langle 1 \rangle 1$ and definition 15.21 of \rightsquigarrow_{rg} .
- $\langle 2 \rangle 4$. $o' = (p', n') = (\emptyset, p'_1 \cup n'_1) \in [\![\text{refuse } d']\!]$
- P : $\langle 2 \rangle 3$ and definitions 15.5 and 15.8 of refuse.
- $\langle 2 \rangle 5$. $(p, n) \rightsquigarrow_{rr} (p', n')$
- P : $\langle 2 \rangle 2, \langle 2 \rangle 3, \langle 2 \rangle 4$ and theorem 8 (monotonicity of \rightsquigarrow_{rr} w.r.t refuse).
- $\langle 2 \rangle 6$. $\forall o \in [\![\text{refuse } d]\!]^i : \exists o' \in [\![\text{refuse } d']\!]^i : o \rightsquigarrow_{rr} o'$
- P : $\langle 2 \rangle 1, \langle 2 \rangle 4, \langle 2 \rangle 5$ and \forall -rule.
- $\langle 2 \rangle 7$. Q.E.D.
- P : $\langle 2 \rangle 6$ and definition 15.21 of \rightsquigarrow_{rg} .
- $\langle 1 \rangle 2$. Q.E.D.
- P : \Rightarrow -rule.

□

Theorem 18 (Monotonicity of \rightsquigarrow_{rg} w.r.t seq.) Let d_1, d_2, d'_1 and d'_2 be sequence diagrams that may contain xalt. Then

$$[\![d_1]\!]^i \rightsquigarrow_{rg} [\![d'_1]\!]^i \wedge [\![d_2]\!]^i \rightsquigarrow_{rg} [\![d'_2]\!]^i \Rightarrow [\![d_1 \text{ seq } d_2]\!]^i \rightsquigarrow_{rg} [\![d'_1 \text{ seq } d'_2]\!]^i$$

P .

$$\langle 1 \rangle 1. A : 1. \llbracket d_1 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_1 \rrbracket^i \\ 2. \llbracket d_2 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_2 \rrbracket^i$$

$$P : \llbracket d_1 \text{ seq } d_2 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_1 \text{ seq } d'_2 \rrbracket^i$$

$$\langle 2 \rangle 1. \text{ Choose arbitrary } o = (p, n) \in \llbracket d_1 \text{ seq } d_2 \rrbracket^i$$

P : $\llbracket d \rrbracket^i$ is non-empty for all interactions d .

$$\langle 2 \rangle 2. \text{ Choose } (p_1, n_1) \in \llbracket d_1 \rrbracket^i \text{ and } (p_2, n_2) \in \llbracket d_2 \rrbracket^i \text{ such that } p = p_1 \succsim p_2 \text{ and } n = n_1 \succsim p_2 \cup n_1 \succsim n_2 \cup p_1 \succsim n_2$$

P : $\langle 2 \rangle 1$ and definitions 15.4, 15.7 and 15.19 of seq.

$$\langle 2 \rangle 3. \text{ Choose } (p'_1, n'_1) \in \llbracket d'_1 \rrbracket^i \text{ such that } (p_1, n_1) \rightsquigarrow_{rr} (p'_1, n'_1)$$

P : $\langle 2 \rangle 2, \langle 1 \rangle 1:1$ and definition 15.21 of \rightsquigarrow_{rg} .

$$\langle 2 \rangle 4. \text{ Choose } (p'_2, n'_2) \in \llbracket d'_2 \rrbracket^i \text{ such that } (p_2, n_2) \rightsquigarrow_{rr} (p'_2, n'_2)$$

P : $\langle 2 \rangle 2, \langle 1 \rangle 1:2$ and definition 15.21 of \rightsquigarrow_{rg} .

$$\langle 2 \rangle 5. o' = (p', n') = (p'_1 \succsim p'_2, n'_1 \succsim p'_2 \cup n'_1 \succsim n'_2 \cup p'_1 \succsim n'_2) \in \llbracket d'_1 \text{ seq } d'_2 \rrbracket^i$$

P : $\langle 2 \rangle 3, \langle 2 \rangle 4$ and definitions 15.4, 15.7 and 15.19 of seq.

$$\langle 2 \rangle 6. (p, n) \rightsquigarrow_{rr} (p', n')$$

P : $\langle 2 \rangle 2, \langle 2 \rangle 3, \langle 2 \rangle 4, \langle 2 \rangle 5$ and theorem 9 (monotonicity of \rightsquigarrow_{rr} w.r.t. seq).

$$\langle 2 \rangle 7. \forall o \in \llbracket d_1 \text{ seq } d_2 \rrbracket^i : \exists o' \in \llbracket d'_1 \text{ seq } d'_2 \rrbracket^i : o \rightsquigarrow_{rr} o'$$

P : $\langle 2 \rangle 1, \langle 2 \rangle 5, \langle 2 \rangle 6$ and \forall -rule.

$$\langle 2 \rangle 8. \text{ Q.E.D.}$$

P : $\langle 2 \rangle 7$ and definition 15.21 of \rightsquigarrow_{rg} .

$$\langle 1 \rangle 2. \text{ Q.E.D.}$$

P : \Rightarrow -rule.

□

Theorem 19 (Monotonicity of \rightsquigarrow_{rg} w.r.t par.) Let d_1, d_2, d'_1 and d'_2 be sequence diagrams that may contain xalt. Then

$$\llbracket d_1 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_1 \rrbracket^i \wedge \llbracket d_2 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_2 \rrbracket^i \Rightarrow \llbracket d_1 \text{ par } d_2 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_1 \text{ par } d'_2 \rrbracket^i$$

P .

$$\langle 1 \rangle 1. A : 1. \llbracket d_1 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_1 \rrbracket^i \\ 2. \llbracket d_2 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_2 \rrbracket^i$$

$$P : \llbracket d_1 \text{ par } d_2 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_1 \text{ par } d'_2 \rrbracket^i$$

$$\langle 2 \rangle 1. \text{ Choose arbitrary } o = (p, n) \in \llbracket d_1 \text{ par } d_2 \rrbracket^i$$

P : $\llbracket d \rrbracket^i$ is non-empty for all interactions d .

$$\langle 2 \rangle 2. \text{ Choose } (p_1, n_1) \in \llbracket d_1 \rrbracket^i \text{ and } (p_2, n_2) \in \llbracket d_2 \rrbracket^i \text{ such that } p = p_1 \parallel p_2 \text{ and } n = n_1 \parallel p_2 \cup n_1 \parallel n_2 \cup p_1 \parallel n_2$$

P : $\langle 2 \rangle 1$ and definitions 15.3, 15.6 and 15.19 of par.

$$\langle 2 \rangle 3. \text{ Choose } (p'_1, n'_1) \in \llbracket d'_1 \rrbracket^i \text{ such that } (p_1, n_1) \rightsquigarrow_{rr} (p'_1, n'_1)$$

P : $\langle 2 \rangle 2, \langle 1 \rangle 1:1$ and definition 15.21 of \rightsquigarrow_{rg} .

$$\langle 2 \rangle 4. \text{ Choose } (p'_2, n'_2) \in \llbracket d'_2 \rrbracket^i \text{ such that } (p_2, n_2) \rightsquigarrow_{rr} (p'_2, n'_2)$$

P : $\langle 2 \rangle 2, \langle 1 \rangle 1:2$ and definition 15.21 of \rightsquigarrow_{rg} .

$$\langle 2 \rangle 5. o' = (p', n') = (p'_1 \parallel p'_2, n'_1 \parallel p'_2 \cup n'_1 \parallel n'_2 \cup p'_1 \parallel n'_2) \in \llbracket d'_1 \text{ par } d'_2 \rrbracket^i$$

- P : $\langle 2 \rangle 3, \langle 2 \rangle 4$ and definitions 15.3, 15.6 and 15.19 of par.
- $\langle 2 \rangle 6. (p, n) \rightsquigarrow_{rr} (p', n')$
- P : $\langle 2 \rangle 2, \langle 2 \rangle 3, \langle 2 \rangle 4, \langle 2 \rangle 5$ and theorem 10 (monotonicity of \rightsquigarrow_{rr} w.r.t. par).
- $\langle 2 \rangle 7. \forall o \in [\![d_1 \text{ par } d_2]\!]^i : \exists o' \in [\![d'_1 \text{ par } d'_2]\!]^i : o \rightsquigarrow_{rr} o'$
- P : $\langle 2 \rangle 1, \langle 2 \rangle 5, \langle 2 \rangle 6$ and \forall -rule.
- $\langle 2 \rangle 8. \text{Q.E.D.}$
- P : $\langle 2 \rangle 7$ and definition 15.21 of \rightsquigarrow_{rg} .
- $\langle 1 \rangle 2. \text{Q.E.D.}$
- P : \Rightarrow -rule.

□

Theorem 20 (Monotonicity of \rightsquigarrow_{rg} w.r.t alt.) Let d_1, d_2, d'_1 and d'_2 be sequence diagrams that may contain xalt. Then

$$[\![d_1]\!]^i \rightsquigarrow_{rg} [\![d'_1]\!]^i \wedge [\![d_2]\!]^i \rightsquigarrow_{rg} [\![d'_2]\!]^i \Rightarrow [\![d_1 \text{ alt } d_2]\!]^i \rightsquigarrow_{rg} [\![d'_1 \text{ alt } d'_2]\!]^i$$

- P .
- $\langle 1 \rangle 1. A : 1. [\![d_1]\!]^i \rightsquigarrow_{rg} [\![d'_1]\!]^i$
- $2. [\![d_2]\!]^i \rightsquigarrow_{rg} [\![d'_2]\!]^i$
- P : $[\![d_1 \text{ alt } d_2]\!]^i \rightsquigarrow_{rg} [\![d'_1 \text{ alt } d'_2]\!]^i$
- $\langle 2 \rangle 1. \text{Choose arbitrary } o = (p, n) \in [\![d_1 \text{ alt } d_2]\!]^i$
- P : $[\![d]\!]^i$ is non-empty for all interactions d .
- $\langle 2 \rangle 2. \text{Choose } (p_1, n_1) \in [\![d_1]\!]^i \text{ and } (p_2, n_2) \in [\![d_2]\!]^i \text{ such that } p = p_1 \cup p_2 \text{ and } n = n_1 \cup n_2$
- P : $\langle 2 \rangle 1$ and definitions 15.9, 15.10 and 15.19 of alt.
- $\langle 2 \rangle 3. \text{Choose } (p'_1, n'_1) \in [\![d'_1]\!]^i \text{ such that } (p_1, n_1) \rightsquigarrow_{rr} (p'_1, n'_1)$
- P : $\langle 2 \rangle 2, \langle 1 \rangle 1:1$ and definition 15.21 of \rightsquigarrow_{rg} .
- $\langle 2 \rangle 4. \text{Choose } (p'_2, n'_2) \in [\![d'_2]\!]^i \text{ such that } (p_2, n_2) \rightsquigarrow_{rr} (p'_2, n'_2)$
- P : $\langle 2 \rangle 2, \langle 1 \rangle 1:2$ and definition 15.21 of \rightsquigarrow_{rg} .
- $\langle 2 \rangle 5. o' = (p', n') = (p'_1 \cup p'_2, n'_1 \cup n'_2) \in [\![d'_1 \text{ alt } d'_2]\!]^i$
- P : $\langle 2 \rangle 3, \langle 2 \rangle 4$ and definitions 15.9, 15.10 and 15.19 of alt.
- $\langle 2 \rangle 6. (p, n) \rightsquigarrow_{rr} (p', n')$
- P : $\langle 2 \rangle 2, \langle 2 \rangle 3, \langle 2 \rangle 4, \langle 2 \rangle 5$ and theorem 12 (monotonicity of \rightsquigarrow_{rr} w.r.t. alt).
- $\langle 2 \rangle 7. \forall o \in [\![d_1 \text{ alt } d_2]\!]^i : \exists o' \in [\![d'_1 \text{ alt } d'_2]\!]^i : o \rightsquigarrow_{rr} o'$
- P : $\langle 2 \rangle 1, \langle 2 \rangle 5, \langle 2 \rangle 6$ and \forall -rule.
- $\langle 2 \rangle 8. \text{Q.E.D.}$
- P : $\langle 2 \rangle 7$ and definition 15.21 of \rightsquigarrow_{rg} .
- $\langle 1 \rangle 2. \text{Q.E.D.}$
- P : \Rightarrow -rule.

□

Theorem 21 (Monotonicity of \rightsquigarrow_{rg} w.r.t xalt.) Let d_1, d_2, d'_1 and d'_2 be sequence diagrams that may contain xalt. Then

$$[\![d_1]\!]^i \rightsquigarrow_{rg} [\![d'_1]\!]^i \wedge [\![d_2]\!]^i \rightsquigarrow_{rg} [\![d'_2]\!]^i \Rightarrow [\![d_1 \text{ xalt } d_2]\!]^i \rightsquigarrow_{rg} [\![d'_1 \text{ xalt } d'_2]\!]^i$$

P .

- $\langle 1 \rangle 1.$ A : 1. $\llbracket d_1 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_1 \rrbracket^i$
 2. $\llbracket d_2 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_2 \rrbracket^i$

P : $\llbracket d_1 \text{xalt } d_2 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_1 \text{xalt } d'_2 \rrbracket^i$

- $\langle 2 \rangle 1.$ Choose arbitrary $o \in \llbracket d_1 \text{xalt } d_2 \rrbracket^i$

P : $\llbracket d \rrbracket^i$ is non-empty for all interactions d .

- $\langle 2 \rangle 2.$ $\exists o' \in \llbracket d'_1 \text{xalt } d'_2 \rrbracket^i : o \rightsquigarrow_{rr} o'$

$\langle 3 \rangle 1.$ C : $o \in \llbracket d_1 \rrbracket^i$

- $\langle 4 \rangle 1.$ Choose $o' \in \llbracket d'_1 \rrbracket^i$ such that $o \rightsquigarrow_{rr} o'$

P : $\langle 2 \rangle 1, \langle 1 \rangle 1:1$ and definition 15.21 of \rightsquigarrow_{rg} .

- $\langle 4 \rangle 2.$ $o' \in \llbracket d'_1 \text{xalt } d'_2 \rrbracket^i$

P : $\langle 4 \rangle 1$ and definition 15.18 of xalt.

$\langle 4 \rangle 3.$ Q.E.D.

$\langle 3 \rangle 2.$ C : $o \in \llbracket d_2 \rrbracket^i$

- $\langle 4 \rangle 1.$ Choose $o' \in \llbracket d'_2 \rrbracket^i$ such that $o \rightsquigarrow_{rr} o'$

P : $\langle 2 \rangle 1, \langle 1 \rangle 1:2$ and definition 15.21 of \rightsquigarrow_{rg} .

- $\langle 4 \rangle 2.$ $o' \in \llbracket d'_1 \text{xalt } d'_2 \rrbracket^i$

P : $\langle 4 \rangle 1$ and definition 15.18 of xalt.

$\langle 4 \rangle 3.$ Q.E.D.

$\langle 3 \rangle 3.$ Q.E.D.

P : The cases are exhaustive by $\langle 2 \rangle 1$ and definition 15.18 of xalt.

- $\langle 2 \rangle 3.$ $\forall o \in \llbracket d_1 \text{xalt } d_2 \rrbracket^i : \exists o' \in \llbracket d'_1 \text{xalt } d'_2 \rrbracket^i : o \rightsquigarrow_{rr} o'$

P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and \forall -rule.

- $\langle 2 \rangle 4.$ Q.E.D.

P : $\langle 2 \rangle 3$ and definition 15.21 of \rightsquigarrow_{rg} .

- $\langle 1 \rangle 2.$ Q.E.D.

P : \Rightarrow -rule.

□

Theorem 22 (Monotonicity of \rightsquigarrow_{rl} w.r.t refuse.) Let d be a sequence diagram that may contain xalt. Then

$$\llbracket d \rrbracket^i \rightsquigarrow_{rl} \llbracket d' \rrbracket^i \Rightarrow \llbracket \text{refuse } d \rrbracket^i \rightsquigarrow_{rl} \llbracket \text{refuse } d' \rrbracket^i$$

P .

- $\langle 1 \rangle 1.$ A : $\llbracket d \rrbracket^i \rightsquigarrow_{rl} \llbracket d' \rrbracket^i$

P : $\llbracket \text{refuse } d \rrbracket^i \rightsquigarrow_{rl} \llbracket \text{refuse } d' \rrbracket^i$

- $\langle 2 \rangle 1.$ $\llbracket \text{refuse } d \rrbracket^i \rightsquigarrow_{rg} \llbracket \text{refuse } d' \rrbracket^i$

P : $\langle 1 \rangle 1$, definition 15.22 of \rightsquigarrow_{rl} and theorem 17 (monotonicity of \rightsquigarrow_{rg} with respect to refuse).

- $\langle 2 \rangle 2.$ Choose arbitrary $o' = (p', n') \in \llbracket \text{refuse } d' \rrbracket^i$

P : $\llbracket d \rrbracket^i$ is non-empty for all interaction d .

- $\langle 2 \rangle 3.$ Choose $(p'_1, n'_1) \in \llbracket d' \rrbracket^i$ such that $p' = \emptyset$ and $n' = p'_1 \cup n'_1$

P : $\langle 2 \rangle 2$ and definitions 15.8 and 15.20 of refuse.

- $\langle 2 \rangle 4.$ Choose $(p_1, n_1) \in \llbracket d \rrbracket^i$ such that $(p_1, n_1) \rightsquigarrow_{rr} (p'_1, n'_1)$

- P : $\langle 2 \rangle 3, \langle 1 \rangle 1$ and definition 15.22 of \rightsquigarrow_{rl} .
- $\langle 2 \rangle 5. o = (p, n) = (\emptyset, p_1 \cup n_1) \in [\text{refuse } d]^i$
- P : $\langle 2 \rangle 4$ and definitions 15.8 and 15.20 of refuse.
- $\langle 2 \rangle 6. (p, n) \rightsquigarrow_{rr} (p', n')$
- P : $\langle 2 \rangle 3, \langle 2 \rangle 4, \langle 2 \rangle 5$ and theorem 8 (monotonicity of \rightsquigarrow_{rr} w.r.t. refuse).
- $\langle 2 \rangle 7. \forall o' \in [\text{refuse } d']^i : \exists o \in [\text{refuse } d]^i : o \rightsquigarrow_{rr} o'$
- P : $\langle 2 \rangle 2, \langle 2 \rangle 5, \langle 2 \rangle 6$ and \forall -rule.
- $\langle 2 \rangle 8. \text{Q.E.D.}$
- P : $\langle 2 \rangle 1, \langle 2 \rangle 7$ and definition 15.22 of \rightsquigarrow_{rl} .
- $\langle 1 \rangle 2. \text{Q.E.D.}$
- P : \Rightarrow -rule.

□

Theorem 23 (Monotonicity of \rightsquigarrow_{rl} w.r.t seq.) Let d_1, d_2, d'_1 and d'_2 be sequence diagrams that may contain xalt . Then

$$[\text{d}_1]^i \rightsquigarrow_{rl} [\text{d}'_1]^i \wedge [\text{d}_2]^i \rightsquigarrow_{rl} [\text{d}'_2]^i \Rightarrow [\text{d}_1 \text{seq } d_2]^i \rightsquigarrow_{rl} [\text{d}'_1 \text{seq } d'_2]^i$$

- P .
- $\langle 1 \rangle 1. A : 1. [\text{d}_1]^i \rightsquigarrow_{rl} [\text{d}'_1]^i$
2. $[\text{d}_2]^i \rightsquigarrow_{rl} [\text{d}'_2]^i$
- P : $[\text{d}_1 \text{seq } d_2]^i \rightsquigarrow_{rl} [\text{d}'_1 \text{seq } d'_2]^i$
- $\langle 2 \rangle 1. [\text{d}_1 \text{seq } d_2]^i \rightsquigarrow_{rg} [\text{d}'_1 \text{seq } d'_2]^i$
- P : $\langle 1 \rangle 1:1, \langle 1 \rangle 1:2$, definition 15.22 of \rightsquigarrow_{rl} and theorem 18 (monotonicity of \rightsquigarrow_{rg} with respect to seq).
- $\langle 2 \rangle 2. \text{Choose arbitrary } o' = (p', n') \in [\text{d}'_1 \text{seq } d'_2]^i$
- P : $[\text{d}]^i$ is non-empty for all interactions d .
- $\langle 2 \rangle 3. \text{Choose } (p'_1, n'_1) \in [\text{d}'_1]^i \text{ and } (p'_2, n'_2) \in [\text{d}'_2]^i \text{ such that } p' = p'_1 \succsim p'_2 \text{ and } n' = n'_1 \succsim p'_2 \cup n'_1 \succsim n'_2 \cup p'_1 \succsim n'_2$
- P : $\langle 2 \rangle 2$ and definitions 15.4, 15.7 and 15.19 of seq.
- $\langle 2 \rangle 4. \text{Choose } (p_1, n_1) \in [\text{d}_1]^i \text{ such that } (p_1, n_1) \rightsquigarrow_{rr} (p'_1, n'_1)$
- P : $\langle 2 \rangle 3, \langle 1 \rangle 1:1$ and definition 15.22 of \rightsquigarrow_{rl} .
- $\langle 2 \rangle 5. \text{Choose } (p_2, n_2) \in [\text{d}_2]^i \text{ such that } (p_2, n_2) \rightsquigarrow_{rr} (p'_2, n'_2)$
- P : $\langle 2 \rangle 3, \langle 1 \rangle 1:2$ and definition 15.22 of \rightsquigarrow_{rl} .
- $\langle 2 \rangle 6. o = (p, n) = (p_1 \succsim p_2, n_1 \succsim p_2 \cup n_1 \succsim n_2 \cup p_1 \succsim n_2) \in [\text{d}_1 \text{seq } d_2]^i$
- P : $\langle 2 \rangle 4, \langle 2 \rangle 5$ and definitions 15.4, 15.7 and 15.19 of seq.
- $\langle 2 \rangle 7. (p, n) \rightsquigarrow_{rr} (p', n')$
- P : $\langle 2 \rangle 3, \langle 2 \rangle 4, \langle 2 \rangle 5, \langle 2 \rangle 6$ and theorem 9 (monotonicity of \rightsquigarrow_{rr} w.r.t. seq).
- $\langle 2 \rangle 8. \forall o' \in [\text{d}'_1 \text{seq } d'_2]^i : \exists o \in [\text{d}_1 \text{seq } d_2]^i : o \rightsquigarrow_{rr} o'$
- P : $\langle 2 \rangle 2, \langle 2 \rangle 6, \langle 2 \rangle 7$ and \forall -rule.
- $\langle 2 \rangle 9. \text{Q.E.D.}$
- P : $\langle 2 \rangle 1, \langle 2 \rangle 8$ and definition 15.22 of \rightsquigarrow_{rl} .
- $\langle 1 \rangle 2. \text{Q.E.D.}$
- P : \Rightarrow -rule.

□

Theorem 24 (Monotonicity of $\rightsquigarrow_{(r)l}$ w.r.t par.) Let d_1, d_2, d'_1 and d'_2 be sequence diagrams that may contain xalt . Then

$$\llbracket d_1 \rrbracket^i \rightsquigarrow_{(r)l} \llbracket d'_1 \rrbracket^i \wedge \llbracket d_2 \rrbracket^i \rightsquigarrow_{(r)l} \llbracket d'_2 \rrbracket^i \Rightarrow \llbracket d_1 \text{ par } d_2 \rrbracket^i \rightsquigarrow_{(r)l} \llbracket d'_1 \text{ par } d'_2 \rrbracket^i$$

P .

$$\langle 1 \rangle 1. A : 1. \llbracket d_1 \rrbracket^i \rightsquigarrow_{(r)l} \llbracket d'_1 \rrbracket^i \\ 2. \llbracket d_2 \rrbracket^i \rightsquigarrow_{(r)l} \llbracket d'_2 \rrbracket^i$$

$$P : \llbracket d_1 \text{ par } d_2 \rrbracket^i \rightsquigarrow_{(r)l} \llbracket d'_1 \text{ par } d'_2 \rrbracket^i$$

$$\langle 2 \rangle 1. \llbracket d_1 \text{ par } d_2 \rrbracket^i \rightsquigarrow_{(r)g} \llbracket d'_1 \text{ par } d'_2 \rrbracket^i$$

P : $\langle 1 \rangle 1:1, \langle 1 \rangle 1:2$, definition 15.22 of $\rightsquigarrow_{(r)l}$ and theorem 14 in [HHR06]/theorem 19 (monotonicity of $\rightsquigarrow_{(r)g}$ with respect to par).

$$\langle 2 \rangle 2. \text{ Choose arbitrary } o' = (p', n') \in \llbracket d'_1 \text{ par } d'_2 \rrbracket^i$$

P : $\llbracket d \rrbracket^i$ is non-empty for all interactions d .

$$\langle 2 \rangle 3. \text{ Choose } (p'_1, n'_1) \in \llbracket d'_1 \rrbracket^i \text{ and } (p'_2, n'_2) \in \llbracket d'_2 \rrbracket^i \text{ such that } p' = p'_1 \parallel p'_2 \text{ and } n' = n'_1 \parallel p'_2 \cup n'_1 \parallel n'_2 \cup p'_1 \parallel n'_2$$

P : $\langle 2 \rangle 2$ and definitions 15.3, 15.6 and 15.19 of par .

$$\langle 2 \rangle 4. \text{ Choose } (p_1, n_1) \in \llbracket d_1 \rrbracket^i \text{ such that } (p_1, n_1) \rightsquigarrow_{(r)r} (p'_1, n'_1)$$

P : $\langle 2 \rangle 3, \langle 1 \rangle 1:1$ and definition 15.22 of $\rightsquigarrow_{(r)l}$.

$$\langle 2 \rangle 5. \text{ Choose } (p_2, n_2) \in \llbracket d_2 \rrbracket^i \text{ such that } (p_2, n_2) \rightsquigarrow_{(r)r} (p'_2, n'_2)$$

P : $\langle 2 \rangle 3, \langle 1 \rangle 1:2$ and definition 15.22 of $\rightsquigarrow_{(r)l}$.

$$\langle 2 \rangle 6. o = (p, n) = (p_1 \parallel p_2, n_1 \parallel p_2 \cup n_1 \parallel n_2 \cup p_1 \parallel n_2) \in \llbracket d_1 \text{ par } d_2 \rrbracket^i$$

P : $\langle 2 \rangle 4, \langle 2 \rangle 5$ and definitions 15.3, 15.6 and 15.19 of par .

$$\langle 2 \rangle 7. (p, n) \rightsquigarrow_{(r)r} (p', n')$$

P : $\langle 2 \rangle 3, \langle 2 \rangle 4, \langle 2 \rangle 5, \langle 2 \rangle 6$ and lemma 31 in [HHR06]/theorem 10 (monotonicity of $\rightsquigarrow_{(r)r}$ w.r.t. par).

$$\langle 2 \rangle 8. \forall o' \in \llbracket d'_1 \text{ par } d'_2 \rrbracket^i : \exists o \in \llbracket d_1 \text{ par } d_2 \rrbracket^i : o \rightsquigarrow_{(r)r} o'$$

P : $\langle 2 \rangle 2, \langle 2 \rangle 6, \langle 2 \rangle 7$ and \forall -rule.

$$\langle 2 \rangle 9. Q.E.D.$$

P : $\langle 2 \rangle 1, \langle 2 \rangle 8$ and definition 15.22 of $\rightsquigarrow_{(r)l}$.

$$\langle 1 \rangle 2. Q.E.D.$$

P : \Rightarrow -rule.

□

Theorem 25 (Monotonicity of \rightsquigarrow_{rl} w.r.t alt.) Let d_1, d_2, d'_1 and d'_2 be sequence diagrams that may contain xalt . Then

$$\llbracket d_1 \rrbracket^i \rightsquigarrow_{rl} \llbracket d'_1 \rrbracket^i \wedge \llbracket d_2 \rrbracket^i \rightsquigarrow_{rl} \llbracket d'_2 \rrbracket^i \Rightarrow \llbracket d_1 \text{ alt } d_2 \rrbracket^i \rightsquigarrow_{rl} \llbracket d'_1 \text{ alt } d'_2 \rrbracket^i$$

P .

$$\langle 1 \rangle 1. A : 1. \llbracket d_1 \rrbracket^i \rightsquigarrow_{rl} \llbracket d'_1 \rrbracket^i \\ 2. \llbracket d_2 \rrbracket^i \rightsquigarrow_{rl} \llbracket d'_2 \rrbracket^i$$

$$P : \llbracket d_1 \text{ alt } d_2 \rrbracket^i \rightsquigarrow_{rl} \llbracket d'_1 \text{ alt } d'_2 \rrbracket^i$$

$$\langle 2 \rangle 1. \llbracket d_1 \text{ alt } d_2 \rrbracket^i \rightsquigarrow_{rg} \llbracket d'_1 \text{ alt } d'_2 \rrbracket^i$$

- P : $\langle 1 \rangle 1:1, \langle 1 \rangle 1:2$, definition 15.22 of \rightsquigarrow_{rl} and theorem 20 (monotonicity of \rightsquigarrow_{rg} with respect to alt).
- $\langle 2 \rangle 2.$ Choose arbitrary $o' = (p', n') \in [\![d'_1 \text{ alt } d'_2]\!]^i$
- P : $[\![d]\!]^i$ is non-empty for all interactions d .
- $\langle 2 \rangle 3.$ Choose $(p'_1, n'_1) \in [\![d'_1]\!]^i$ and $(p'_2, n'_2) \in [\![d'_2]\!]^i$ such that $p' = p'_1 \cup p'_2$ and $n' = n'_1 \cup n'_2$
- P : $\langle 2 \rangle 2$ and definitions 15.9, 15.10 and 15.19 of alt.
- $\langle 2 \rangle 4.$ Choose $(p_1, n_1) \in [\![d_1]\!]^i$ such that $(p_1, n_1) \rightsquigarrow_{rr} (p'_1, n'_1)$
- P : $\langle 2 \rangle 3, \langle 1 \rangle 1:1$ and definition 15.22 of \rightsquigarrow_{rl} .
- $\langle 2 \rangle 5.$ Choose $(p_2, n_2) \in [\![d_2]\!]^i$ such that $(p_2, n_2) \rightsquigarrow_{rr} (p'_2, n'_2)$
- P : $\langle 2 \rangle 3, \langle 1 \rangle 1:2$ and definition 15.22 of \rightsquigarrow_{rl} .
- $\langle 2 \rangle 6.$ $o = (p, n) = (p_1 \cup p_2, n_1 \cup n_2) \in [\![d_1 \text{ alt } d_2]\!]^i$
- P : $\langle 2 \rangle 4, \langle 2 \rangle 5$ and definitions 15.9, 15.10 and 15.19 of alt.
- $\langle 2 \rangle 7.$ $(p, n) \rightsquigarrow_{rr} (p', n')$
- P : $\langle 2 \rangle 3, \langle 2 \rangle 4, \langle 2 \rangle 5, \langle 2 \rangle 6$ and theorem 12 (monotonicity of \rightsquigarrow_{rr} w.r.t. alt).
- $\langle 2 \rangle 8.$ $\forall o' \in [\![d'_1 \text{ alt } d'_2]\!]^i : \exists o \in [\![d_1 \text{ alt } d_2]\!]^i : o \rightsquigarrow_{rr} o'$
- P : $\langle 2 \rangle 2, \langle 2 \rangle 6, \langle 2 \rangle 7$ and \forall -rule.
- $\langle 2 \rangle 9.$ Q.E.D.
- P : $\langle 2 \rangle 1, \langle 2 \rangle 8$ and definition 15.22 of \rightsquigarrow_{rl} .
- $\langle 1 \rangle 2.$ Q.E.D.
- P : \Rightarrow -rule.

□

Theorem 26 (Monotonicity of \rightsquigarrow_{rl} w.r.t xalt.) Let d_1, d_2, d'_1 and d'_2 be sequence diagrams that may contain xalt. Then

$$[\![d_1]\!]^i \rightsquigarrow_{rl} [\![d'_1]\!]^i \wedge [\![d_2]\!]^i \rightsquigarrow_{rl} [\![d'_2]\!]^i \Rightarrow [\![d_1 \text{ xalt } d_2]\!]^i \rightsquigarrow_{rl} [\![d'_1 \text{ xalt } d'_2]\!]^i$$

- P .
- $\langle 1 \rangle 1.$ A : 1. $[\![d_1]\!]^i \rightsquigarrow_{rl} [\![d'_1]\!]^i$
2. $[\![d_2]\!]^i \rightsquigarrow_{rl} [\![d'_2]\!]^i$
- P : $[\![d_1 \text{ xalt } d_2]\!]^i \rightsquigarrow_{rl} [\![d'_1 \text{ xalt } d'_2]\!]^i$
- $\langle 2 \rangle 1.$ $[\![d_1 \text{ xalt } d_2]\!]^i \rightsquigarrow_{rg} [\![d'_1 \text{ xalt } d'_2]\!]^i$
- P : $\langle 1 \rangle 1:1, \langle 1 \rangle 1:2$, definition 15.22 of \rightsquigarrow_{rl} and theorem 21 (monotonicity of \rightsquigarrow_{rg} with respect to xalt).
- $\langle 2 \rangle 2.$ Choose arbitrary $o' \in [\![d'_1 \text{ xalt } d'_2]\!]^i$
- P : $[\![d]\!]^i$ is non-empty for all interactions d .
- $\langle 2 \rangle 3.$ $\exists o \in [\![d_1 \text{ xalt } d_2]\!]^i : o \rightsquigarrow_{rr} o'$
- $\langle 3 \rangle 1.$ C : $o' \in [\![d'_1]\!]^i$
- $\langle 4 \rangle 1.$ Choose $o \in [\![d_1]\!]^i$ such that $o \rightsquigarrow_{rr} o'$
- P : $\langle 2 \rangle 2, \langle 1 \rangle 1:1$ and definition 15.22 of \rightsquigarrow_{rl} .
- $\langle 4 \rangle 2.$ $o \in [\![d_1 \text{ xalt } d_2]\!]^i$
- P : $\langle 4 \rangle 1$ and definition 15.18 of xalt.
- $\langle 4 \rangle 3.$ Q.E.D.
- $\langle 3 \rangle 2.$ C : $o' \in [\![d'_2]\!]^i$
- $\langle 4 \rangle 1.$ Choose $o \in [\![d_2]\!]^i$ such that $o \rightsquigarrow_{rr} o'$

P : $\langle 2 \rangle 2, \langle 1 \rangle 1:2$ and definition 15.22 of \rightsquigarrow_{rl} .

$\langle 4 \rangle 2.$ $o \in [\![d_1 \text{xalt } d_2]\!]^i$

P : $\langle 4 \rangle 1$ and definition 15.18 of xalt.

$\langle 4 \rangle 3.$ Q.E.D.

$\langle 3 \rangle 3.$ Q.E.D.

P : The cases are exhaustive by $\langle 2 \rangle 2$ and definition 15.18 of xalt.

$\langle 2 \rangle 4.$ $\forall o' \in [\![d'_1 \text{xalt } d'_2]\!]^i : \exists o \in [\![d_1 \text{xalt } d_2]\!]^i : o \rightsquigarrow_{rr} o'$

P : $\langle 2 \rangle 2, \langle 2 \rangle 3$ and \forall -rule.

$\langle 2 \rangle 5.$ Q.E.D.

P : $\langle 2 \rangle 1, \langle 2 \rangle 4$ and definition 15.22 of \rightsquigarrow_{rl} .

$\langle 1 \rangle 2.$ Q.E.D.

P : \Rightarrow -rule.

□

15.B.3 Correspondence

Lemma 5 Let (p, n) be an interaction obligation for the sequence diagram d , I be a system and h a well-formed trace.

A : a. $(p, n) \rightsquigarrow_r (\text{traces}(I), \mathcal{H}^{ll(d)} \setminus \text{traces}(I))$
 b. $h \in \text{traces}(I)$

P : $(p, n) \mapsto_r (\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\})$,
 i.e. $(p, n) \rightsquigarrow_r (\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\})$ by definition 15.14.

P .

$\langle 1 \rangle 1.$ Requirement 1: $n \subseteq \mathcal{H}^{ll(d)} \setminus \{h\}$

$\langle 2 \rangle 1.$ $n \subseteq \mathcal{H}^{ll(d)} \setminus \text{traces}(I)$

P : Assumption 1 and definition 15.11 of \rightsquigarrow_r .

$\langle 2 \rangle 2.$ $h \in \text{traces}(I)$

P : Assumption 2.

$\langle 2 \rangle 3.$ Q.E.D.

P : $\langle 2 \rangle 1, \langle 2 \rangle 2$ and $A \subseteq B \setminus X \wedge x \in X \Rightarrow A \subseteq B \setminus \{x\}$ for arbitrary sets A, B and X .

$\langle 1 \rangle 2.$ Requirement 2: $p \subseteq \{h\} \cup (\mathcal{H}^{ll(d)} \setminus \{h\})$, i.e. $p \subseteq \{h\} \cup \mathcal{H}^{ll(d)}$

P : $p \subseteq \mathcal{H}^{ll(d)}$ by definition of $\mathcal{H}^{ll(d)}$, as (p, n) is an interaction obligation for d .

$\langle 1 \rangle 3.$ Q.E.D.

P : $\langle 1 \rangle 1, \langle 1 \rangle 2$ and definition 15.11 of \rightsquigarrow_r .

□

Lemma 6 Let (p, n) be an interaction obligation for the sequence diagram d , I be a system and h a well-formed trace.

A : $\forall h \in \text{traces}(I) : (p, n) \rightsquigarrow_r (\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\})$

P : $(p, n) \mapsto_r (\text{traces}(I), \mathcal{H}^{ll(d)} \setminus \text{traces}(I))$,
 i.e. $(p, n) \rightsquigarrow_r (\text{traces}(I), \mathcal{H}^{ll(d)} \setminus \text{traces}(I))$ by definition 15.14.

P .

$\langle 1 \rangle 1.$ Requirement 1: $n \subseteq \mathcal{H}^{ll(d)} \setminus traces(I)$

$\langle 2 \rangle 1.$ $\forall h \in traces(I) : n \subseteq \mathcal{H}^{ll(d)} \setminus \{h\}$

P : The assumption and definition 15.11 of \rightsquigarrow_r .

$\langle 2 \rangle 2.$ $\forall h \in traces(I) : \{h\} \cap n = \emptyset$

P : $\langle 2 \rangle 1$ and definition of \subseteq .

$\langle 2 \rangle 3.$ $traces(I) \cap n = \emptyset$

P : $\langle 2 \rangle 2$ and $X \cap A = \emptyset \wedge Y \cap A = \emptyset \Rightarrow (X \cup Y) \cap A = \emptyset$ for arbitrary sets A, X and Y .

$\langle 2 \rangle 4.$ $n \subseteq \mathcal{H}^{ll(d)}$

P : Definition of $\mathcal{H}^{ll(d)}$, as (p, n) is an interaction obligation for d .

$\langle 2 \rangle 5.$ Q.E.D.

P : $\langle 2 \rangle 3, \langle 2 \rangle 4$ and $A \subseteq B \wedge A \cap X = \emptyset \Rightarrow A \subseteq B \setminus X$ for arbitrary sets A, B and X .

$\langle 1 \rangle 2.$ Requirement 2: $p \subseteq traces(I) \cup (\mathcal{H}^{ll(d)} \setminus traces(I))$, i.e. $p \subseteq traces(I) \cup \mathcal{H}^{ll(d)}$

P : $p \subseteq \mathcal{H}^{ll(d)}$ by definition of $\mathcal{H}^{ll(d)}$, as (p, n) is an interaction obligation for d .

$\langle 1 \rangle 3.$ Q.E.D.

P : $\langle 1 \rangle 1, \langle 1 \rangle 2$ and definition 15.11 of \rightsquigarrow_r .

□

Theorem 27 (Correspondence.) Let d be a sequence diagram with no `xalt` operator. Then

$$\llbracket d \rrbracket^u \mapsto_r \langle I \rangle_d^u \Rightarrow \llbracket d \rrbracket^i \mapsto_g \langle I \rangle_d^i$$

P .

L : $\llbracket d \rrbracket^u = (p, n)$,

i.e. $\llbracket d \rrbracket^i = \{(p, n)\}$

$\langle 1 \rangle 1.$ A : $\llbracket d \rrbracket^u \mapsto_r \langle I \rangle_d^u$

P : $\llbracket d \rrbracket^i \mapsto_g \langle I \rangle_d^i$

$\langle 2 \rangle 1.$ $(p, n) \mapsto_r (traces(I), \mathcal{H}^{ll(d)} \setminus traces(I))$

P : $\langle 1 \rangle 1, \llbracket d \rrbracket^u = (p, n)$ and definition 15.13 of $\langle I \rangle_d^u$.

$\langle 2 \rangle 2.$ $(p, n) \rightsquigarrow_r (traces(I), \mathcal{H}^{ll(d)} \setminus traces(I))$

P : $\langle 2 \rangle 1$ and definition 15.14 of \mapsto_r

$\langle 2 \rangle 3.$ Choose arbitrary $h \in traces(I)$

P : $traces(I)$ is non-empty for all real systems I .

$\langle 2 \rangle 4.$ $(p, n) \mapsto_r (\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\})$

P : $\langle 2 \rangle 2, \langle 2 \rangle 3$ and lemma 5.

$\langle 2 \rangle 5.$ $(\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\}) \in \langle I \rangle_d^i$

P : $\langle 2 \rangle 3$ and definition 15.23 of $\langle I \rangle_d^i$.

$\langle 2 \rangle 6.$ $\forall o \in \llbracket d \rrbracket^i : \exists o' \in \langle I \rangle_d^i : o \mapsto_r o'$

P : $\llbracket d \rrbracket^i = \{(p, n)\}$, $\langle 2 \rangle 4, \langle 2 \rangle 5$ and \forall -rule.

$\langle 2 \rangle 7.$ Q.E.D.

P : $\langle 2 \rangle 6$ and definition 15.24 of \mapsto_g .

$\langle 1 \rangle 2.$ Q.E.D.

P : \Rightarrow -rule.

□

Theorem 28 (Correspondence.) Let d be a sequence diagram with no `xalt` operator. Then

$$\llbracket d \rrbracket^u \mapsto_{rr} \langle I \rangle_d^u \Rightarrow \llbracket d \rrbracket^i \mapsto_{rg} \langle I \rangle_d^i$$

P .

L : $\llbracket d \rrbracket^u = (p, n),$
i.e. $\llbracket d \rrbracket^i = \{(p, n)\}$

$\langle 1 \rangle 1.$ A : $\llbracket d \rrbracket^u \mapsto_{rr} \langle I \rangle_d^u$

P : $\llbracket d \rrbracket^i \mapsto_{rg} \langle I \rangle_d^i$

$\langle 2 \rangle 1.$ $(p, n) \mapsto_{rr} (\text{traces}(I), \mathcal{H}^{ll(d)} \setminus \text{traces}(I))$

P : $\langle 1 \rangle 1, \llbracket d \rrbracket^u = (p, n)$ and definition 15.13 of $\langle I \rangle_d^u$.

$\langle 2 \rangle 2.$ $(p, n) \rightsquigarrow_r (\text{traces}(I), \mathcal{H}^{ll(d)} \setminus \text{traces}(I))$

P : $\langle 2 \rangle 1$ and definitions 15.14 and 15.15 of \mapsto_{rr} .

$\langle 2 \rangle 3.$ $p \cap \text{traces}(I) \neq \emptyset$

P : $\langle 2 \rangle 1$ and definition 15.15 of \mapsto_{rr} .

$\langle 2 \rangle 4.$ Choose $h \in \text{traces}(I)$ such that $p \cap \{h\} \neq \emptyset$

P : $\langle 2 \rangle 3.$

$\langle 2 \rangle 5.$ $(p, n) \mapsto_r (\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\})$

P : $\langle 2 \rangle 2, \langle 2 \rangle 4$ and lemma 5.

$\langle 2 \rangle 6.$ $(\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\}) \in \langle I \rangle_d^i$

P : $\langle 2 \rangle 4$ and definition 15.23 of $\langle I \rangle_d^i$.

$\langle 2 \rangle 7.$ $\forall o \in \llbracket d \rrbracket^i : \exists o' \in \langle I \rangle_d^i : o \mapsto_{rr} o'$

P : $\llbracket d \rrbracket^i = \{(p, n)\}, \langle 2 \rangle 4, \langle 2 \rangle 5, \langle 2 \rangle 6$ and \forall -rule.

$\langle 2 \rangle 8.$ Q.E.D.

P : $\langle 2 \rangle 7$ and definition 15.24 of \mapsto_{rg} .

$\langle 1 \rangle 2.$ Q.E.D.

P : \Rightarrow -rule.

□

Theorem 29 (Correspondence.) Let d be a sequence diagram with no `xalt` operator. Then

$$\llbracket d \rrbracket^u \mapsto_r \langle I \rangle_d^u \Rightarrow \llbracket d \rrbracket^i \mapsto_l \langle I \rangle_d^i$$

P .

L : $\llbracket d \rrbracket^u = (p, n),$
i.e. $\llbracket d \rrbracket^i = \{(p, n)\}$

$\langle 1 \rangle 1.$ A : $\llbracket d \rrbracket^u \mapsto_r \langle I \rangle_d^u$

P : $\llbracket d \rrbracket^i \mapsto_l \langle I \rangle_d^i$

- $\langle 2 \rangle 1. \llbracket d \rrbracket^i \mapsto_g \langle I \rangle_d^i$
P : $\langle 1 \rangle 1$ and theorem 27 (correspondence between \mapsto_r and \mapsto_g).
 $\langle 2 \rangle 2. (p, n) \mapsto_r (\text{traces}(I), \mathcal{H}^{ll(d)} \setminus \text{traces}(I))$
P : $\langle 1 \rangle 1, \llbracket d \rrbracket^u = (p, n)$ and definition 15.13 of $\langle I \rangle_d^u$.
 $\langle 2 \rangle 3. (p, n) \rightsquigarrow_r (\text{traces}(I), \mathcal{H}^{ll(d)} \setminus \text{traces}(I))$
P : $\langle 2 \rangle 2$ and definition 15.14 of \mapsto_r
 $\langle 2 \rangle 4. \text{Choose arbitrary } o' \in \langle I \rangle_d^i$
P : $\langle I \rangle_d^i$ is non-empty for all real systems I .
 $\langle 2 \rangle 5. \text{Choose } h \in \text{traces}(I) \text{ such that } o' = (\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\})$
P : $\langle 2 \rangle 4$ and definition 15.23 of $\langle I \rangle_d^i$.
 $\langle 2 \rangle 6. (p, n) \mapsto_r (\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\})$
P : $\langle 2 \rangle 3, \langle 2 \rangle 5$ and lemma 5.
 $\langle 2 \rangle 7. \forall o' \in \langle I \rangle_d^i : \exists o \in \llbracket d \rrbracket^i : o \mapsto_r o'$
P : $\llbracket d \rrbracket^i = \{(p, n)\}$, $\langle 2 \rangle 4, \langle 2 \rangle 5, \langle 2 \rangle 6$ and \forall -rule.
 $\langle 2 \rangle 8. \text{Q.E.D.}$
P : $\langle 2 \rangle 1, \langle 2 \rangle 7$ and definition 15.25 of \mapsto_l .
 $\langle 1 \rangle 2. \text{Q.E.D.}$
P : \Rightarrow -rule.

□

Theorem 30 (Correspondence.) Let d be a sequence diagram with no `xalt` operator. Then

$$\llbracket d \rrbracket^u \mapsto_r \langle I \rangle_d^u \Leftarrow \llbracket d \rrbracket^i \mapsto_l \langle I \rangle_d^i$$

- P .
L : $\llbracket d \rrbracket^i = \{(p, n)\}$,
i.e. $\llbracket d \rrbracket^u = (p, n)$
 $\langle 1 \rangle 1. A : \llbracket d \rrbracket^i \mapsto_l \langle I \rangle_d^i$
P : $\llbracket d \rrbracket^u \mapsto_r \langle I \rangle_d^u$
 $\langle 2 \rangle 1. \{(p, n)\} \mapsto_l \{(\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\}) \mid h \in \text{traces}(I)\}$
P : $\langle 1 \rangle 1, \llbracket d \rrbracket^i = \{(p, n)\}$ and definition 15.23 of $\langle I \rangle_d^i$.
 $\langle 2 \rangle 2. \forall h \in \text{traces}(I) : (p, n) \mapsto_r (\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\})$
P : $\langle 2 \rangle 1$ and definition 15.25 of \mapsto_l .
 $\langle 2 \rangle 3. \forall h \in \text{traces}(I) : (p, n) \rightsquigarrow_r (\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\})$
P : $\langle 2 \rangle 2$ and definition 15.14 of \mapsto_r .
 $\langle 2 \rangle 4. (p, n) \mapsto_r (\text{traces}(I), \mathcal{H}^{ll(d)} \setminus \text{traces}(I))$
P : $\langle 2 \rangle 3$ and lemma 6.
 $\langle 2 \rangle 5. \text{Q.E.D.}$
P : $\langle 2 \rangle 4, \llbracket d \rrbracket^u = (p, n)$ and definition 15.13 of $\langle I \rangle_d^u$.
 $\langle 1 \rangle 2. \text{Q.E.D.}$
P : \Leftarrow -rule.

□

Theorem 31 (Correspondence.) Let d be a sequence diagram with no `xalt` operator. Then

$$\llbracket d \rrbracket^u \mapsto_{rr} \langle I \rangle_d^u \Leftarrow \llbracket d \rrbracket^i \mapsto_{rl} \langle I \rangle_d^i$$

P .

L : $\llbracket d \rrbracket^i = \{(p, n)\}$,

i.e. $\llbracket d \rrbracket^u = (p, n)$

$\langle 1 \rangle 1.$ A : $\llbracket d \rrbracket^i \mapsto_{rl} \langle I \rangle_d^i$

P : $\llbracket d \rrbracket^u \mapsto_{rr} \langle I \rangle_d^u$

$\langle 2 \rangle 1.$ $\{(p, n)\} \mapsto_{rl} \{(\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\}) \mid h \in \text{traces}(I)\}$

P : $\langle 1 \rangle 1, \llbracket d \rrbracket^i = \{(p, n)\}$ and definition 15.23 of $\langle I \rangle_d^i$.

$\langle 2 \rangle 2.$ $\forall h \in \text{traces}(I) : (p, n) \mapsto_{rr} (\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\})$

P : $\langle 2 \rangle 1$ and definition 15.25 of \mapsto_{rl} .

$\langle 2 \rangle 3.$ $\forall h \in \text{traces}(I) : (p, n) \rightsquigarrow_r (\{h\}, \mathcal{H}^{ll(d)} \setminus \{h\})$

P : $\langle 2 \rangle 2$ and definitions 15.14 and 15.15 of \mapsto_{rr} .

$\langle 2 \rangle 4.$ $(p, n) \mapsto_r (\text{traces}(I), \mathcal{H}^{ll(d)} \setminus \text{traces}(I))$

P : $\langle 2 \rangle 3$ and lemma 6.

$\langle 2 \rangle 5.$ $\forall h \in \text{traces}(I) : p \cap \{h\} \neq \emptyset$

P : $\langle 2 \rangle 2$ and definition 15.15 of \mapsto_{rr} .

$\langle 2 \rangle 6.$ $p \cap \text{traces}(I) \neq \emptyset$

P : $\langle 2 \rangle 5$

$\langle 2 \rangle 7.$ $(p, n) \mapsto_{rr} (\text{traces}(I), \mathcal{H}^{ll(d)} \setminus \text{traces}(I))$

P : $\langle 2 \rangle 4, \langle 2 \rangle 6$ and definition 15.15 of \mapsto_{rr} .

$\langle 2 \rangle 8.$ Q.E.D.

P : $\langle 2 \rangle 7, \llbracket d \rrbracket^u = (p, n)$ and definition 15.13 of $\langle I \rangle_d^u$.

$\langle 1 \rangle 2.$ Q.E.D.

P : \Leftarrow -rule.

□

Chapter 16

STAIRS Case Study: The BuddySync System

Ragnhild Kobro Runde

Publication.

Published as Technical Report 345, Department of Informatics, University of Oslo, 2007.

Abstract.

This paper presents a case study evaluating the use of STAIRS when specifying a system for connecting service providers and people requesting those services. As part of the case study, we give an example of how STAIRS may be used in combination with development methodologies like e.g. RUP. We conclude that STAIRS seems a promising method for working with UML 2.x interactions, and indicate some possible directions for future research.

16.1 Introduction

STAIRS is a method for the compositional development of interactions in the setting of UML 2.x [OMG06]. The main motivation for STAIRS is

- understanding the meaning of the different advanced interaction operators in UML 2.x, and how these should be used in specifications using interactions, and
- explaining how initial specifications in the form of interactions may be refined into more complete descriptions of the system under development.

In order to achieve this, STAIRS provides a denotational trace semantics for the main parts of UML 2.x interactions, as well as a set of refinement notions.

Previous papers on STAIRS ([HHR05a], [HHR05b], [RHS05b], [RHS05a] and [RRS06]) have mainly been concerned with explaining STAIRS through formal definitions. Although the papers contain various examples, these have all been constructed for specific illustrative purposes. In [RHS06] we gave a tutorial introduction to STAIRS, explaining the practical relevance of STAIRS through pragmatical guidelines.

This paper presents an evaluation of STAIRS based on a thorough case study performed by ourselves. From an informal description of a system, a specification in the form of UML 2.x interactions is developed using our general knowledge of STAIRS and in particular the guidelines from [RHS06]. This paper presents mainly the resulting specifications, and should not be understood as providing a complete documentation of the development process.

In this paper we have chosen to present only the case study, and not STAIRS itself. For material on STAIRS we refer to the above mentioned papers, and in particular [RHS06]. The remainder of this paper is structured as follows: In Section 16.2 we present our evaluation criteria for STAIRS. In Section 16.3 we present the BuddySync system, the subject of this case study. Section 16.4 presents the development methodology used, and Sections 16.5–16.7 present the resulting UML 2.x interactions specifying the system. In Section 16.8 we evaluate STAIRS with respect to the evaluation criteria, before giving some concluding remarks in Section 16.9.

16.2 Evaluation Criteria

In this section we list our evaluation criteria for STAIRS. The criteria are inspired by [KS03], a framework for understanding quality in conceptual modelling. For evaluating language quality, a distinction is made between criteria for the concepts in the underlying basis, and criteria for the syntactic constructs used to represent these concepts visually. Many of the criteria in [KS03] are criteria that apply to UML in general, such as criteria regarding the size, solidity and position of the various elements in UML 2.x interactions. These criteria are not relevant here, where we evaluate only what is STAIRS specific. For an evaluation of UML in general, see e.g. [Kro05].

The criteria we have chosen for evaluating STAIRS are divided into three categories as follows:

- Criteria for the conceptual basis of the language:
 - All relevant knowledge should be expressible.

- b. The concepts should be general, limiting the total number of concepts.
- c. The concepts should be composable, so that related requirements may be grouped together.
- d. Both precise and vague knowledge should be expressible.
- e. The concepts should be easily distinguished from each other.
- f. A concept should mean the same thing every time it is used.
- g. The concepts should allow flexibility in the level of detail.
- h. It should be possible to divide the models into natural parts, enabling work division.
- i. The most frequent kinds of requirements should be expressible in a compact form.
- Criteria for the visual representation of the language:
 - j. The mapping from syntactic constructs to the underlying concepts must be unambiguous.
 - k. The constructs should be easily distinguished from each other.
 - l. A construct should represent the same concept in all contexts.
 - m. The constructs should be composable, in order to support the grouping of related requirements.
 - n. Constructs without information should be avoided.
- Criteria with respect to refinement:
 - o. The refinement relations should be powerful enough to capture all refinement steps made in practice.
 - p. The refinement relations should be general, limiting the total number of relations.
 - q. The refinement relations should be easily distinguished from each other.
 - r. It should be possible to refine the different parts of a specification separately.

As pointed out in [KS03], deficiencies in the language may be addressed by the methodology used. In our case, this means that we will also evaluate how the guidelines given in [RHS06] help with e.g. distinguishing the different concepts, constructs and refinement relations. For easy reference, these guidelines are included in Appendix 16.A.

16.3 Initial Description of the BuddySync System

In this section we describe the BuddySync system, based on the text for a mandatory assignment in the course INF-UIT at the University of Oslo autumn 2001. This has been chosen as the system for this case study for the following reasons:

- It is a different kind of system than those used as examples in our previous papers.
- Communication with different kinds of users is an essential part of the system.

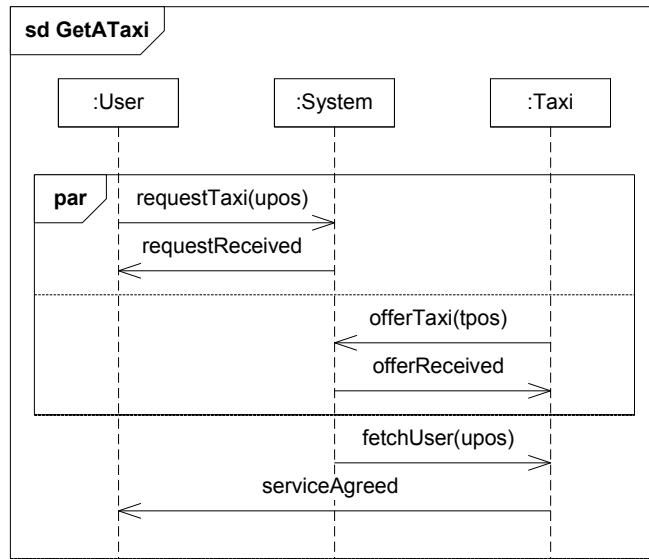


Figure 16.1: Example scenario for taxi service

- The assignment was created for using sequence diagrams (i.e. interactions), but not specifically tailored towards STAIRS (which did not exist at the time).
- Being a course assignment, the size of the system is manageable.

The BuddySync system is a system that helps connecting people that need a service with people that offer it. The idea is that users should be able to send an SMS to the system, either offering a service or requesting one. The system should then automatically match service requests and service offers. For some services, the system may also forward requests to potential service providers. Before requesting or offering a service, the user must be registered. Also, it should be possible to remove service requests and offers.

An example of a BuddySync service is taking a taxi, described as follows:

- A taxi offers its service as soon as it becomes available, stating its geographical position.
- A passenger requests a taxi. Either he is automatically assigned to one of the available taxis in his area, or he must wait for a taxi to become available.

Some initial example scenarios for this service is described by the sequence diagrams in Figures 16.1 and 16.2. In GetATaxi in Figure 16.1 the user requests a taxi (to position upos), and the taxi offers its service (at position tpos). The use of `par` indicates that either the user or taxi may initiate the interaction, or they may do so in parallel. After matching the request and the offer (implicitly comparing upos and tpos), the system tells the taxi to pick up the user at upos, and the taxi informs the user of its upcoming arrival.

In RemoveTaxiRequest in Figure 16.2, the user first requests a taxi, but then he withdraws the request. This may happen for instance if it takes too long before a taxi becomes available, and the user decides to go by the bus instead.

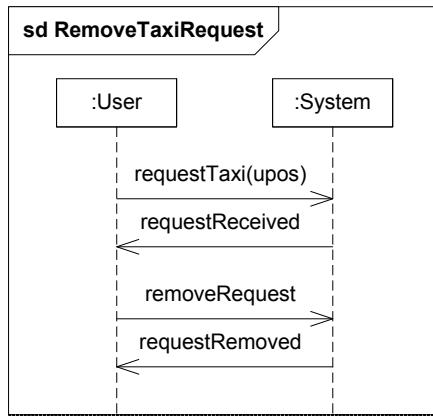


Figure 16.2: Example scenario for removing a taxi request

16.4 Development Methodology

STAIRS is not intended to be a complete methodology for system development, but should rather be seen as a supplement to existing methodologies such as e.g. RUP [Kru04]. The purpose of this case study is to create a specification in the form of UML 2.x interactions, in order to perform an evaluation of STAIRS. As this is not a real project, there are no customers and no budget involved. Also, there will be no actual implementation of the system. Hence, many parts of RUP are not relevant for this case study although they would have been in a real project for the same system.

Of the four main phases of RUP, the activities performed in this case study fit in as parts of the elaboration phase, which is the phase where the majority of the functional and non-functional requirements are specified. Each of the main phases of RUP consists of a sequence of iterations. RUP also defines nine disciplines that cut across the iterations. The disciplines relevant in the setting of this case study is the requirements discipline, where the goal is establishing an agreement with the customers as to what the system should do, and the analysis and design discipline, where the requirements are translated into a specification of how the system should be implemented.

From the informal requirements in Section 16.3, we plan three iterations each adding a few more features to the system (specification):

- **Iteration 1:** Requesting and offering services. This is the main functionality of the system, and a natural starting point. For this first iteration, we assume that all users are registered, and leave the actual checking of this to iteration 3.
- **Iteration 2:** Removing service requests and offers.
- **Iteration 3:** Subscribing and unsubscribing. First of all, this iteration should add (un-)subscription mechanisms to the system. Also, subscription checking should be added to the handling of service requests/offers from iteration 1.

Each of the above iterations should consist of the following activities:

- Specify the required functionality from the perspective of a system user.
- Specify how the system should implement the required functionality.
- Check the resulting interactions with respect to the guidelines given in [RHS06] for creating interactions (included here in Appendix 16.A.1).
- Whenever there is more than one interaction specifying the same functionality, check that they are in a refinement relationship according to the guidelines in [RHS06] (Appendix 16.A.2).

In a specification process such as the one described here, there will usually be some degree of trial and error, where e.g. alternative designs are explored and an increased understanding of the domain and the system to built results in modifications of diagrams made earlier in the process. These parts of the case study are not reported here, as we are mainly interested in the diagrams that are correct descriptions of the BuddySync system and the relationships between these diagrams.

16.5 Iteration 1: RequestService and OfferService

A general overview of the behaviours of the BuddySync system is given in Figure 16.3. All of the referenced interactions describe functionality that is required of the system. The decision with respect to which of the referenced interactions will be performed in each of the iterations depends on the given user input. Hence, the operator xalt is used according to the guidelines in Appendix 16.A.1. For simplicity, we assume that the system handles only one user input at the time. For handling several users at the same time, the system should be able to perform arbitrary many instances of Figure 16.3 in parallel.

As described in Section 16.4, we start by specifying the main functionality of the system, that is how it supports the requesting and offering of services. As the system should support a wide range of services, and the exact selection of services may change over time, it is important that the specifications should be generic with respect to service. For this first iteration, we assume that all users are registered.

16.5.1 User Requirements: RequestService

The composite diagram in Figure 16.4 gives the initial context for the BuddySync System. As illustrated, the system interacts with two different users, a requester and a provider, but these two never interact.

Figure 16.5 gives the initial specification of how a user interacts with the system when requesting a service. First, the user sends a request to the system, stating the requested service (e.g. taking a taxi) and detailed requirements (e.g. the time and place). These details are not important for our case study, so we have put them together in the parameter called *service*. As a reply to the request, the user should get either an agree-message indicating that a matching service provider has been found, or a message that the request is registered (waiting for a matching provider). Checking with the guidelines in Appendix 16.A.1, we specify these alternatives using

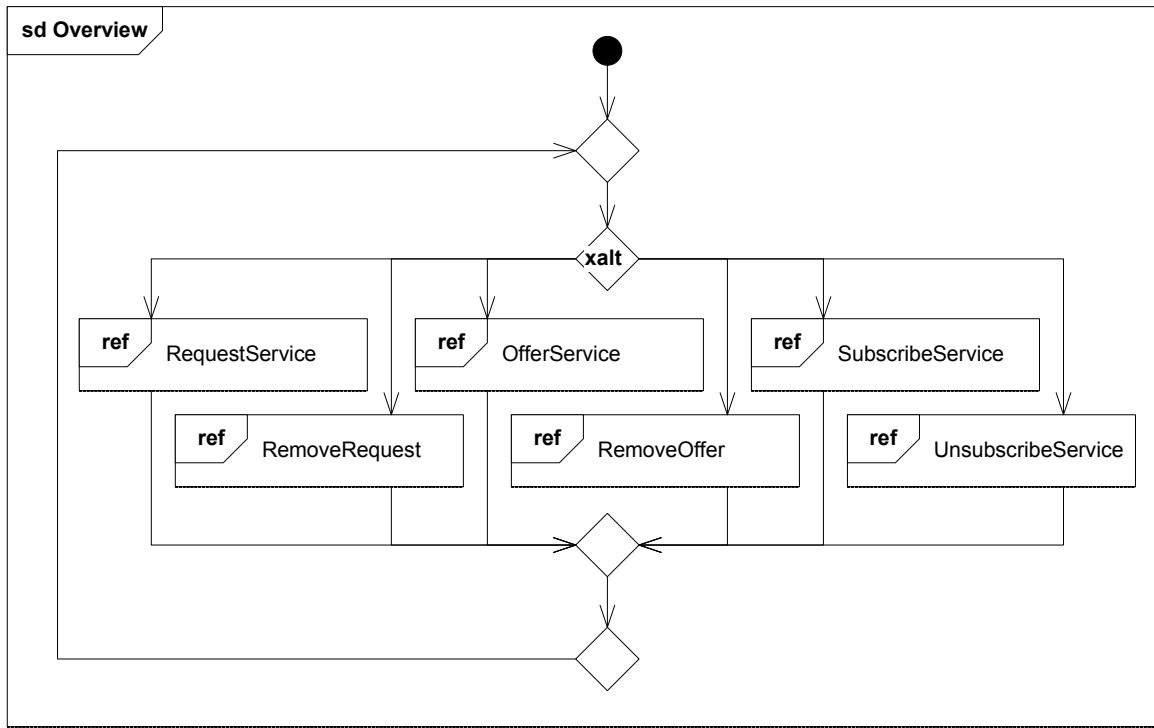


Figure 16.3: Overview of the BuddySync System

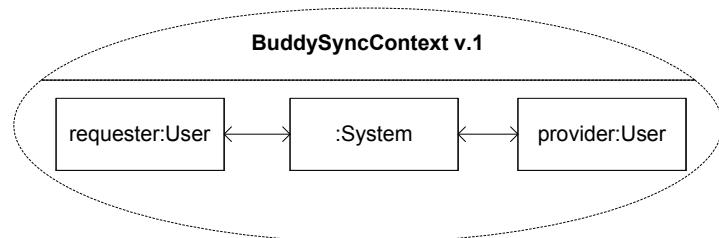


Figure 16.4: Composite structure for the BuddySync System

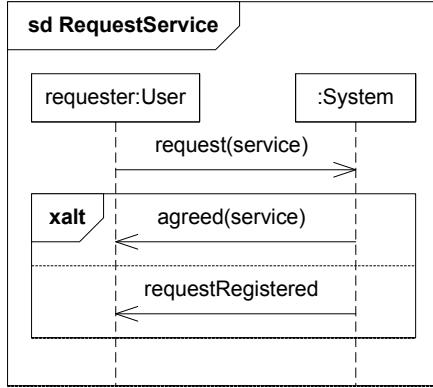


Figure 16.5: Request service — user view

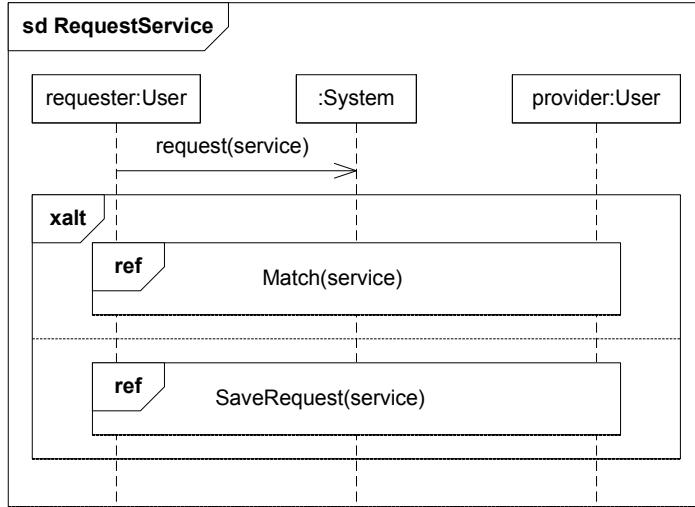


Figure 16.6: Request service — user view with provider

xalt as there is obviously some underlying condition in the system directing which one of these reply messages will be sent.

In Figure 16.6 we have added interaction with the provider (another user) to the specification in Figure 16.5. For readability and maintainability the two alternative system responses have been separated into the diagrams in Figures 16.7 and 16.8. If a match is found, the provider is notified via a perform-message, while in the case where the request is simply registered, the system may optionally also notify a potential provider of the requested service. (If the provider then chooses to offer the service, this should be treated as an ordinary service offer according to the specifications in Section 16.5.2.) The opt-construct is a shorthand for an alt between the given operand and the empty diagram (skip). Checking again with the guidelines in Appendix 16.A.1, using alt (and not xalt) is correct here, as this is an instance of underspecification where we do not require both alternatives to present in an implementation.

The requester and the provider may be seen as two different interfaces of the system, of which

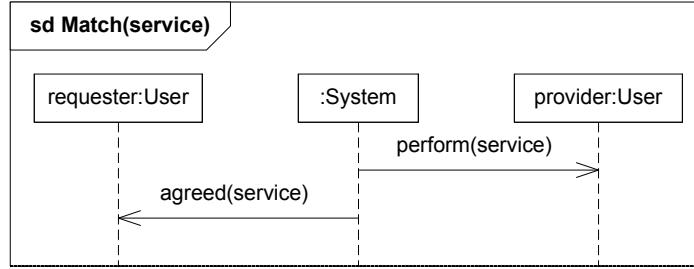


Figure 16.7: Matching service requester and provider

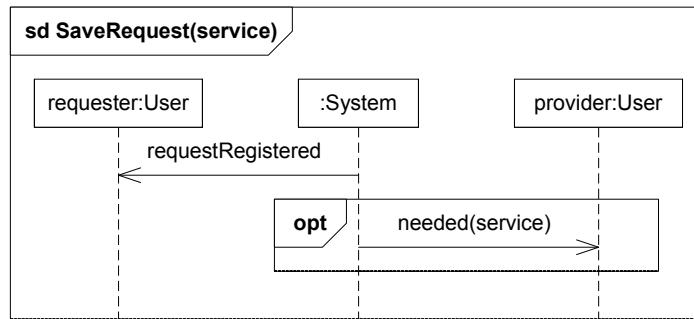


Figure 16.8: Saving request

Figure 16.5 only includes the requester while Figure 16.6 presents both interfaces. Comparing these two figures, it is straightforward to see that from the requester's perspective, the behaviour is the same. The notion of interfaces is not included in STAIRS. However, from the perspective of the requester, Figure 16.6 may also be understood as a detailing refinement of Figure 16.5 as it gives more details about how the system handles the request. In the terminology of Appendix 16.A.2, this would then correspond to a decomposition of the system, with the following lifeline mapping:

$$ID[\text{provider:User} \mapsto :System]$$

i.e. the identity mapping ID updated so that provider:User maps to :System.

16.5.2 User Requirements: OfferService

We now move on to specify how a user may offer his service via the BuddySync system. The initial specification of how the user interacts with the system is given in Figure 16.9. First, the user sends an offer to the system, giving the details of the provided service as parameter. As a reply, the user should get either a perform-message indicating that the system has found a matching service request, or a message that the offer has been registered in the system. Again, these alternatives are specified using `xalt` due to the underlying condition.

In Figure 16.10 interaction with the requester (another user) is added to the specification in Figure 16.9. If a match is found, the requester is notified via an agreed-message (in fact, here we reuse the specification of Match(service) in Figure 16.7), while in the case that the offer is simply

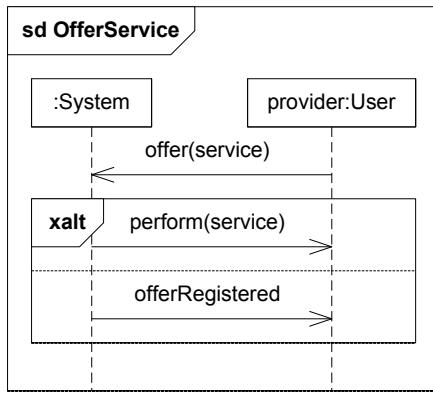


Figure 16.9: Offer service — user view

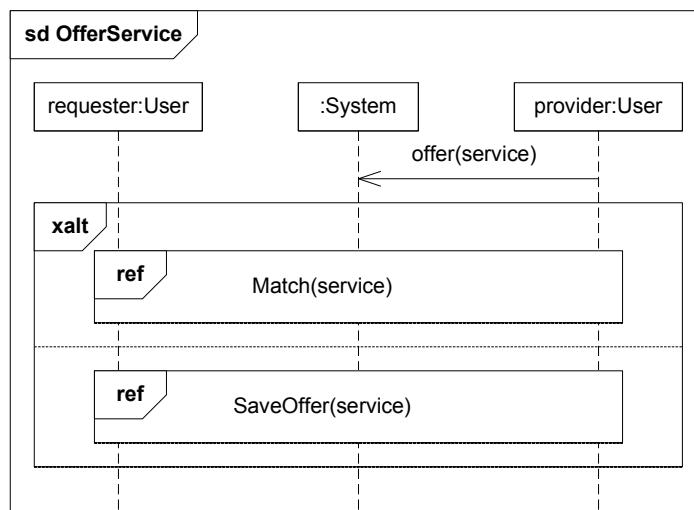


Figure 16.10: Offer service — user view with requester

saved, no interaction with the requester takes place as specified in Figure 16.11.

Comparing Figures 16.10 and 16.9, it is again straightforward to see that from the provider's perspective the behaviour is the same, meaning that this may be understood as a detailing refinement with lifeline mapping:

$$ID[\text{requester:User} \mapsto \text{:System}]$$

16.5.3 System Specification: RequestService

After having specified the functionality from the perspective of the user(s), we now turn to specifying how the system should implement this functionality. The revised composite structure diagram in Figure 16.12 illustrates how the system consist of two parts, a control and a messageboard. Only the control interacts with the users of the system. The messageboard should maintain a list of pending offers and a list of pending requests (not shown in the diagram).

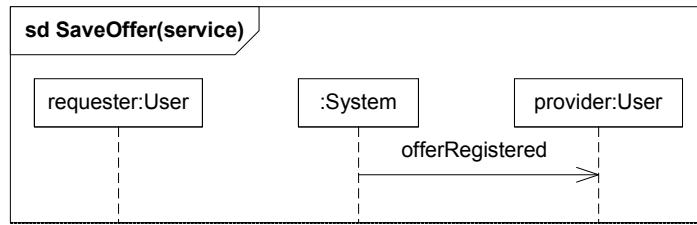


Figure 16.11: Saving offer

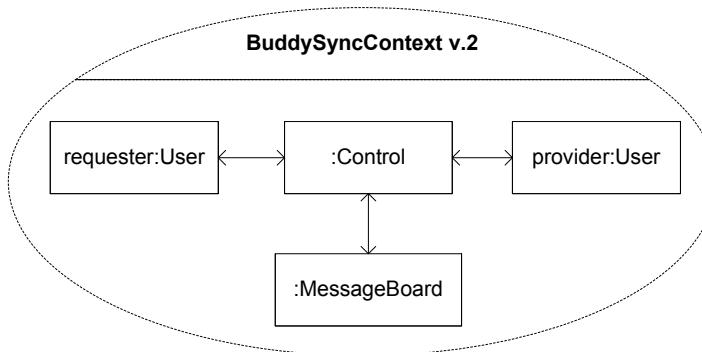


Figure 16.12: Revised composite structure diagram

In Figure 16.13, we take the specification of RequestService in Figure 16.6 and expand the system into control and messageboard.¹ Similarly, Figure 16.16 is an update of SaveRequest(service) in Figure 16.8. For the first `xalt-operand` in Figure 16.6, however, the reference to `Match(service)` is in Figure 16.13 replaced with a reference to `MatchRequest(service)` specified in Figure 16.14 (which again references the updated version of `Match(service)` in Figure 16.15).

Constraints are added to the beginning of `MatchRequest` (Figure 16.14) and `SaveRequest` (Figure 16.16), specifying the required conditions for performing each of them. The constraint `s ≈ service` takes into account that the match between a service request and a service offer does not need to be exact, as long as it is sufficiently similar. For instance, from its own position a taxi will usually have to drive for a few minutes before picking up its customer. In the context of Figure 16.13, the effect of the given constraints is exactly the same as if they instead had been added in the form of guards to the `xalt-construct`. In general, adding guards is a valid narrowing refinement according to Appendix 16.A.2.

However, adding guards is not the only change introduced by the specifications in this section. With respect to `Match(service)` in Figure 16.7, `MatchRequest(service)` in Figure 16.14 also adds an internal message between the control and the messageboard, and an assignment on messageboard. As the communication with the users is the same, these changes constitute a detailing

¹In UML 2.x, such a decomposition is usually given by using the `ref` construct in the header of the system lifeline, and then giving the decomposition in a separate diagram. As STAIRS do not cover extra global combined fragments, the same effect is obtained by expanding the system lifelines directly in the diagram.

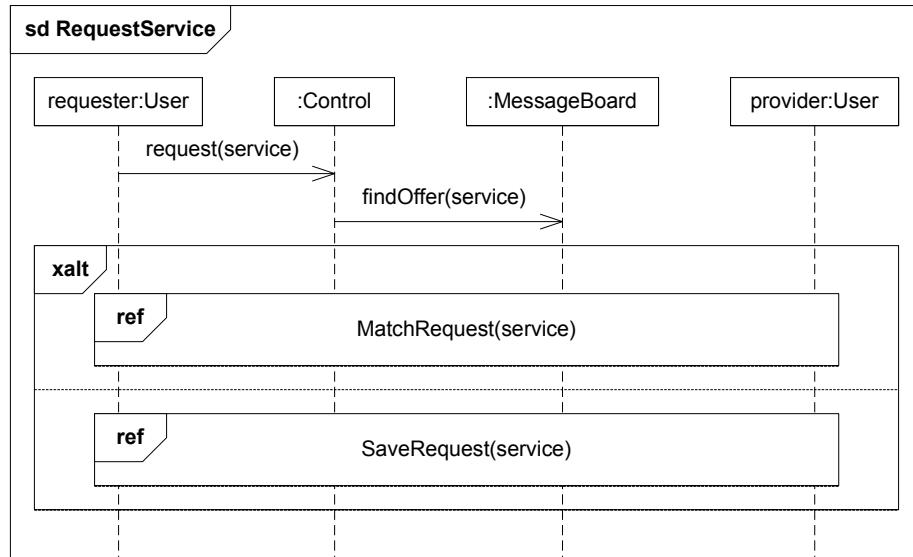


Figure 16.13: Request service — with control and messageboard

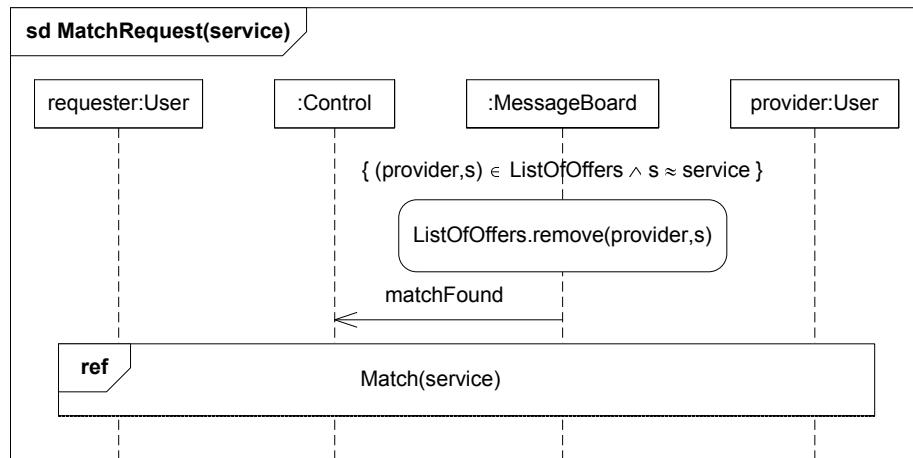


Figure 16.14: Matching service request with existing offer

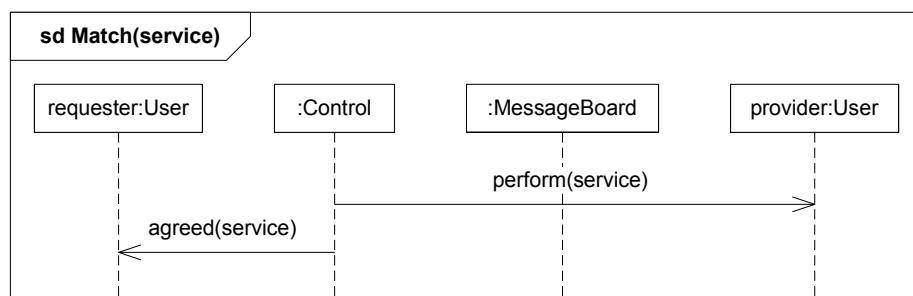


Figure 16.15: Matching service requester and provider — with control and messageboard

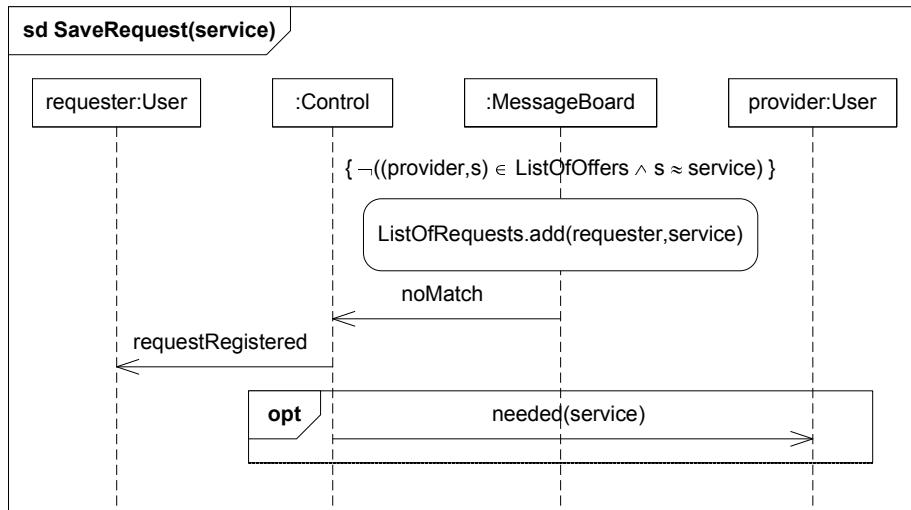


Figure 16.16: Saving request — with control and messageboard

refinement according to Appendix 16.A.2, with the lifeline mapping:

$ID[:Control \mapsto :System][:MessageBoard \mapsto :System]$

According to Appendix 16.A.2, narrowing and detailing may be performed in a single step, making Figure 16.14 a valid refinement of Figure 16.7.

Similarly, SaveRequest(service) in Figure 16.16 is a narrowing and detailing refinement of Figure 16.8 with the same lifeline mapping. For Figure 16.13, we now have that the operands of `xalt` are refined separately, giving a valid refinement of the total `xalt` construct. As the only other change from the specification in Figure 16.6 is the addition of a message between control and messageboard, we have again a case of narrowing and detailing refinement with the same lifeline mapping as above.

16.5.4 System Specification: OfferService

In Figure 16.17, the specification of OfferService in Figure 16.10 is changed to account for the new system structure in Figure 16.12. Similarly, Figure 16.19 is an update of SaveOffer(service) in Figure 16.11. For the first `xalt`-operand in Figure 16.17, the earlier reference to Match(service) is replaced with a reference to MatchOffer(service) specified in Figure 16.18.

We now see that MatchOffer(service) in Figure 16.18 is another narrowing and detailing refinement of Match(service) in Figure 16.7 with the same lifeline mapping as in Section 16.5.3. Similarly, Figure 16.19 is a general refinement of Figure 16.11, and Figure 16.17 is a general refinement of Figure 16.10.

16.5.5 Finishing the Iteration

Before ending the iteration, the resulting diagrams (Figures 16.13–16.19) should be checked against the guidelines for creating interactions given in Appendix 16.A.1.

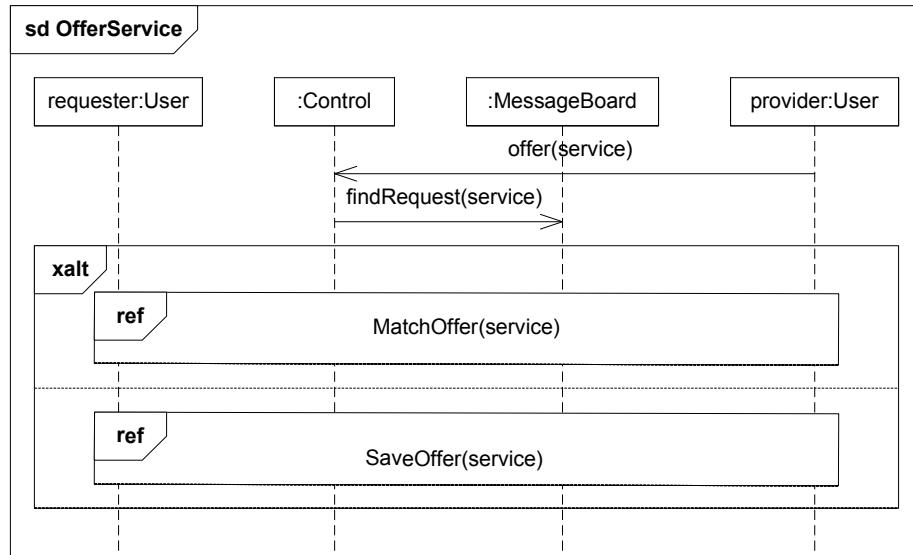


Figure 16.17: Offer service — with control and messageboard

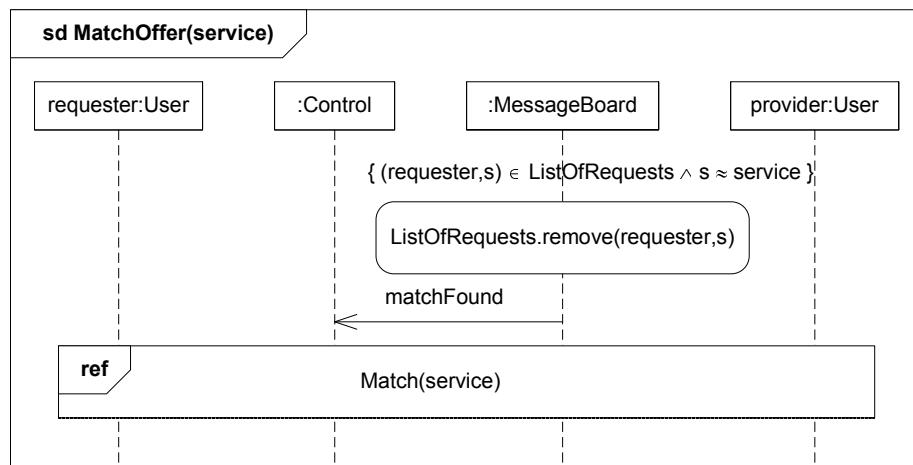


Figure 16.18: Matching service offer with existing request

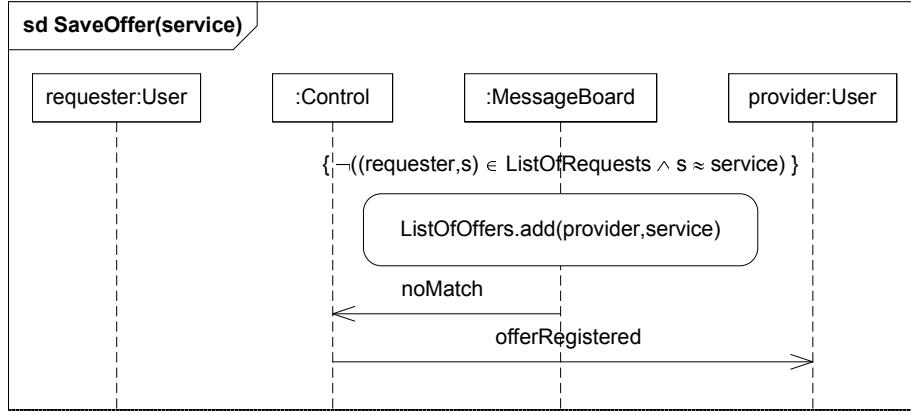


Figure 16.19: Saving offer — with control and messageboard

For the choice between `alt` and `xalt`, this has been checked each time one of these was selected for use in the specifications. Concerning the guards (specified as constraints in Figures 16.14, 16.16, 16.18 and 16.19), the guidelines given in Appendix 16.A.1 state that each guard should capture all possible situations for which the described traces are positive. Obviously, `MatchRequest` and `MatchOffer` may only be performed if there exists a corresponding offer, or request, at the messageboard. However, a reasonable question is whether it should be possible to do `SaveRequest` or `SaveOffer` also in the case of an existing match. After some considerations, we conclude that it would not be directly wrong, but less user-friendly and probably bad for business. Hence, the guards are left as they are.

Finally, the guidelines on negation given in Appendix 16.A.1 state that the specification should include a reasonable set of negative traces. In our specifications so far, the only negative traces we have are the traces where the constraints are false. This is clearly not sufficient. A simple way of adding negative traces to the specification would be to use `assert` as the top-level operator for `RequestService` and `OfferService`. However, before doing so we must be certain that all possible positive traces for these scenarios are described. We therefore consider each (sub-)diagram separately, trying to identify more positive and negative behaviours for that diagram.

For `Match(service)` in Figure 16.15, the order in which the system sends the messages to the requester and the provider is not important, so we add the `par`-operator as given in Figure 16.20. The same observation applies to `SaveRequest(service)` in Figure 16.16, where the new specification is given in Figure 16.21. According to the guidelines in Appendix 16.A.2, these changes are valid supplementing refinements as all of the original traces (positive and negative) are kept, while new positive traces are added to the specifications.

For `RequestService` in Figure 16.13 and `OfferService` in Figure 16.17, it is quite obvious that the two given alternatives are meant to be exclusive. For instance, the alternative with `MatchRequest` should not be implemented with traces from `SaveRequest` and vice versa. There are at least two ways of fixing this. One possibility is to use `assert` on each operand (or on the specification as a whole).

However, we already know that the specifications are not complete, as we intend to add subscription checking in a later iteration. Instead, we have chosen to supplement the traces of

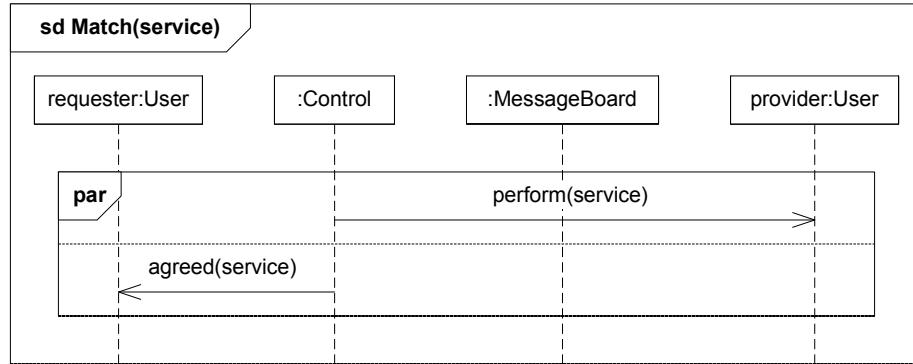


Figure 16.20: Match(service) after guideline-checking

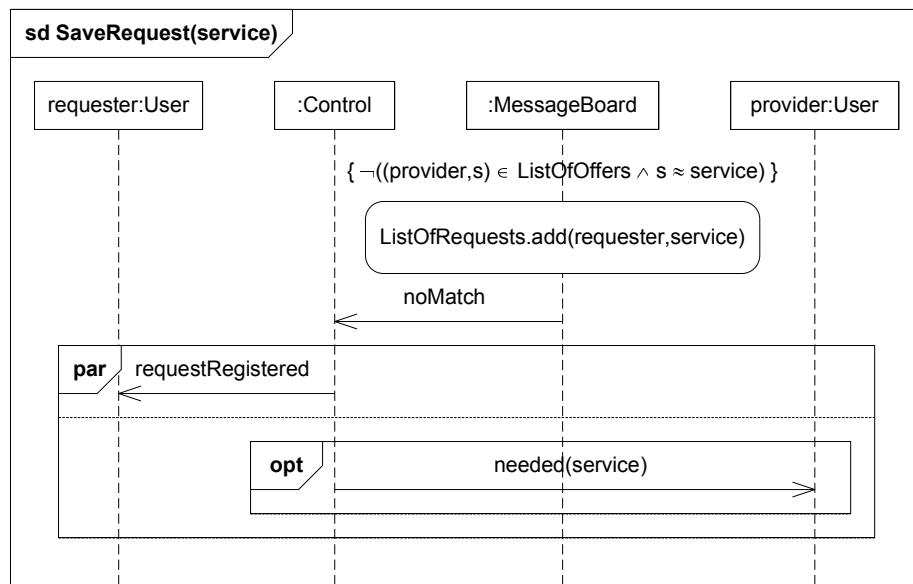


Figure 16.21: SaveRequest(service) after guideline-checking

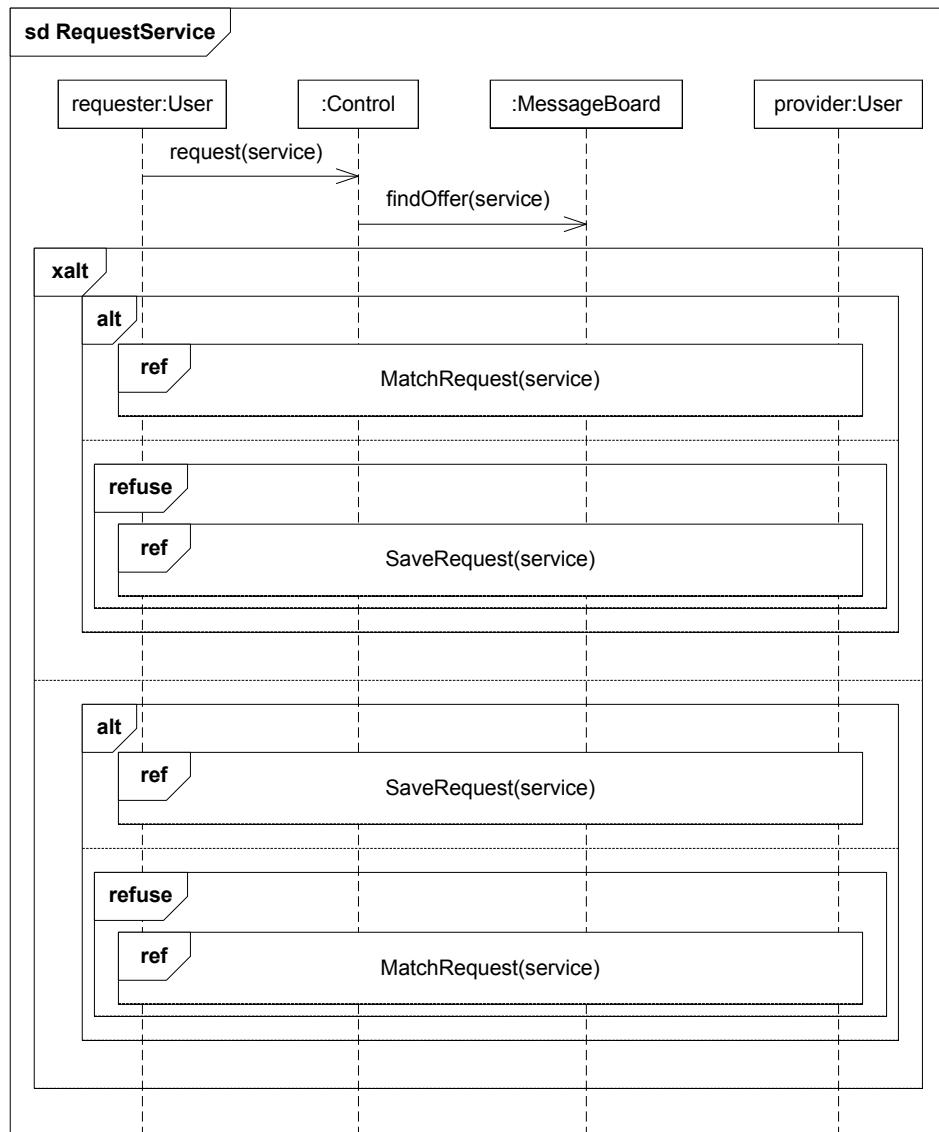


Figure 16.22: RequestService after guideline-checking

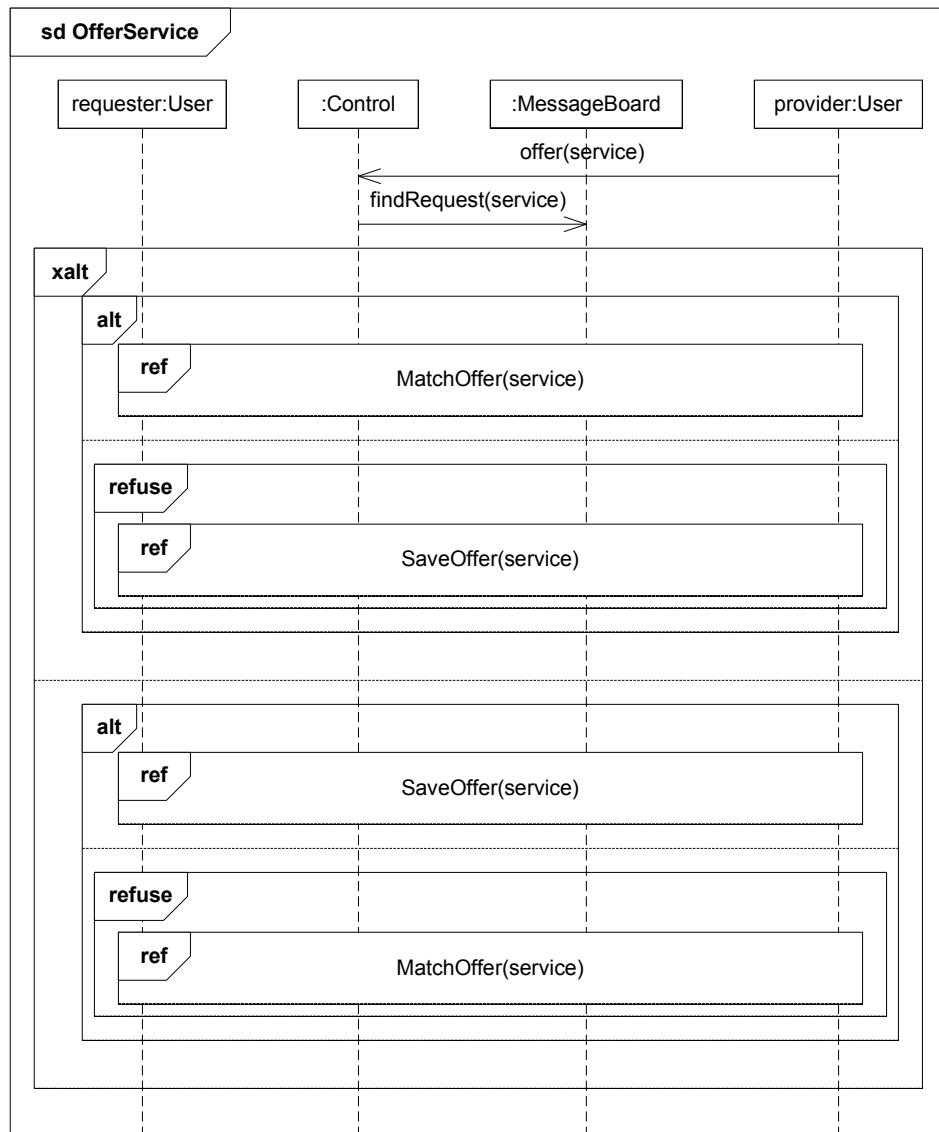


Figure 16.23: OfferService after guideline-checking

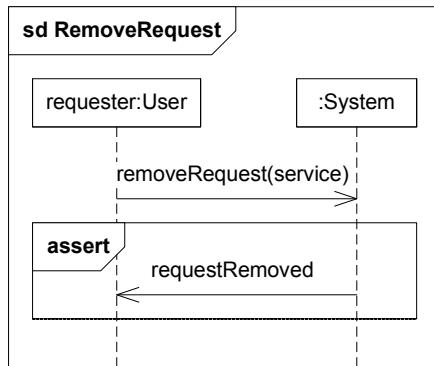


Figure 16.24: Remove request — user view

one `xalt` operand as negative for the other operand, as shown in Figures 16.22 and 16.23. In these specifications, `alt` is used in order to specify positive traces in one operand and negative traces in another operand. For the negation operator, `refuse` is used in accordance with the guidelines in Appendix 16.A.1.

To conclude this iteration, the specification now consists of the following diagrams:

RequestService	Figure 16.22
OferService	Figure 16.23
MatchRequest(service)	Figure 16.14
MatchOffer(service)	Figure 16.18
Match(service)	Figure 16.20
SaveRequest(service)	Figure 16.21
SaveOffer(service)	Figure 16.19

16.6 Iteration 2: RemoveRequest and RemoveOffer

As described in Section 16.4, this iteration should specify support for removing service requests and service offers.

16.6.1 User Requirements: RemoveRequest

Figure 16.24 specifies how a user interacts with the system when removing a service request. For the user, it is very important that the system really removes the request when asked to do so, hence the guidelines in Appendix 16.A.1 tells us to use `assert` on this message.

16.6.2 User Requirements: RemoveOffer

Figure 16.25 specifies how a user interacts with the system when removing a service request. The specification is symmetrical to RemoveRequest in Figure 16.24.

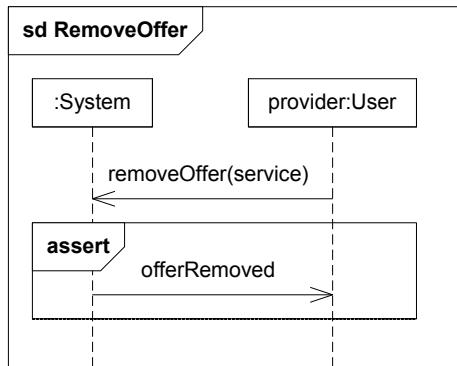


Figure 16.25: Remove offer — user view

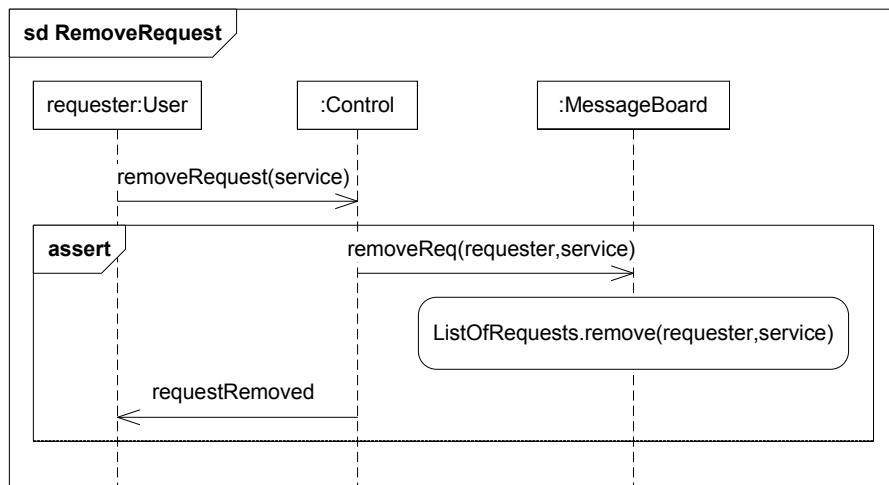


Figure 16.26: Remove request — with control and messageboard

16.6.3 System Specification: RemoveRequest

After having specified the functionality from the perspective of the user(s), we now turn to specifying how this should be implemented by a system consisting of a control and a messageboard as given in Figure 16.12.

Figure 16.26 gives the specification of how the system should handle request removals. A difficult choice here, is deciding how much of the diagram should be covered by the assert operator. It is not obvious that the one message from control to messageboard together with the assignment on messageboard is the only possible way for the system to handle the removal. For instance, an acknowledgment message back from messageboard to control is now forbidden. However, it is important for us to require that the remove-assignment is actually performed, and the simplest way of doing that is using assert as shown.

As before, comparing the specification in Figure 16.26 with the original specification in Figure 16.24, it should be fairly straightforward to see that the user interaction is the same. According to Appendix 16.A.2, Figure 16.26 is then a detailing refinement of Figure 16.24 with the

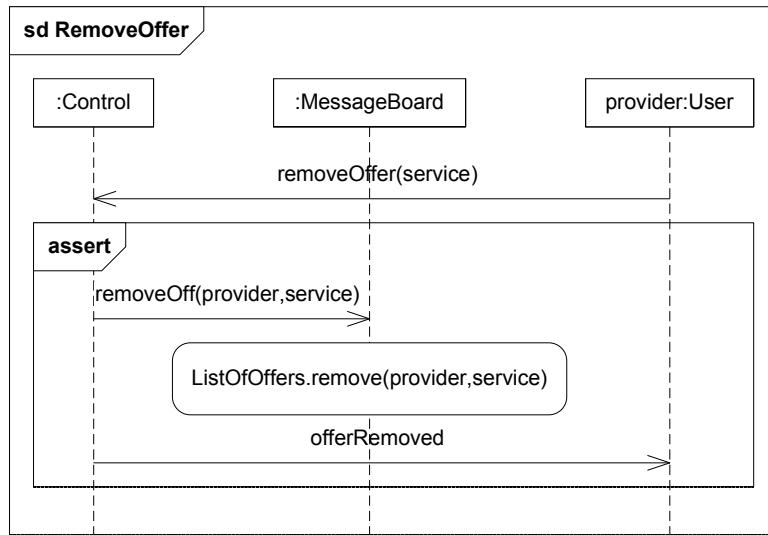


Figure 16.27: Remove offer — with control and messageboard

lifeline mapping:

$ID[:Control \mapsto :System][:MessageBoard \mapsto :System]$

16.6.4 System Specification: RemoveOffer

Figure 16.27 gives the specification of how the system should handle offer removals. Again, this is symmetrical to the specification of RemoveRequest in Figure 16.26. As the user interaction is the same, Appendix 16.A.2 gives that Figure 16.27 is a detailing refinement of Figure 16.25 with the lifeline mapping:

$ID[:Control \mapsto :System][:MessageBoard \mapsto :System]$

16.6.5 Finishing the Iteration

Before ending the iteration, the resulting diagrams (Figures 16.26 and 16.27) should be checked against the guidelines for creating interactions given in Appendix 16.A.1. These specifications contain no alternatives or guards, meaning that the guidelines in Appendices 16.A.1 and 16.A.1 are not relevant. According to the guidelines in Appendix 16.A.1, the specification should include a reasonable set of negative traces in order to effectively constrain an implementation of the system. In the interactions in Figures 16.26 and 16.27 this is ensured by the use of assert. As a conclusion, we leave these interactions as they are.

Also, no changes have been made to the system that affects the specifications from the previous iteration.

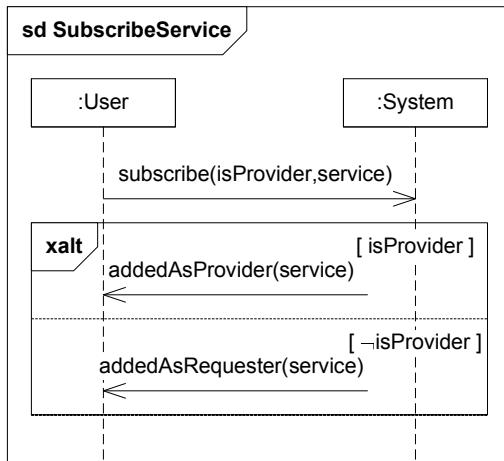


Figure 16.28: Subscribe service — user view

16.7 Iteration 3: SubscribeService and UnsubscribeService

As described in Section 16.4, in this final iteration we should add subscription and unsubscription mechanisms to the system. First, we decide that the subscription should apply to a single service only, and not to the complete system. This facilitates the same user being subscribed as a requester of one service, and as a provider of another service. Also, there are cases where the user may be subscribed as both requester and provider of the same service.

16.7.1 User Requirements: SubscribeService

Figure 16.28 specifies how subscription may look from the perspective of the user. Here, `isProvider` is a boolean flag indicating whether this is a subscription as a provider or as a requester (if the flag is false). If the subscription is as a provider, the user should get the message `addedAsProvider` as a receipt, if the subscription is as a requester the receipt message should be `addedAsRequester`. As these are alternatives with conditions, `xalt` are used according to the guidelines in Appendix 16.A.1.

16.7.2 User Requirements: UnsubscribeService

Figure 16.29 specifies unsubscription from the perspective of the user. As unsubscription should always be possible, `assert` is used on the receipt message unsubscribed back from the system to the user.

16.7.3 System Specification: SubscribeService

Again, we turn to how the system should implement the required functionality. To handle subscriptions, we add a new component `MemberList` to the system as illustrated by the composite structure diagram in Figure 16.30. `MemberList` should maintain a list of service providers and a list of service requesters (not shown in the diagram).

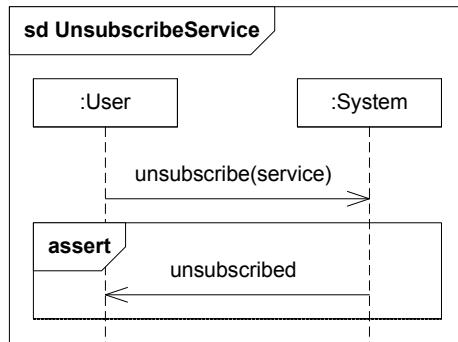


Figure 16.29: Unsubscribe service — user view

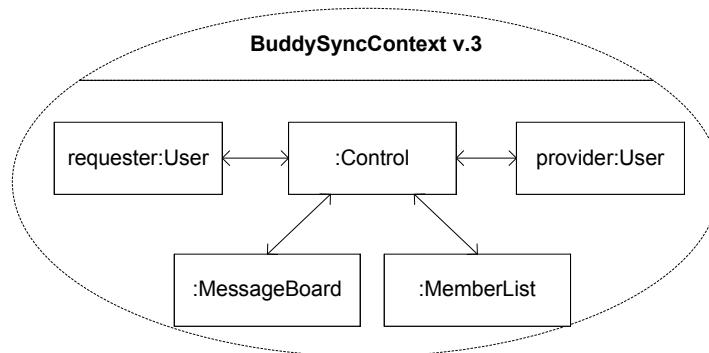


Figure 16.30: Revised composite structure diagram

Figure 16.31 specifies how SubscribeService in Figure 16.28 should be implemented by the system. The messageboard is not relevant for subscriptions, but the control and memberlist are. As before, we have that the communication with the user stays the same, meaning that Figure 16.31 is a detailing refinement with the lifeline mapping:

$ID[:Control \mapsto :System][:MemberList \mapsto :System]$

16.7.4 System Specification: UnsubscribeService

Figure 16.32 specifies how UnsubscribeService in Figure 16.29 should be implemented by the system. Again, Appendix 16.A.2 gives that this is a detailing refinement with the same lifeline mapping as in Section 16.7.3.

16.7.5 System Specification: RequestService Updated

As noted in Section 16.4, the specification of RequestService should be updated so that only subscribed users may request services. This is achieved by the specification in Figure 16.33.

Compared to the previous specification in Figure 16.22, messages are added between Control and MemberList in order to check for subscription. As a result of this checking, the specification

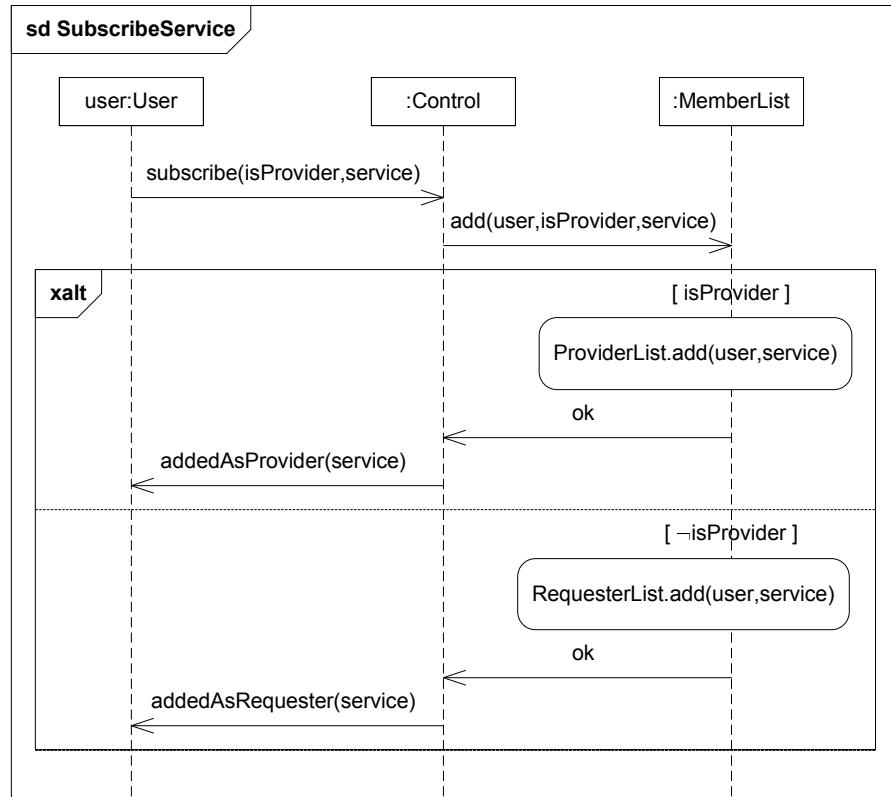


Figure 16.31: Subscribe service — with control and memberlist

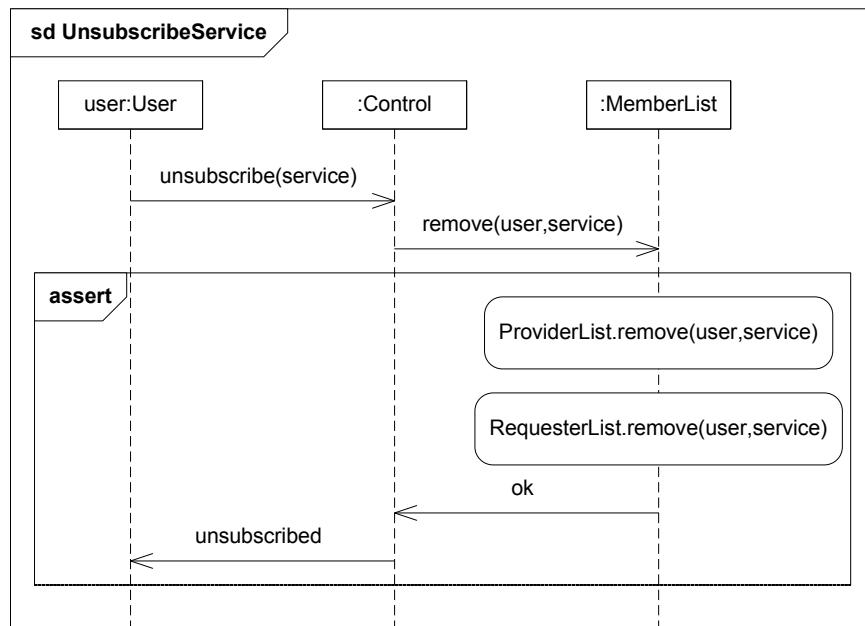


Figure 16.32: Unsubscribe service — with control and memberlist

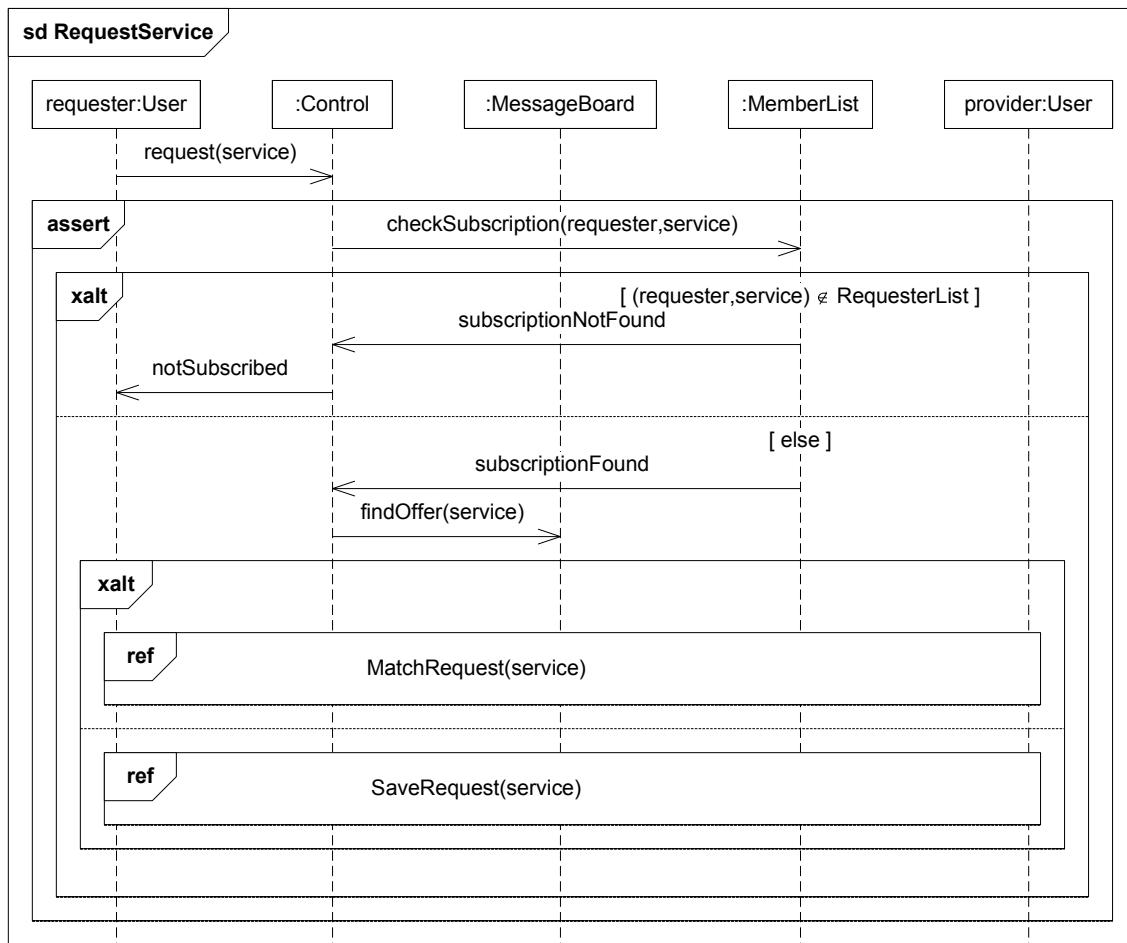


Figure 16.33: Request service — with subscription checking

includes a new alternative where the message `notSubscribed` is sent to the requester. This alternative is also specified with `xalt` according to the guidelines in Appendix 16.A.1, as it should be the alternative when the necessary subscription is not found. Finally, the `alt+refuse` constructs used in Figure 16.22 are removed, and an `assert` is added to the complete specification.

We now have to check that Figure 16.33 is a valid refinement of Figure 16.22. First of all, we note that it will have to be a detailing refinement, with the lifeline mapping:

$$ID[:MemberList \mapsto :Control]$$

A standard way of checking for refinement is to separately take each `xalt`-operand of the original diagram, and find a refining `xalt`-operand in the new diagram.

The first `xalt`-operand in Figure 16.22 corresponds to a scenario that starts with the messages `request(service)` and `findOffer(service)`, and then performs `MatchRequest(service)` and *not* `SaveRequest(service)`. This scenario is found in Figure 16.33 by choosing the second operand of the outer `xalt`, and then the first operand of the inner `xalt`. Here, `SaveRequest` is not explicitly negative, but its traces still become negative due to the outer `assert`. As all original positive and negative traces are kept, according to the guidelines in Appendix 16.A.2 this is a valid supplementing refinement where the `assert` construct adds more negative traces to the specification.

Similarly, the second `xalt`-operand in Figure 16.22 is found in Figure 16.33 by choosing the second operand of both `xalt`'s, making this another instance of supplementing refinement.

Finally, we notice that Figure 16.33 also adds a new `xalt`-alternative to the specification, which is a valid refinement step.

To conclude, Figure 16.33 is a valid general refinement of Figure 16.22 combining supplementing and detailing. This is also a limited refinement, as the first `xalt`-operand in Figure 16.33 specifies behaviour that is inconclusive in Figure 16.22.

16.7.6 System Specification: OrderService Updated

Also `OrderService` must be updated with subscription checking. This is done in Figure 16.34. With an argument similar to the one performed for `RequestService` in the previous section, we may conclude that Figure 16.34 is a valid general refinement of Figure 16.23 combining supplementing and detailing. It is also a valid limited refinement.

16.7.7 Finishing the Iteration

Before ending this iteration, the last one, we take a final check that the resulting diagrams (Figures 16.31, 16.32, 16.33 and 16.34) are as recommended by the guidelines in Appendix 16.A.1.

For the choice between `alt` and `xalt`, this has been checked each time one of these has been used in the specifications. Concerning the guards in Figure 16.31 these are obviously sufficiently covering as required by the guidelines in Appendix 16.A.1. Adding a user as provider should only be performed if the `isProvider` flag is true, and adding him as a requester should only be performed if the flag is false. For the guards in Figures 16.33 and 16.34, these are also correct as the error message `notSubscribed` should be a result only if the user is not subscribed to the service, and continuing to handle the request or offer should be done only if the user is subscribed.

With respect to negation (Appendix 16.A.1), Figures 16.32, 16.33 and 16.34 all contain `assert`, meaning that the specification also includes a number of negative traces as recommended.

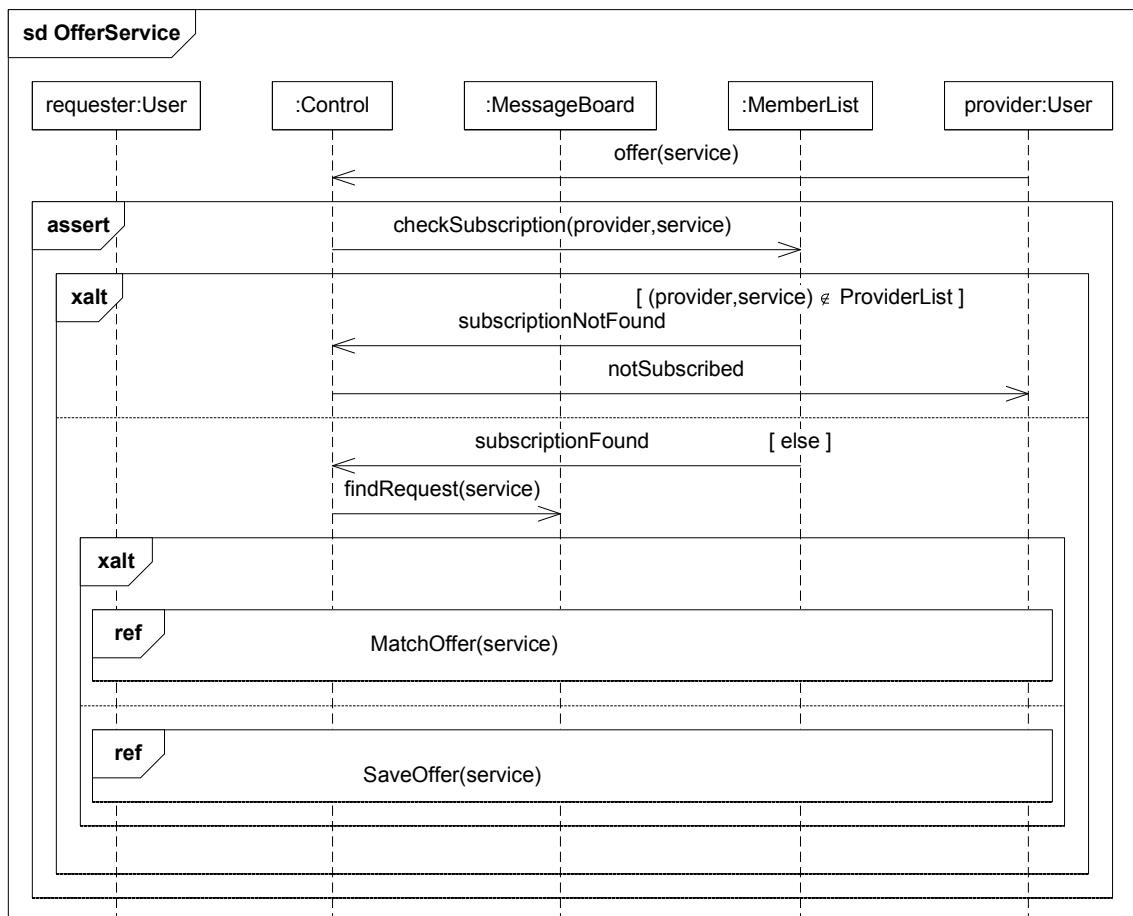


Figure 16.34: Offer service — with subscription checking

Figure 16.31 contains some negative traces (the traces with a false guard), but we could consider adding an assert here as well. For simplicity, the resulting diagram is not shown here as it is not important for the case study.

16.8 Discussion

16.8.1 Validating the Specification

In this section we validate the final specification with respect to the initial example scenarios given in Section 16.3. As the interactions in Figures 16.1 and 16.2 do not contain any negative behaviours, every possible interaction will be a valid general refinement of these two. For validating that our specification is in accordance with the intention behind Figures 16.1 and 16.2, we must instead use informal reasoning.

In Figure 16.2, the user first requests a taxi, and then removes the request again. Taking the message `requestTaxi(upos)` as an instance of the generic message `request(service)`, `requestReceived` as `requestRegistered` and `removeRequest` as `removeRequest(service)`, the given sequence may be found in the specification by first performing `RequestService` as specified in Figure 16.33, choosing the alternative with `SaveRequest`, and then performing `RemoveRequest` as specified in Figure 16.26.

In Figure 16.1, a user requests a taxi in parallel (or sequence) with a taxi offering its service, after which the system notifies both parts. Even with the natural translation of the messages, this scenario is not found as positive (but rather as negative due to the use of `assert`) in our final specification. The reason for this is that if the user's request comes first, this will already be registered when the offer from the taxi arrives, meaning that according to our specification the taxi does not need a receipt-message. Instead he is immediately notified with a message to actually perform the service. This means that although the original scenario is not explicitly covered, the combination of `RequestService` (Figure 16.33) and `OfferService` (Figure 16.34) results in a scenario that fulfils the same purpose as the one intended in Figure 16.1, and we may conclude that the final specification is in accordance with the initial example scenarios.

16.8.2 Evaluating STAIRS

In this section we evaluate STAIRS with respect to the evaluation criteria in Section 16.2, based on the specification of the BuddySync system as presented in Sections 16.5–16.7.

- a. **All relevant knowledge should be expressible.**

For the BuddySync system, we have encountered no problems expressing the desired functional requirements. In Section 16.3, it is suggested that the communication between the system and its users is performed via SMS. This is not described in the interactions, but could have been included by e.g. using the name `UserMobile` instead of `User` for the requester and the provider lifelines.

- b. **The concepts should be general.**

All concepts used are general in the sense that they are not tailored towards this particular application, but may be used to capture a number of different requirements.

c. The concepts should be composable.

Using the various compositions operators (in particular opt (alt), xalt, refuse and assert), it has been easy to group alternative positive behaviours for the same system functionality, and also to specify the related negative behaviours together with the positive behaviours.

d. Both precise and vague knowledge should be expressible.

In this case study, vague knowledge has primarily been expressed using abstraction, i.e. by not describing system details that are irrelevant or not known. For instance, it is not described how the messageboard and memberlist maintain their lists or the exact nature of these lists (sets, unordered/ordered sequences, ...).

Another way to describe vague knowledge in STAIRS is using underspecification (i.e. alt), describing possible alternative behaviours for the system.

e. The concepts should be easily distinguished from each other.

STAIRS includes two operators, alt and xalt, for describing alternative behaviours. Using the guidelines in Appendix 16.A.1, it has been easy to decide which one of these to use in each particular situation.

We also have three operators, assert, refuse and veto, for describing negative behaviours. The difference between assert and the two others are obvious, but there is only a small difference between refuse and veto. However, using the guidelines in Appendix 16.A.1, it has been easy to select between them.

In the specification of e.g. RequestService in Figure 16.13, the constraints in the referenced interactions MatchRequest(service) (Figure 16.14) and SaveRequest(service) (Figure 16.16) are interpreted as guards. From this, it follows that the distinction between constraints and guards is not obvious. Also, the similarities and differences between these two concepts are not covered by any of the guidelines in Appendix 16.A.

f. A concept should mean the same thing every time it is used.

The meaning of each concept is independent of the context in which it is used. As can be seen from the guidelines in Appendix 16.A.1, xalt may be used in different situations, but the underlying semantics is always that all alternatives must be reflected in the final implementation.

g. The concepts should allow flexibility in the level of detail.

Our different specifications of the same functionality (e.g. Figures 16.5, 16.6, 16.13, 16.22 and 16.33 of RequestService) demonstrates that the constructs are suitable for creating interactions with a varying degree of detail.

h. It should be possible to divide the models into natural parts.

In the specification of the BuddySync system, one interaction was created for each main part of the functionality. In addition, sub-interactions were created wherever natural in order to increase readability and the possibility of reuse.

i. **The most frequent kinds of requirements should be expressible in a compact form.**

The use of xalt-alt-refuse together with ref as illustrated in Figures 16.22 and 16.23, is a common pattern in STAIRS specifications, specifying that the positive behaviours of the first xalt-operand should be negative for the second xalt-operand and vice versa. This combination of operators is not at all compact, and adding a new high-level operator for this use should be considered in future work on STAIRS.

Apart from this, we find that all requirements are expressed fairly compact.

j. **The mapping from syntactic constructs to the underlying concepts must be unambiguous.**

In STAIRS, there is a one-to-one mapping between the main concepts and the syntactic constructs used to express these.

k. **The constructs should be easily distinguished from each other.**

STAIRS does mainly use the syntax of UML 2.x interactions, which are not part of what is being evaluated here. The STAIRS operators refuse and veto are easily distinguished syntactically. The STAIRS operators alt and xalt are more similar, but we have never experienced any problems with these either. The 'x' in xalt makes the difference between xalt and alt clearly visible.

l. **A construct should represent the same concept in all contexts.**

This follows from the one-to-one mapping between constructs and concepts.

m. **The constructs should be composable.**

All of the advanced interaction operators of UML 2.x are composable in the sense that they may be nested to an arbitrary depth. The same applies to the specific STAIRS operators.

n. **Constructs without any information should be avoided.**

In the specification of the BuddySync system, none of the interactions contain any informationless construct.

In general, it would probably be easy to make constructs without any information if that was the aim. However, using common UML techniques and the guidelines in Appendix 16.A, we believe that all resulting constructs will provide meaningful information.

o. **The refinement relations should be powerful enough to capture all refinement steps made in practice.**

For the BuddySync system, all interactions specifying the same functionality could be related using the notion of refinement. With respect to the relation between e.g. Figures 16.5 and 16.6, this may be seen as a detailing refinement. However, the guidelines in Appendix 16.A.2 associate detailing with decomposition, which is a term that intuitively does not fit very well with respect to these two figures. A possible solution to this could be to extend the guidelines in order to capture that detailing may mean more than just decomposition.

p. The refinement relations should be general.

All refinement relations used are general in the sense that they are not defined for this particular application, but may be used for a number of different specifications.

q. The refinement relations should be easily distinguished from each other.

Using the guidelines in Appendix 16.A.2, it is easy to distinguish between supplementing, narrowing and detailing. As demonstrated by the BuddySync specifications, most refinement are a combination of these, and it is easy to find out which of the refinement relations that explains each of the changes in the interactions.

However, the distinction between general and limited refinement is not clear from the guidelines, which gives no help for e.g. finding out whether O_rerService in Figure 16.34 is a limited refinement of O_rerService in Figure 16.23 or not.

r. It should be possible to refine the different parts of a specification separately.

Separate refinement is possible due to the monotonicity results in previous papers on STAIRS, and we have used this for refining the different system functionality separately, and also for performing separate refinement of sub-interactions such as e.g. SaveRequest(service) (Figures 16.8, 16.16 and 16.21) and SaveO_rer(service) (Figures 16.11 and 16.19).

However, this is not explicitly covered by the guidelines in Appendix 16.A.2. For relating the specifications of RequestService in Figures 16.22 and 16.33, we used that for general refinement we may take each xalt-operand of the original diagram and find a refining xalt-operand in the second diagram. This is also not covered by the guidelines. Implicitly, we also used the fact that with general refinement, new xalt-operands may be added freely to the specifications.

The main difficulty we encountered during the specification of the BuddySync system, was how and when to use assert. For instance, it was not clear if it could be used in the specifications of RequestService and O_rerService in Figures 16.22 and 16.23. Also, it was difficult to decide on how much should be covered by assert in Figures 16.26, 16.27 and 16.32.

16.9 Conclusions

The presented case study has demonstrated the usefulness of the STAIRS method. As argued in Section 16.8.2, most of the evaluation criteria from Section 16.8.2 have been met. In particular, the guidelines given in [RHS06] proved to be very useful, but they did not cover everything needed for the specifications in this case study. In particular, the following guidelines would have been useful:

- General refinement: Except from the operands of assert, all operands in an interaction may be refined separately.
- General refinement: With general refinement, all xalt-operands of the original interaction must be reflected in the refinement, but new xalt-operands may be added freely.

- Limited refinement: With limited refinement, new `xalt`-operands may be added to the interaction only if the specified behaviour is a refinement of some behaviour specified in the original interaction. In particular, behaviour that is not described by the original interaction (i.e. inconclusive behaviour) may be added in the new `xalt`-operands.

We have also identified the need for more guidelines with respect to constraints and guards, and the use of `assert`. As it is not obvious what the necessary guidelines are, we leave the formulation of these to future work. Also, a natural next step for future research on STAIRS would be to use it in a real world project where an actual implementation is created.

Acknowledgements. We thank Øystein Haugen and Ketil Stølen for useful feedback on previous versions of this paper.

References

- [HHR05a] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. STAIRS towards formal design with sequence diagrams. *Journal of Software and Systems Modeling*, 22(4):349–458, 2005.
- [HHR05b] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. Why timed sequence diagrams require three-event semantics. In *Scenarios: Models, Transformations and Tools*, volume 3466 of *LNCS*, pages 1–25. Springer, 2005.
- [Kro05] John Krogstie. Quality of UML. In *Encyclopedia of Information Science and Technology (IV)*, pages 2387–2391. Idea Group, 2005.
- [Kru04] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, third edition, 2004.
- [KS03] John Krogstie and Arne Sølvberg. *Information Systems Engineering: Conceptual Modeling in a Quality Perspective*. Kompendieforlaget, Trondheim, Norway, 2003.
- [OMG06] Object Management Group. *UML 2.1 Superstructure Specification*, document: ptc/06-04-02 edition, 2006.
- [RHS05a] Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen. How to transform UML neg into a useful construct. In *Norsk Informatikkonferanse NIK’2005*, pages 55–66. Tapir, 2005.
- [RHS05b] Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen. Refining UML interactions with underspecification and nondeterminism. *Nordic Journal of Computing*, 12(2):157–188, 2005.
- [RHS06] Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen. The pragmatics of STAIRS. In *Proc. 4th Int. Symposium on Formal Methods for Components and Objects (FMCO’05)*, volume 4111 of *LNCS*, pages 88–114. Springer, 2006.
- [RRS06] Atle Refsdal, Ragnhild Kobro Runde, and Ketil Stølen. Underspecification, inherent nondeterminism and probability in sequence diagrams. In *Proc. Formal Methods for Open Object-Based Distributed Systems (FMOODS 2006)*, volume 4037 of *LNCS*, pages 138–155. Springer, 2006.

16.A Guidelines from “The Pragmatics of STAIRS”

For easy reference, this appendix includes the guidelines given in [RHS06].

16.A.1 The Pragmatics of Creating Interations

The Pragmatics of alt vs xalt

- Use alt to specify alternatives that represent similar traces, i.e. to model
 - underspecification.
- Use xalt to specify alternatives that must all be present in an implementation, i.e. to model
 - inherent nondeterminism, as in the specification of a coin toss.
 - alternative traces due to different inputs that the system must be able to handle;
 - alternative traces where the conditions for these being positive are abstracted away.

The Pragmatics of Guards

- Use guards in an alt/xalt-construct to constrain the situations in which the different alternatives are positive.
- Always make sure that for each alternative, the guard is sufficiently general to capture all possible situations in which the described traces are positive.
- In an alt-construct, make sure that the guards are exhaustive. If doing nothing is valid, specify this by using the empty diagram, skip.

The Pragmatics of Negation

- To effectively constrain the implementation, the specification should include a reasonable set of negative traces.
- Use refuse when specifying that one of the alternatives in an alt-construct represents negative traces.
- Use veto when the empty trace (i.e. doing nothing) should be positive, as when specifying a negative message in an otherwise positive scenario.
- Use assert on an interaction fragment when all possible positive traces for that fragment have been described.

16.A.2 The Pragmatics of Refining Interactions

The Pragmatics of Supplementing

- Use supplementing to add positive or negative traces to the specification.
- When supplementing, all of the original positive traces must remain positive and all of the original negative traces must remain negative.
- Do not use supplementing on the operand of an assert.

The Pragmatics of Narrowing

- Use narrowing to remove underspecification by redefining positive traces as negative.
- In cases of narrowing, all of the original negative traces must remain negative.
- Guards may be added to an alt-construct as a legal narrowing step.
- Guards may be added to an xalt-construct as a legal narrowing step.
- Guards may be narrowed, i.e. the refined condition must imply the original one.

The Pragmatics of Detailing

- Use detailing to increase the level of granularity of the specification by decomposing lifelines.
- When detailing, document the decomposition by creating a mapping L from the concrete to the abstract lifelines.
- When detailing, make sure that the refined traces are equal to the original ones when abstracting away internal communication and taking the lifeline mapping into account.

The Pragmatics of General Refinement

- Use general refinement to perform a combination of supplementing, narrowing and detailing in a single step.
- To define that a particular trace *must* be present in an implementation use xalt and assert to characterize an obligation with this trace as the only positive one and all other traces as negative.

The Pragmatics of Limited Refinement

- Use assert and switch to limited refinement in order to avoid fundamentally new traces being added to the specification.
- To specify globally negative traces, define these as negative in all operands of xalt, and switch to limited refinement.