SINTEF

A19749- **Unrestricted**

Report

Stepwise refinement of sequence diagrams with soft real-time requirements

Author(s)
Atle Refsdal
Ragnhild Kobro Runde, Ketil Stølen

SINTEF ICT
Networked Systems and Services
2011-09-08

SINTEF

SINTEF IKT
SINTEF ICT

Address:
Postboks 124 Blindern
NO-0314 Oslo
NORWAY

Telephone:+47 73593000
Telefax:+47 22067350

postmottak.IKT@sintef.no
www.sintef.no
Enterprise /VAT No:
NO 948 007 029 MVA

KEYWORDS:
Soft real-time specification, sequence diagram, refinement, probabilistic choice

Report

Stepwise refinement of sequence diagrams with soft real-time requirements

Subtitle

VERSION
1

DATE
2011-09-08

AUTHOR(S)
Atle Refsdal
Ragnhild Kobro Runde, Ketil Stølen

CLIENT(S)
Research Council of Norway

CLIENT'S REF.
180052/S10

PROJECT NO.
90B245

NUMBER OF PAGES/APPENDICES:
41 + 2 Appendices

ABSTRACT

Abstract heading

UML sequence diagrams and similar notations are much used to specify computer systems, serving for example as specifications for programmers, or as a means for validating requirements. When specifying and analyzing computer systems, probabilities are often essential, in particular for capturing soft real-time requirements. It is also important to be able to specify systems at different levels of abstraction, depending on how far the development has progressed and the purpose of the specification. Refinement is a means to relate abstract specifications to more concrete specifications in such a way that requirements and analysis results are preserved through the transition from the abstract to the more concrete level.

This paper presents an approach to extend UML 2.x sequence diagrams to capture probabilistic choice in general and soft real-time requirements in particular. The approach is supported by formal semantics and pragmatic refinement relations. The refinement relations have mathematical properties that allow specifications to be developed in a stepwise and modular manner. An example focusing on communication is provided to demonstrate the use and usefulness of the language and the refinement relations.

PREPARED BY
Atle Refsdal

Atle Refsdal

SIGNATURE

CHECKED BY
Bjørnar Solhaug

Bjørnar Solhaug

SIGNATURE

APPROVED BY
Ketil Stølen

Ketil Stølen

SIGNATURE

REPORT NO.
A19749

ISBN
9788214049855

CLASSIFICATION
Unrestricted

CLASSIFICATION THIS PAGE
Unrestricted

SINTEF

Document history

VERSION	DATE	VERSION DESCRIPTION
1	2011-09-08	First version

Stepwise refinement of sequence diagrams with soft real-time requirements*

Atle Refsdal Ragnhild Kobro Runde Ketil Stølen

September 8, 2011

Abstract

UML sequence diagrams and similar notations are much used to specify computer systems, serving for example as specifications for programmers, or as a means for validating requirements. When specifying and analyzing computer systems, probabilities are often essential, in particular for capturing soft real-time requirements. It is also important to be able to specify systems at different levels of abstraction, depending on how far the development has progressed and the purpose of the specification. Refinement is a means to relate abstract specifications to more concrete specifications in such a way that requirements and analysis results are preserved through the transition from the abstract to the more concrete level.

This paper presents an approach to extend UML 2.x sequence diagrams to capture probabilistic choice in general and soft real-time requirements in particular. The approach is supported by formal semantics and pragmatic refinement relations. The refinement relations have mathematical properties that allow specifications to be developed in a stepwise and modular manner. An example focusing on communication is provided to demonstrate the use and usefulness of the language and the refinement relations.

1 Introduction

UML 2.x sequence diagrams [40] and similar notations are used to specify dynamic or behavioral aspects of computer systems. Sequence diagrams are particularly suited to model communication, which is an essential aspect of most computer systems today. According to [11] and [54], sequence diagrams (and use case diagrams) are the most popular UML languages for modeling the dynamic aspects of a system. Sequence diagrams are used, for example, as specifications

*The research on which this paper reports has been partly carried out within the DIGIT project (180052/S10), funded by the Research Council of Norway, and the SecureChange and NESSoS projects, both funded from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreements FP7-231101 and FP7-256980, respectively.

for programmers or as maintenance documentation, to verify and validate requirements with client representatives, and to clarify the understanding of the application among technical members of a project team [11]. In [36] it is shown how MSCs [24] (which are very similar to UML sequence diagrams) can be used in a number of different phases that occur in most software development methods.

Probabilities sometimes play a major role when specifying and analyzing computer systems. In particular, soft real-time requirements are often important when specifying communication scenarios. By soft real-time requirements we mean requirements such as "when sending a request, the probability of receiving a reply within 5 seconds should be at least 0.95". In other words, soft real-time requirements are real-time requirements that need only be fulfilled with a certain (usually high) probability. Soft real-time requirements are important because the corresponding hard requirements may be impossible or too costly to fulfill.

Refinement relations define what it means for one specification to be a more concrete or detailed representation of another specification. Refinement relations should capture the notion of abstraction in an intuitive manner and ensure that requirements are preserved in the transition from the more abstract to the more concrete specification. Furthermore, an analysis of requirements performed at an abstract specification should remain valid for the more refined specifications. The refinement relations facilitate a development method where more information and details are added as the development process progresses. They also allow specifications to be presented at different levels of abstraction, depending on their purpose and target audience. An important aspect of abstraction is the concept of underspecification. Underspecification means that some choices are left to those responsible for refining the specification or implementing the system, and is an essential feature of specification languages. Reducing the amount of underspecification brings a specification closer to an actual implementation, and is therefore an important form of refinement.

This paper presents an approach to extend UML 2.x sequence diagrams to capture soft real-time, supporting underspecification with respect to probabilities as well as behaviors. We refer to this approach as probabilistic STAIRS, or pSTAIRS. Probabilistic STAIRS offers two different relations capturing refinement with respect to both kinds of underspecification, each relation targeting a different phase of a development process. Both relations have essential transitivity and monotonicity properties allowing the refinement to be conducted in a stepwise and modular fashion.

The rest of the paper is organized as follows: In Section 2 we introduce the pSTAIRS approach. We focus on demonstrating the suitability of probabilistic sequence diagrams to capture soft real-time requirements, as well as the stepwise application of the refinement relations. For this purpose we use an example addressing soft real-time requirements in a communication scenario. Section 3 presents the formal definition of probabilistic choice. In Section 4, we formally define the two refinement relations and present theoretical results of practical importance. In Section 5 we present related work before concluding in Section 6. There are also two appendices: Appendix A provides the formal semantics of

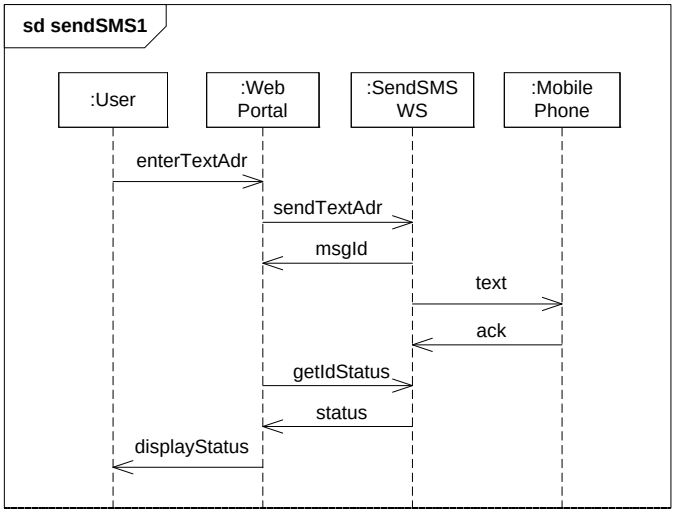

Figure 1: A scenario where an SMS message is sent from a web portal to a mobile phone.

the composition operators referred to (except for the operator for probabilistic choice which is covered by Section 3). Appendix B presents proofs.

2 Probabilistic sequence diagrams – an informal introduction

In this section we demonstrate the suitability of probabilistic sequence diagrams as defined in pSTAIRS to capture interaction scenarios with hard and soft real-time requirements, as well as the use and usefulness of the refinement relations. Only informal explanations are given at this point; formal definitions are provided in Section 3 and Section 4. The presentation is based on a scenario describing SMS-based interaction between a web portal and a mobile phone. We start with a simple diagram, which is further elaborated and refined in subsequent diagrams. This simple diagram is shown in Figure 1. It describes that a user sends an SMS message to a mobile phone from a web portal. The specification is based on [12], but we expand on the original specification in order to illustrate more features of the specification language.

There are four entities taking part in the interaction, each represented by a vertical dashed line called a lifeline. The lifelines :User, :WebPortal, :SendSMSWS, and :MobilePhone represent the user, the web portal, the web service, and the mobile phone, respectively. Messages sent between the lifelines are represented by arrows. Each message gives rise to two events; the arrow tail represents the transmission of the message and the arrow head represents recep-

tion. All communication is assumed to be asynchronous. Events on each lifeline are ordered from top to bottom. In addition, for each message, the transmission event occurs before the reception event. There are no other ordering constraints, unless there are real-time constraints in the diagram.

The first message enterTextAdr represents the user entering the SMS message text and address (phone number) to the web portal. Next, the sendTextAdr message represents the call to the web service from the web portal. The web service responds by sending an SMS message identifier to the web portal, as shown by the msgId message, before sending the SMS text to the mobile phone, as shown by the text message. After receiving this text, the mobile phone acknowledges the SMS by sending the ack message.

After receiving the message id, the web portal asks the web service for the current status of the SMS message, as shown by the getIdStatus message. According to the above ordering rules, the transmission of this message may occur before or after the transmission of the ack message on the mobile phone. However, according to the ordering of events on the :SendSMSWS lifeline, the ack message is received before the getIdStatus message in the scenario described by the sendSMS1 diagram. This restriction will be relaxed later. The :SendSMSWS lifeline responds to the getIdStatus message by sending a status message to the web portal. After receiving the message status, the web portal then displays it to the user, as shown by the displayStatus message. The status of an SMS message can be, for example "MessageWaiting" (the message is still queued for delivery), "DeliveredToNetwork" (the message has been successfully delivered to the network), or "DeliveredToTerminal" (the message has been successfully delivered to the mobile phone). However, we do not go further into this.

The diagram sendSMS1 in Figure 1 captures the scenario in an intuitive and comprehensible manner. However, as mentioned above, according to the diagram, the ack message is received by the web service before the getIdStatus message. In reality, this is not possible to ensure, as it depends on, among other things, at what time the mobile phone is switched on. Therefore we extend the specification in order to express that there are more valid ways of fulfilling this scenario. The result is shown in Figure 2, where a par operator for parallel composition has been introduced to specify that the transmission and reception of the ack message may be interleaved with the transmission and reception of the getIdStatus and status messages. A par operator allows the events of its operators to be interleaved in any manner, as long as the ordering rules are followed for each operator. In our example, this means that the web service will still receive the getIdStatus message before sending the status message, but the ack message may be received at any time with respect to these events. Hence, the diagram in Figure 2 includes the behavior of the diagram in Figure 1, but has also added new behavior that was not described in Figure 1.

Adding new behavior in this way is a kind of refinement that we refer to as *supplementing*. Supplementing is primarily aimed at the early stages of development, where alternative ways of fulfilling a scenario are explored. To understand why we consider supplementing a form of refinement, it is important to remember that sequence diagrams, unlike most specification languages,

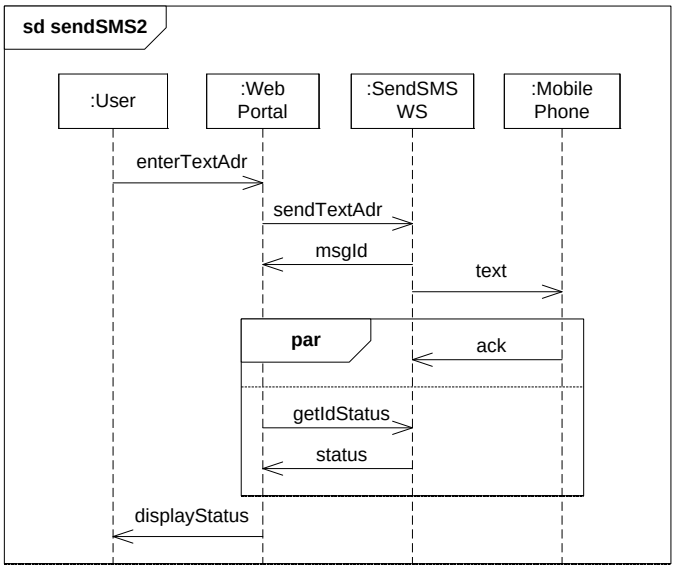

Figure 2: A par operator has been introduced, allowing the transmission and reception of the ack message to be interleaved with the transmission and reception of the getIdStatus and status messages.

give only partial descriptions of behavior. By this we mean that a sequence diagram does not categorize *all* behavior as either positive (acceptable) or negative (unacceptable). In most specification languages, such as state machines, the positive behavior is described explicitly, and all behavior that is not described explicitly is considered to be negative. Sequence diagrams, on the other hand, allow behavior to be described explicitly as positive or negative through dedicated operators. The remaining behavior, which is not described as either positive or negative, is considered to be inconclusive, in the sense that is has not (yet) been decided whether this behavior is acceptable. Hence, the diagram in Figure 1 does not necessarily describe the *only* acceptable way of fulfilling the scenario in question. It simply states that the described behavior is acceptable. We have not used any of the operators for describing negative behavior, therefore all other behavior is inconclusive according to this diagram. In Figure 2 we have reduced the set of inconclusive behavior by specifying some of the behavior that was inconclusive according to Figure 1 as positive. We call this positive supplementing, while describing previously inconclusive behavior as negative is called negative supplementing. Supplementing, whether positive or negative, is a bit similar to broadening the scope of a specification by weakening the pre-condition in the classical pre-post specification paradigm [26], with the following correspondence: Positive behavior in pSTAIRS corresponds to fulfilling both the pre-condition and the post-condition. Negative behavior corresponds to fulfilling the pre-condition, but not the post-condition. Inconclusive behavior corresponds to not fulfilling the pre-condition. Weakening the pre-condition means that more cases fulfill the pre-condition, thus becoming positive or negative, depending on whether the post-condition is fulfilled or not.

In the next refinement step we introduce a hard real-time requirement, i.e. a real-time requirement that should always be fulfilled. In order to allow the web service some time to send the SMS message and receive an acknowledgment from the mobile phone before it is asked for the status of the message, we introduce a requirement stating that there should be a delay of at least 3 seconds from the msgId message is received by the web portal to the getIdStatus message is sent. Figure 3 shows a specification where this requirement has been added. The real-time requirement is expressed in terms of so-called timestamp tags that are assigned to the events. In this case the timestamp tag t1 has been assigned to the reception of the msgId message and the timestamp tag t2 has been assigned to the transmission of the getIdStatus message. The real-time requirement is expressed by a note associated to the transmission of the getIdStatus message that contains a predicate over timestamp tags.[1] In this case the predicate is t2-t1 $\geq$ 3s. All events have a timestamp tag, but only the timestamp tags referred to in a predicate are shown explicitly in the diagram.

There are no real-time requirements in Figure 2. Hence, all time delays are acceptable according to Figure 2, as long as the ordering of events is obeyed. By introducing the real-time requirement in Figure 3, we narrow the range of

[1] We prefer to use this notation rather than the standard UML notation for real-time requirements, as it makes it easier to specify real-time constraints in cases where the relevant events occur in operands of operators.

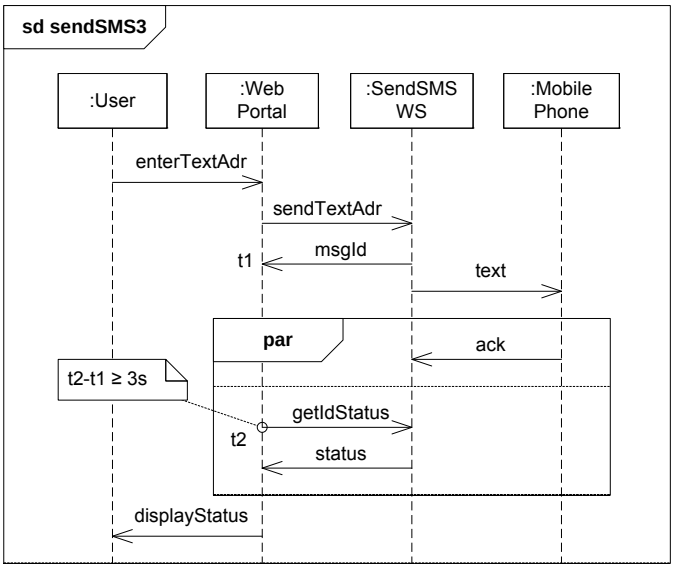

Figure 3: A hard real-time requirement has been introduced stating that there should be a delay of at least 3 seconds from the msgId message is received by the web portal to the getIdStatus is sent.

acceptable behavior by rejecting cases where it takes less than 3 seconds from the msgId message is received by the web portal to the getIdStatus message is sent. This kind of refinement, where previously positive behavior is redefined as negative, for example through introduction of new or stronger requirements, we call *narrowing*.

Next, we show how the specification can be further refined by the introduction of a soft real-time requirement. In pSTAIRS, soft real-time requirements are captured by the use of the operator palt for probabilistic choice. Figure 4 shows a diagram where we have imposed the soft real-time requirement that the probability of sending the displayStatus message from the web portal within 8 seconds after receiving the enterTextAdr message should be at least 0.8. We use the palt operator to achieve this. Both its operands contain the displayStatus message, but only the first operand fulfills the desired real-time requirement. Sets of acceptable probabilities are assigned to the operands after the operand name in the upper left-hand corner of the operator frame in the same order as the operands occur, starting from the top. Hence, the alternative where the predicate t3-t0$\leq$ 8s is fulfilled should occur with a probability in the interval $[0.8, 1]$, whereas the alternative with the complimentary predicate t3-t0$>$8s should occur with a probability in $[0, 0.2]$. The assignment of sets of acceptable probabilities to alternatives, rather than exact probabilities, represents underspecification with respect to probability. Without a means to represent underspecification with

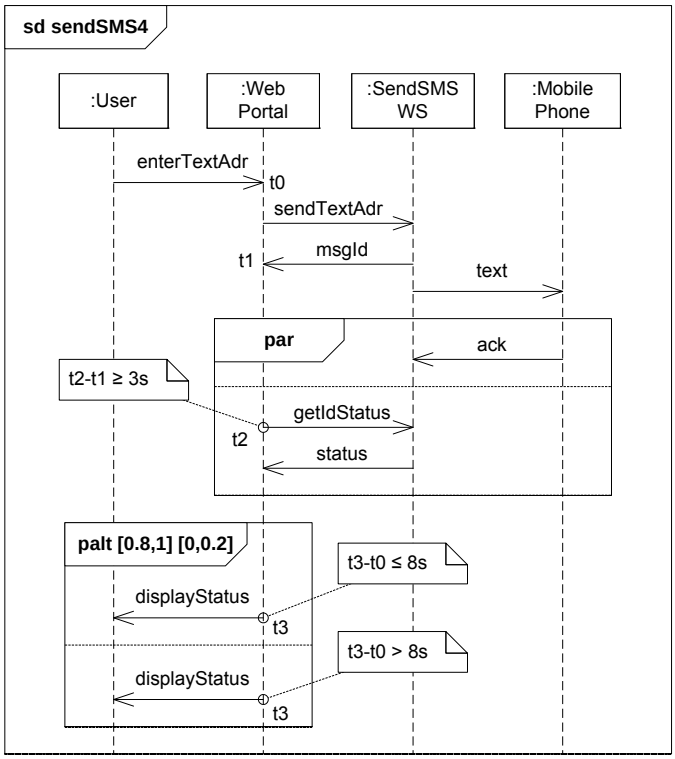

Figure 4: A soft real-time requirement has been added, stating that the probability of sending the displayStatus message on the web portal within 8 seconds after receiving the enterTextAdr message should be at least 0.8.

respect to probability, we would not be able to capture soft real-time requirements, as soft real-time requirements impose limits on acceptable probabilities, rather than exact probabilities.

Note that the introduction of the soft real-time requirement in the diagram sendSMS4 in Figure 4 also constitutes a narrowing refinement of the diagram sendSMS3 in Figure 3, as sendSMS4 does not allow behavior where the probability of a delay of more than 8 seconds from transmission of the enterTextAdr to transmission of the displayStatus message is higher than 0.2.

As the next step of refinement, we supplement the specification with more acceptable behavior. That is, we add the option of displaying the message identifier to the user after the identifier has been received by the web portal. This could be useful, for example, if the user is required to pay for the use of the web portal and wants to confirm that the bill is correct. Figure 5 shows the resulting specification. The message displayId has been added at the appropriate place and enclosed by an opt operator. This operator means that the content of its operand (in this case transmission and reception of the displayId message) is optional. Essentially, it describes two alternatives: one where the content of the operand is executed and one where it is not. These alternatives represent underspecification with respect to system behavior, as they constitute an implementation or design choice left to those responsible of implementing or further refining the specification. Again we note that the soft and hard real-time requirements from the previous specification are preserved.

At a certain point in the development process, we may decide that all relevant behavior has been identified and captured by the specification. From this point onwards, the task is to bring the specification closer to an implementation by reducing underspecification (i.e. making design/implementation choices) and possibly strengthening real-time requirements. Hence, supplementing with new behavior is no longer acceptable, and we therefore adopt a more strict definition of refinement hereafter.

We end the example by conducting a further narrowing of the diagram sendSMS5 in Figure 5. The resulting diagram is sendSMS6 in Figure 6. Three changes have been made, each of which constitutes a narrowing. In order to save space we have included them all in a single diagram, rather than presenting them in a stepwise fashion. First, the opt operator around the displayId message has been replaced by a veto operator. The veto operator is used to express that the behavior of its operand is not acceptable. Thus, this change represents a design decision; the message identifier should not be displayed to the user after being received by the web portal after all. The reason for deciding this could, for example, be that the web designer refuses to clutter the web page with long strings of letters and numbers that does not really mean anything to the user. Second, the acceptable time delay between reception of the enterTextAdr and transmission of the displayStatus message for the "fast alternative" represented by the first palt operand has been reduced from 8 to 6 seconds, and the real-time requirements associated with both operands of the palt have been changed accordingly. Third, the minimum acceptable probability for the first operand has been increased from 0.8 to 0.9.

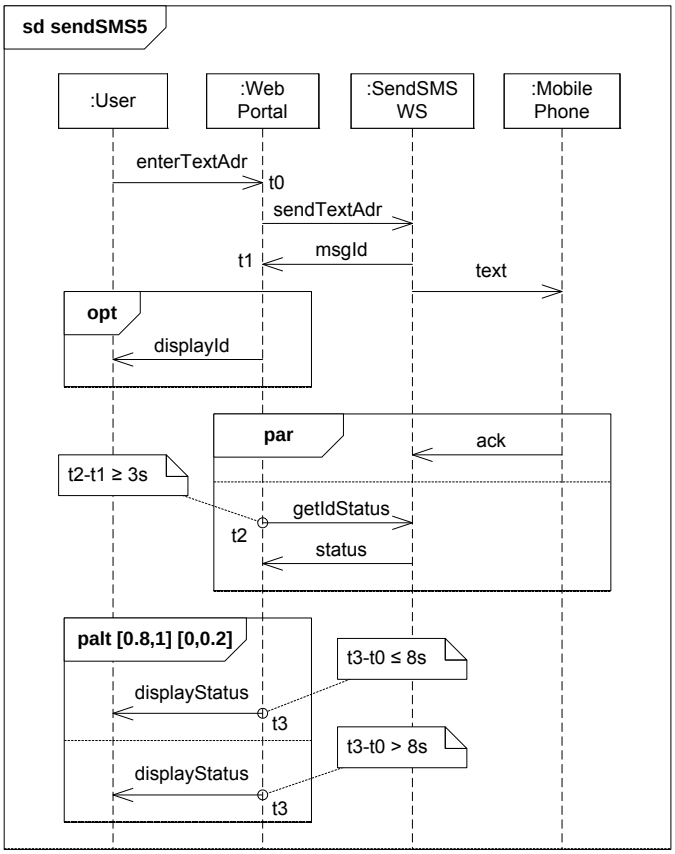

Figure 5: The option of displaying the message identifier to the user have been added.

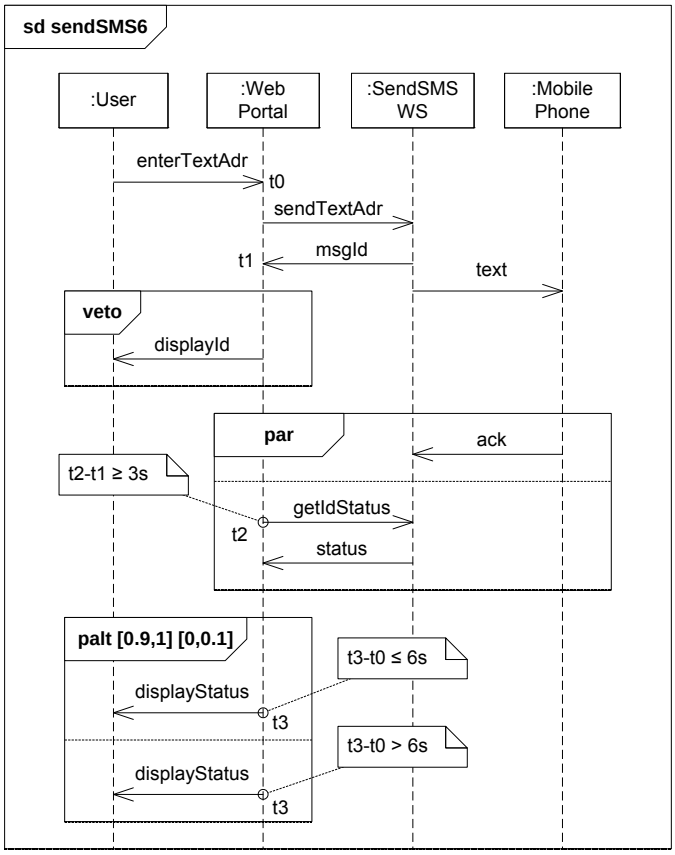

Figure 6: Strengthening of the soft real-time requirement. The probability of sending the displayStatus message on the web portal within 6 seconds after receiving the enterTextAdr message should be at least 0.9.

Together, the two latter changes constitute a strengthening of the soft real-time requirement that was introduced in Figure sendSMS4. Clearly, any system that fulfills the requirements of sendSMS6 also fulfills the requirements of sendSMS5. Thus the requirements of the earlier specifications have also been preserved in this final refinement step.

We round off this informal presentation of pSTAIRS by a simple illustration of how the result of an analysis of requirements remains valid through refinement. Assume we are interested in how long the described scenario may take for the web portal. That is, we ask the question "how long is the delay x from the enterTextAdr message is received by the web portal to the displayStatus message is transmitted?". The specification sendSMS1 does not include any real-time constraints, so the only result we get from analyzing this specification is the trivial observation that "x is at least 0 seconds". The same holds for specification sendSMS2. However, in sendSMS3 we added the requirement that it should take at least 3 seconds from reception of msgId to transmission of getIdStatus. As events are ordered from top to bottom on the :WebPortal lifeline, this means that the delay is at least this long also between the first and final events on this lifeline, even if this is not stated explicitly as a real-time requirement. Hence, from analyzing sendSMS3 we get "x is at least 3 seconds", which is consistent with the previous result, but more specific. After introduction of the soft real-time requirement in sendSMS4, we are able to obtain an even more specific result: "x is at least 3 seconds, and the probability that x is less than or equal to 8 seconds is at least 0.8". The specification sendSMS5 does not add any new information with respect to real-time requirements, and the result of analyzing sendSMS5 with respect to our question remains the same as for sendSMS4. However, in sendSMS6 the soft real-time requirement has been strengthened, allowing us to conclude that "x is at least 3 seconds, and the probability that x is less than or equal to 6 seconds is at least 0.9". Again, this is consistent with the previous analysis, but more specific. Hence, for each refinement of the original specification we have obtained an analysis result that is consistent with all earlier results, but possibly more specific.

3 Probabilistic choice – its formal definition

In this section we first introduce the semantic domain of pSTAIRS, before providing and explaining the formal definition of the operator for probabilistic choice.

3.1 Semantic domain: probability obligations

We base our approach on the semantic model for sequence diagrams taken from the UML 2.1 standard [40]. Here, behavior is represented by traces, which are sequences of event occurrences. Sequence diagrams focus on communication, therefore we are primarily interested in events representing transmission or reception of messages. Since sequence diagrams give only a partial description

of behavior, as discussed in the previous section, UML categorizes the traces described by a sequence diagram as either valid (positive) or invalid (negative); the traces not described are the inconclusive ones as explained above. Hence, the semantics of a UML sequence diagram is represented by a pair of trace sets (p, n) where p denotes the set of positive traces and n denotes the set of negative traces. The inconclusive ones are implicitly represented as the rest of the trace universe.

In pSTAIRS we extend this semantic model in order to capture probability. The semantics of a pSTAIRS specification is given as a set of *probability obligations*, or p-obligations for short. A p-obligation is a pair (o, Q) where $o = (p, n)$ is a pair of a set of positive traces p and a set of negative traces n, and Q is a set of probabilities, i.e. $Q \subseteq [0, 1]$. We use the name *interaction obligation* for pairs (p, n) of sets of positive and negative traces. Every interaction obligation represents an obligation for the specified system; the system is obliged to produce behavior that represents the interaction obligation. An interaction obligation still leaves freedom, as the interaction obligation does not in general require any particular trace to be produced. Instead it allows a number of traces from which the implementer may choose. In this way, an interaction obligation represents implementation freedom or underspecification with respect to system behavior.

Thus, an interaction obligation represents an "abstract piece of behavior" that may be implemented by a number of different traces. From the specifier's point of view, all these traces are considered to be equivalent. Conceptually, it is therefore natural to assign probabilities to interaction obligations, rather than to traces. This also gives a clean separation between underspecification with respect to traces (system behavior) and probabilities; underspecification with respect to traces is captured by the interaction obligation, while the assignment of a *set* of probabilities Q rather than a single probability to each interaction obligation captures underspecification with respect to probability, as the implementer is free to implement the p-obligation with any probability in Q.

3.2 The palt operator for capturing probabilistic choice

The palt operator describes the probabilistic choice between two or more alternative operands whose joint probability should add up to 1. Each operand is assigned a set of probabilities, and each operand should be chosen with a probability in its probability set. Using sets of probabilities rather than a single probability for each operand allows us to capture underspecification with respect to probabilities. In particular, it allows us to specify a minimum probability for an operand, which is essential to capture soft real-time requirements.

Any specification without a palt operator contains exactly one p-obligation, and the probability set of this p-obligation is $\{1\}$. Thus, for specifications without palt operators, the semantic representation according to pSTAIRS corresponds to the semantic representation according to UML, except that the former assigns probability 1 to the pair of positive and negative trace sets. The definition of the palt semantics involves some new operators on p-obligations

and probability sets. We therefore develop this definition in a stepwise manner. First we give two preliminary definitions and explain why these do not work as desired. The preliminary definitions are (3) and (5). Then we present Definition (8), which is the final one. Definition (5) is a strengthening of (3), and (8) is a strengthening of (5).

For operators other than palt, we explain their semantics only to the extent necessary for the examples to be understandable, in order to avoid cluttering the presentation with details that are not important for the main issue of this paper. For the full definitions of these operators, we refer to Appendix A.

We start by introducing the notion of probability decoration, which is used to assign probabilities to each operand of the palt operator. Probability decorations may only occur in the operands of a palt, and is denoted by $d;Q$, where d is a sequence diagram and Q is a set of probabilities. Intuitively, $d;Q$ states that the sequence diagram operand d should be selected with a probability in Q. Semantically, probability decoration is defined by:

$$[\![\ d;Q'\ ]\!] \stackrel{\mathsf{def}}{=} \{(o, Q * Q') \mid (o, Q) \in [\![\ d\ ]\!]\} \tag{1}$$

where multiplication of probability sets is pointwise:

$$Q_1 * Q_2 \stackrel{\mathsf{def}}{=} \{q_1 * q_2 \mid q_1 \in Q_1 \wedge q_2 \in Q_2\} \tag{2}$$

In the textual syntax, a palt operator and its operands is represented by $\mathsf{palt}(d_1;Q_1, \ldots, d_n;Q_n)$. This can be read as "at least one of the operands $d_1, \ldots, d_n$ should be selected; operand d_1 should be selected with a probability in Q_1 and ... and the operand d_n should be selected with a probability in Q_n". It would be intuitively tempting to define the palt semantics as a simple union of its operands. This would give the following definition:

$$[\![\ \mathsf{palt}(d_1;Q_1, \ldots, d_n;Q_n)\ ]\!] \stackrel{\mathsf{pre}}{=} \bigcup_{j=1}^{n} [\![\ d_j;Q_j\ ]\!] \tag{3}$$

where we use $\stackrel{\mathsf{pre}}{=}$ to highlight that this is a preliminary definition. However, Definition (3) is not satisfactory. The reason is that the definition does not ensure that the probabilities of the operands are chosen so that they add up to one.

To see this, assume we want to specify a soft real-time constraint where there is an upper limit to the time delay that should not be exceeded in any run. Diagram paltEx in Figure 7 shows such a specification. Intuitively, the paltEx specification requires that the probability is at least 0.8 that the displayStatus message is transmitted from the web portal at most 8 seconds after reception of enterTextAdr, and that the delay is between 8 and 12 seconds in the remaining cases. Hence, the traces described by paltEx can be divided into three trace sets s_1, s_2, s_3, where s_1 denotes the set of traces where the delay between reception of enterTextAdr and transmission of displayStatus is at most 8 seconds, s_2 denotes the set of traces where the delay is more than 8 seconds but no more than 12

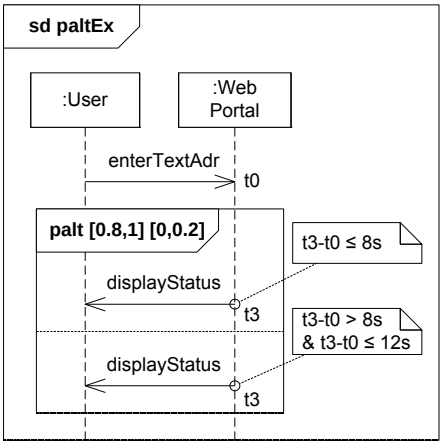

Figure 7: A specification of a soft real-time requirement with an absolute upper limit for the time delay.

seconds, and s_3 denotes the set where the delay is more than 12 seconds. With Definition (3), we get

$$[\![\ \mathsf{paltEx}\ ]\!] = \{((s_1, s_2 \cup s_3), [0.8, 1]), ((s_2, s_1 \cup s_3), [0, 0.2])\}$$

where the first p-obligation represents the first palt operand, and the second p-obligations represents the second palt operand.

But this semantics does not ensure that probabilities are chosen from each p-obligation in such a way that they add up to 1. For example we may select 0.8 as probability for the first operand, 0.1 for the second, and still have room for a refinement step adding a third operand with probability 0.1 allowing delays of more than 12 seconds.

To ensure that the chosen probabilities of the operands add up to 1, we strengthen the palt semantics with an additional p-obligation representing the combination of all the p-obligations we obtain from the operands. The only acceptable probability for this combined p-obligation is 1. This formalizes that one of the operands must be chosen; i.e. the probabilistic choice will be made among the specified operands. For the paltEx specification this means that we add a p-obligation $((p, n), \{1\})$ representing the combination of the two alternatives. The positive and negative traces of this combined p-obligation are determined by the interaction obligations of the original p-obligations $((s_1, s_2 \cup s_3), [0.8, 1])$ and $((s_2, s_1 \cup s_3), [0, 0.2])$. If a trace is positive in one of these then it is acceptable for the system to produce this trace. Therefore, if a trace is positive in at least one p-obligation (and not inconclusive in any p-obligation) then it is positive in the combined p-obligation. For the paltEx specification this means that traces in $s_1 \cup s_2$ are positive. If a trace is negative in all the original p-obligations then this means that it should not be produced at all. Hence it

is also negative in the combined p-obligation. For the paltEx specification this means that traces in s_3 are negative. If a trace is inconclusive in at least one of the original p-obligations then it has not been considered for all alternatives. It is therefore considered to be inconclusive also in the combined p-obligation. In the paltEx specification the set of inconclusive traces is the same for each p-obligation - from a practical point of view this is normally advisable.

The interaction obligation of the combined p-obligation is formalized by the $\oplus$ operator, whose operand is a set of p-obligations:

$$\oplus S \stackrel{\mathsf{def}}{=} ((\bigcup_{((p,n),Q)\in S} p) \cap (\bigcap_{((p,n),Q)\in S} p \cup n), \bigcap_{((p,n),Q)\in S} n) \tag{4}$$

As explained, a trace is negative only if it is negative in all p-obligations; a trace is inconclusive if it is inconclusive in at least one p-obligation; otherwise it is positive. In the paltEx specification the interaction obligation of the combined p-obligation is

$$\oplus\{((s_1, s_2 \cup s_3), [0.8, 1]), ((s_2, s_1 \cup s_3), [0, 0.2])\} = (s_1 \cup s_2, s_3)$$

To include the combined p-obligation in the palt semantics we add another line to the palt definition:

$$[\![\ \mathsf{palt}(d_1;Q_1,\ldots,d_n;Q_n)\ ]\!] \stackrel{\mathsf{pre}}{=} \tag{5}$$

$$\bigcup_{j=1}^{n} [\![\ d_j;Q_j\ ]\!] \cup \tag{a}$$

$$\{(\oplus \bigcup_{j=1}^{n} [\![\ d_j;Q_j\ ]\!], \{1\})\} \tag{b}$$

Note that line (b) in Definition (5) implies that nesting of palt operators is significant, as the sum of probabilities for the operands of each particular palt operator should add up to 1. This means that a specification with nested palt operators cannot in general be rewritten into an equivalent specification with only a single palt operator. As an example, consider the specifications nested and flat in Figure 8. The ref constructs are references to other diagrams whose specifications are of no significance here and therefore left out. The specification nested is stricter than flat, because nested requires that the probability of selecting one of d3 and d4 is a value in Q. The reason is that the probability set Q is assigned to the third operand of the outermost palt, which contains the choice between d3 and d4 represented by the innermost palt. This requirement is not present in flat. For example, let $Q = [\frac{1}{4}, \frac{1}{2}]$, which gives $Q * Q = [\frac{1}{16}, \frac{1}{4}]$. According to Flat, it would be acceptable to select d1 with probability $\frac{1}{2}$, d2 with probability $\frac{3}{8}$, d3 with probability $\frac{1}{16}$ and d4 with probability $\frac{1}{16}$. According to Nested, this is not acceptable, since the probability of selecting one of d3 and d4 is then $\frac{1}{8}$, which is not a value in $[\frac{1}{4}, \frac{1}{2}]$. This illustrates the extra expressiveness obtained by including line (b) in Definition (5).

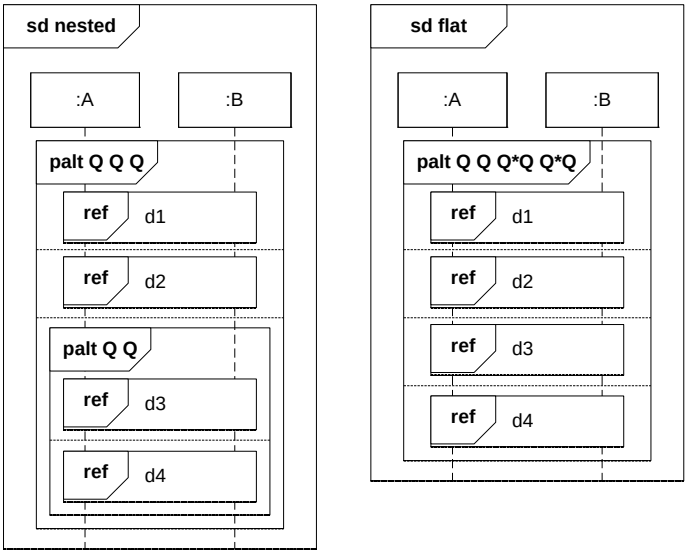

Figure 8: Specification nested, which contains a palt operator within a palt operand, is stricter than specification flat. Q represents probability sets.

But Definition (5) is also not fully satisfactory. The reason is that semantically overlapping operands may be interpreted to correspond to the same probabilistic choice in the implementation, thereby giving the implementation room to include additional behavior not intended by the specification. To avoid this problem we strengthen the semantics of palt with p-obligations representing the combined sum of *any subset* of p-obligations from the original specification. If two p-obligations are overlapping they can no longer be implemented by the same probabilistic choice since the implementation must also offer a choice corresponding to their combination. As before, the interaction obligation of a combined p-obligation is produced by the $\oplus$ operator. But since each new combination represents only a subset of the original p-obligations, we cannot use 1 as the only acceptable probability. Instead we use the sum of the probability sets of each p-obligation of the subset. The combined sum operator $\bar{\oplus}$ combines an indexed set $\{(o_i, Q_i)\}_{i \in N}$ of p-obligations into a single p-obligation as follows:

$$\bar{\oplus}(\{(o_i, Q_i)\}_{i \in N}) \stackrel{\text{def}}{=} (\oplus\{(o_i, Q_i) \mid i \in N\}, \sum_{i \in N} Q_i) \tag{6}$$

Summation of probability sets is pointwise by choosing one value from each set and then adding those combinations that do not exceed 1. Formally, summation of n probability sets is defined by:

$$\sum_{i=1}^{n} Q_i \stackrel{\text{def}}{=} \{\mathsf{min}(\sum_{j=1}^{n} q_j, 1) \mid \forall j : q_j \in Q_j\} \tag{7}$$

Note that $\bar{\oplus}\{(o,Q)\} = (o,Q)$ for any $Q \subseteq [0,1]$.

The following definition of palt, in which line (a) in Definition (5) has been replaced, ensures that all possible combinations of p-obligations coming from the operands of the palt are included:

$$[\![\ \mathsf{palt}(d_1;Q_1,\ldots,d_n;Q_n)\ ]\!] \stackrel{\mathsf{def}}{=} \tag{8}$$

$$\{\bar{\oplus}(\{po_i\}_{i\in N}) \mid N \subseteq \{1,\ldots,n\} \wedge N \neq \emptyset \wedge \forall i \in N : po_i \in [\![\ d_i;Q_i\ ]\!]\} \cup \tag{a}$$

$$\{(\oplus \bigcup_{j=1}^{n} [\![\ d_j;Q_j\ ]\!], \{1\} \cap \sum_{j=1}^{n} Q_j)\} \tag{b}$$

Note that the set of p-obligations we get from (8a) is a superset of the set we get from (5a), since $po = \bar{\oplus}\{po\}$ for any p-obligation po. The palt operator represents a complete probabilistic choice, in the sense that the sum of the probabilities chosen for each operand can not be less than 1. If this cannot be achieved, then no system should comply with the specification. We ensure this by substituting $\{1\}$ in (5b) with $\{1\} \cap \sum_{j=1}^{n} Q_j$. An implemented computer system is assumed to be represented by a set of p-obligations whose probability set contain exactly one probability, as there is no underspecification with respect to probability in an implementation. Therefore, no system can comply with a specification whose semantics contains a p-obligation with an empty probability set.

As indicated above, it is normally advisable to make sure that the set of inconclusive traces for each operand of a palt is the same. We then say that the palt is well-balanced. Formally, $\mathsf{palt}(d_1;Q_1,\ldots,d_k;Q_k)$ is well-balanced if

$$\forall((p_i,n_i),Q_i),((p_j,n_j),Q_j) \in \bigcup_{l=1}^{k} [\![\ d_l\ ]\!] : p_i \cup n_i = p_j \cup n_j \tag{9}$$

Every palt occurring in this paper is well-balanced, and well-balancedness may easily be imposed syntactically. Every p-obligation in the denotation of a well-balanced palt has the same set of inconclusive traces.

4 Refinement of probabilistic sequence diagrams

Refinement means to add more information or stronger requirements to the specification in order to bring it closer to a real system. In this section we define the notion of refinement of probabilistic sequence diagrams formally. Two different refinement relations, aimed at different stages of the development process, are provided.

Our strategy for defining refinement is the following: In Section 4.1 we first define what it means for one interaction obligation to refine another interaction obligation, and then we lift this definition from interaction obligations to p-obligations. In Section 4.2 we use these definitions to define refinement of pSTAIRS specifications, which are represented semantically by sets of p-obligations. After giving the definitions, we present important theoretical results

in Section 4.3. Throughout the section, we also explain how the definitions and theoretical results may be applied to verify that each of the example diagrams in Section 2 is a refinement of the previous one.

4.1 Refinement relations for single interaction obligations and p-obligations

There are two ways in which an interaction obligation may be refined. Firstly, the incompleteness of an interaction obligation may be reduced by supplementing it with more positive and/or negative traces. This reduces the set of inconclusive traces, thereby giving a more complete description of the behavior. Secondly, underspecification with respect to behavior may be reduced by redefining positive traces as negative. This corresponds to making a design decision by rejecting some of the alternative ways of fulfilling a task that are described in the more abstract specification. These two possibilities are combined in the refinement relation $\rightsquigarrow_r$ (where r stands for "refinement"), which is defined as follows:

$$(p, n) \rightsquigarrow_r (p', n') \quad \stackrel{\mathsf{def}}{=} \quad n \subseteq n' \wedge p \subseteq p' \cup n' \tag{10}$$

A refinement relation that allows a specification to be supplemented with new positive or negative behavior, as well as reducing underspecification, is suitable in the early stage of the development process. At this stage, alternative ways of fulfilling the scenario in question (or failing to do so) are explored. Hence, adding new positive or negative behavior to the specification should be allowed.

Later, we may reach a point where all behavior we consider to be relevant and interesting has been described. This includes normal behavior, exceptional behavior and erroneous behavior. At this point, we may decide that supplementing (introducing new traces) is no longer allowed. However, the specification may still include underspecification in the form of several positive traces in the interaction obligation, as some design decisions may still be open. Hence, narrowing the specification by redefining some of the positive traces as negative should still be allowed. This is captured by the refinement relation $\rightsquigarrow_{nr}$ (where nr stands for "narrowing refinement"), defined by:

$$(p, n) \rightsquigarrow_{nr} (p', n') \quad \stackrel{\mathsf{def}}{=} \quad (p, n) \rightsquigarrow_r (p', n') \wedge p \cup n = p' \cup n' \tag{11}$$

We now lift the two definitions of refinement for interaction obligations to p-obligations. A p-obligation is refined by either refining its interaction obligation, or by reducing its set of acceptable probabilities, thus reducing underspecification with respect to probability. This gives the following refinement relations for p-obligations (where p stands for "probabilistic"):

$$((p, n), Q) \rightsquigarrow_{pr} ((p', n'), Q') \quad \stackrel{\mathsf{def}}{=} \quad (p, n) \rightsquigarrow_r (p', n') \wedge Q' \subseteq Q \tag{12}$$

$$((p, n), Q) \rightsquigarrow_{pnr} ((p', n'), Q') \quad \stackrel{\mathsf{def}}{=} \quad (p, n) \rightsquigarrow_{nr} (p', n') \wedge Q' \subseteq Q \tag{13}$$

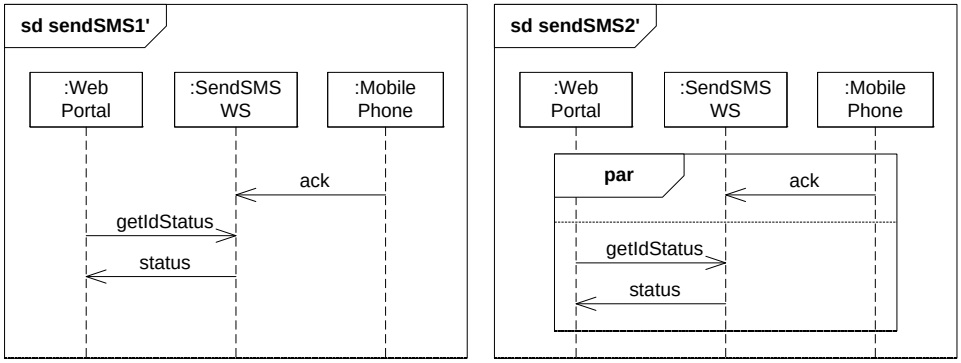

Figure 9: Subdiagrams of sendSMS1 and sendSMS2.

4.1.1 Refinement examples: supplementing of a single p-obligation

Consider the two sequence diagrams in Figure 9, where sendSMS1' is a subdiagram of sendSMS1 in Figure 1 and sendSMS2' is a subdiagram of sendSMS2 in Figure 2. In these two diagrams, no time constraints or other constructs for defining negative behavior are used. Also, the palt operator is not used. This means that the semantics of each of the two diagrams is a single p-obligation where the set of negative traces is empty and the probability set is $\{1\}$. We now demonstrate that the p-obligation for sendSMS2' is a refinement of the p-obligation for sendSMS1'.

For calculating the set of positive traces in sendSMS1', we use the two basic ordering requirements of sequence diagrams: (1) events on each lifeline are ordered from top to bottom; (2) every transmission event occurs before its corresponding reception event. This gives the following traces:

$$\begin{aligned} h_1 &= \langle !a, !gI, ?a, ?gI, !s, ?s \rangle \\ h_2 &= \langle !a, ?a, !gI, ?gI, !s, ?s \rangle \\ h_3 &= \langle !gI, !a, ?a, ?gI, !s, ?s \rangle \end{aligned}$$

Here we use a simplified notation for traces where transmitters, receivers, and timestamps are omitted. In addition, message names are shortened so that a=ack, gI=getIdStatus and s=status. The exclamation mark denotes a transmission event, and the question mark denotes a reception event. Hence, $!a$ denotes the transmission event of the ack message from :MobilePhone to :SendSMSWS, while $?a$ denotes the corresponding reception event. To conclude, we get that the single p-obligation for sendSMS1' is $((p, \emptyset), \{1\})$, where $p = \{h_1, h_2, h_3\}$.

For sendSMS2', we first calculate the traces of each of the par operands separately, and get $h' = \langle !a, ?a \rangle$ as the only trace in the first operand and $h'' = \langle !gI, ?gI, !s, ?s \rangle$ as the only trace in the second operand. The par operator then defines that these two traces can be interleaved in any order. This means

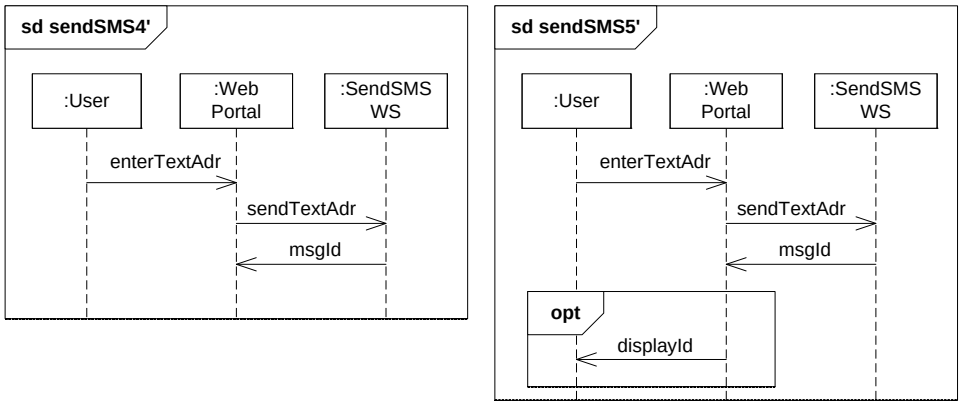

Figure 10: Subdiagrams of sendSMS4 and sendSMS5.

that we get the following new traces in addition to h_1, h_2, h_3:

$$
\begin{array}{ll}
h_4 = \langle !a, !gI, ?gI, ?a, !s, ?s \rangle & h_5 = \langle !a, !gI, ?gI, !s, ?a, ?s \rangle \\
h_6 = \langle !a, !gI, ?gI, !s, ?s, ?a \rangle & h_7 = \langle !gI, !a, ?gI, ?a, !s, ?s \rangle \\
h_8 = \langle !gI, !a, ?gI, !s, ?a, ?s \rangle & h_9 = \langle !gI, !a, ?gI, !s, ?s, ?a \rangle \\
h_{10} = \langle !gI, ?gI, !a, ?a, !s, ?s \rangle & h_{11} = \langle !gI, ?gI, !a, !s, ?a, ?s \rangle \\
h_{12} = \langle !gI, ?gI, !a, !s, ?s, ?a \rangle & h_{13} = \langle !gI, ?gI, !s, !a, ?a, ?s \rangle \\
h_{14} = \langle !gI, ?gI, !s, !a, ?s, ?a \rangle & h_{15} = \langle !gI, ?gI, !s, ?s, !a, ?a \rangle
\end{array}
$$

Thus, we get that the single p-obligation for sendSMS2' is $((p', \emptyset), \{1\})$, where $p' = \{h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, h_{10}, h_{11}, h_{12}, h_{13}, h_{14}, h_{15}\}$. This means, by Definition (10), that $(p, \emptyset) \rightsquigarrow_r (p', \emptyset)$ and consequently $((p, \emptyset), \{1\}) \rightsquigarrow_{pr} ((p, \emptyset), \{1\})$ by Definition (12).

As a second example, consider the two diagrams in Figure 10, where sendSMS4' is a subdiagram of sendSMS4 in Figure 4 and sendSMS5' is a subdiagram of sendSMS5 in Figure 5. As in the previous example, neither of these diagrams uses the palt operator or any constructs for defining negative behavior. The semantics of each of these two diagrams is then a single p-obligation with an empty set of negative traces and $\{1\}$ as the probability set.

The diagram sendSMS4' has a single positive trace $h_a = \langle !eT, ?eT, !sT, ?sT, !mI, ?mI \rangle$, where eT=enterTextAdr, sT=sendTextAdr and mI=msgId. For sendSMS5', the opt operator results in the diagram having two traces, one with and one without the optional message displayId. The trace without displayId is identical to h_a from sendSMS4', while the other trace is $h_b = \langle !eT, ?eT, !sT, ?sT, !mI, ?mI, !dI, ?dI \rangle$, where dI=displayId.

Thus, we get that the single p-obligation for sendSMS4' is $((\{h_a\}, \emptyset), \{1\})$, while the single p-obligation for sendSMS5' is $((\{h_a, h_b\}, \emptyset), \{1\})$. Using definitions (10) and (12), it is easy to verify that $((\{h_a\}, \emptyset), \{1\}) \rightsquigarrow_{pr} ((\{h_a, h_b\}, \emptyset), \{1\})$.

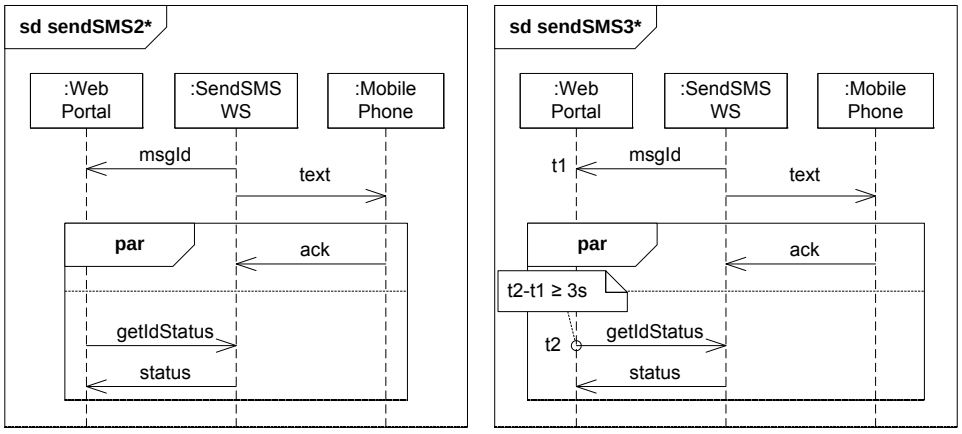

Figure 11: Subdiagrams of sendSMS2 and sendSMS3.

4.1.2 Refinement examples: narrowing of a single p-obligation

The two sequence diagrams sendSMS2* and sendSMS3* in Figure 11 are subdiagrams of sendSMS2 and sendSMS3 in Figures 2 and 3, respectively. As neither diagram uses the palt operator, the semantics of each diagram is again a single p-obligation with probability set $\{1\}$. sendSMS2* has an empty set of negative traces, while sendSMS3* has both positive and negative traces due to the use of a time constraint.

The set p of positive traces for sendSMS2* consists of all traces where weak sequencing is used to combine the messages msgId and text with the traces of the par-fragment. Listing all these traces here would be tedious. For the purposes of this section, it is sufficient to note that since sendSMS2* does not contain time constraints, it allows any possible assignment of timestamps to the individual events in its set of positive traces.[2]

The difference between sendSMS2* and sendSMS3*, is the addition of the time constraint t2-t1 $\geq$ 3s, requiring that the message getIdStatus is sent at least 3 seconds after the msgId message has been received. Semantically, this has the consequence of splitting the positive trace set p for sendSMS2* into two sets p' and n', where traces that are in accordance with this time constraint belong to the positive trace set p', and traces where the time between the two events is less than 3 seconds belong to the set n' of negative traces.

As $p = p' \cup n'$, it is easy to see that the p-obligation $((p', n'), \{1\})$ (for sendSMS3*) is a narrowing refinement of the p-obligation $((p, \emptyset), \{1\})$ (for

[2]As long as there are no time constraints, the only requirement is that the events in a trace are be ordered by time (but two events may happen at the same time). Traces where the timestamps are assigned such that the events are not ordered by time are ill-formed, and not included in the semantic domain (i.e. an ill-formed trace is neither positive, negative, nor inconclusive).

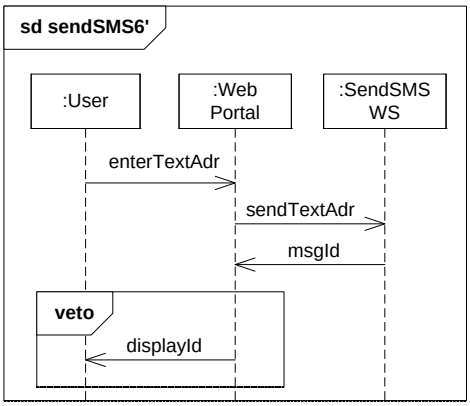

Figure 12: Subdiagram of sendSMS6.

sendSMS2*) according to definition (13).

Another example of a narrowing refinement is given by the diagram sendSMS6' in Figure 12, where the opt operator from sendSMS5' in Figure 10 has been replaced by veto. In Section 4.1.1, we found that the semantics of sendSMS5' was the single p-obligation $((\{h_a, h_b\}, \emptyset), \{1\})$. Semantically, replacing the opt by veto in this context, has the effect of the trace h_b being negative in sendSMS6', and the semantics of sendSMS6' is the single p-obligation $((\{h_a\}, \{h_b\}), \{1\})$.

Redefining a positive trace as negative is a valid narrowing refinement according to definition (11), i.e. $(\{h_a, h_b\}, \emptyset) \rightsquigarrow_{nr} (\{h_a\}, \{h_b\})$, and consequently $((\{h_a, h_b\}, \emptyset), \{1\}) \rightsquigarrow_{pnr} ((\{h_a\}, \{h_b\}), \{1\})$ by definition (13).

4.2 Refinement relations for pSTAIRS specifications

For general refinement of specifications, we define two relations $\rightsquigarrow_{pg}$ and $\rightsquigarrow_{png}$ (where g stands for "general"). The refinement relation $\rightsquigarrow_{pg}$ is based on the relation $\rightsquigarrow_{pr}$ for p-obligations and intended for the early stage of the development process, while $\rightsquigarrow_{png}$ is based on the relation $\rightsquigarrow_{pnr}$ for p-obligations and intended for the late stage of the development process.

As for the definition of palt, we develop the definitions of refinement for specifications in a stepwise manner. We first give a preliminary definition (14) and explain why this is insufficient. Definition (15) is the final definition.

As explained in Section 3.1, the semantics of a pSTAIRS specification is given as a set of p-obligations. Intuitively, each p-obligation represents an abstract class of similar behaviors of which at least one representative is required of the system. Consequently, it is tempting to define refinement of a specification by requiring that every p-obligation at the abstract level should be refined by a p-obligation at the concrete level. This would give the following definition:

$$[\![\ d\ ]\!] \rightsquigarrow_x [\![\ d'\ ]\!] \quad \stackrel{\text{pre}}{=} \quad \forall po \in [\![\ d\ ]\!] : \exists po' \in [\![\ d'\ ]\!] : po \rightsquigarrow_y po' \qquad (14)$$

where $(x, y) \in \{(pg, pr), (png, pnr)\}$.

However, Definition (14) is not satisfactory for soft real-time requirements. Consider the requirement "a request should be followed by a response within 5 seconds with a probability of at least 0.9". A specification replacing this requirement with the corresponding hard real-time requirement (requiring the system to *always* produce a response within 5 seconds) would certainly preserve the original soft real-time requirement, and should therefore be considered a valid refinement, even if the alternative where it takes more than 5 seconds is not represented.

In the refinement definition, this is captured by adding an exception stating that only p-obligations not having 0 as an acceptable probability need to be represented at the concrete level:

$$\begin{aligned} &[\![\ d\ ]\!] \rightsquigarrow_x [\![\ d'\ ]\!] \stackrel{\mathsf{def}}{=} \\ &\quad \forall po \in [\![\ d\ ]\!] : 0 \notin \pi_2.po \Rightarrow \exists po' \in [\![\ d'\ ]\!] : po \rightsquigarrow_y po' \end{aligned} \tag{15}$$

where $(x, y) \in \{(pg, pr), (png, pnr)\}$ and $\pi_2.po$ denotes the second element of a p-obligation, i.e. its probability set.

4.2.1 Refinement examples: supplementing

Consider again the example diagrams in Figure 9. As demonstrated in Section 4.1.1, the semantics of each of these diagrams consists of a single p-obligation, where the p-obligation for sendSMS2' is a (supplementing) refinement of the p-obligation for sendSMS1' according to $\rightsquigarrow_{pr}$. Using Definition (15), it then follows that sendSMS2' is a (supplementing) refinement of sendSMS1' according to $\rightsquigarrow_{pg}$.

Similarly, in Figure 10 sendSMS5' is a (supplementing) refinement of sendSMS4', as the p-obligation for sendSMS5' is a (supplementing) refinement of the p-obligation for sendSMS4'', as demonstrated in Section 4.1.1.

4.2.2 Refinement examples: narrowing

As a first example, consider again the example diagrams in Figure 11. As the single p-obligation for sendSMS3* is a narrowing refinement of the single p-obligation for sendSMS2* (as demonstrated in Section 4.1.2), it follows from Definition (15) that sendSMS3* is a narrowing refinement of sendSMS2*. Similarly, sendSMS6' in Figure 12 is a narrowing refinement of sendSMS5' in Figure 10, as the single p-obligation for sendSMS6' is a narrowing refinement of the single o-obligation for sendSMS5', as demonstrated in Section 4.1.2.

As another example, consider the two diagrams sendSMS3- and sendSMS4- given in Figure 13. These are simplified versions of sendSMS3 (in Figure 3) and sendSMS4 (in Figure 4), respectively, showing only the communication between the :User and :WebPortal lifelines. In sendSMS3- the :User lifeline first sends the enterTextAdr to the :WebPortal, which then sends the dispayStatus message back. No time constraints are given in sendSMS3-. In sendSMS4-, a soft real-time constraint is added, requiring that the probability of the web portal transmitting

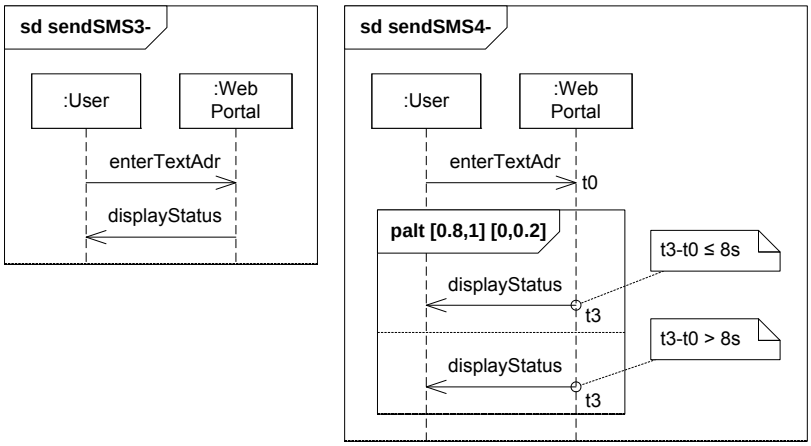

Figure 13: Simplified versions of parts of sendSMS3 and sendSMS4.

the displayStatus message at most 8 seconds after receiving the enterTextAdr message, should be at least 0.8.

Adding a soft real-time requirement like this should constitute a narrowing refinement as the two diagrams describe the same behaviors, with the difference that some of the behavior that was valid in sendSMS3- (i.e. behavior where the probability of the delay between the reception of enterTextAdr and the transmission of displayStatus is higher that 0.2) is not allowed in sendSMS4-.

To see that this is indeed a narrowing refinement, let s_1 denote the set of traces where the delay between reception of enterTextAdr and transmission of displayStatus is at most 8 seconds, while s_2 is the set of traces where the delay is more than 8 seconds.

As sendSMS3- does not include any palt operators, time constraints or other constructs defining negative behavior, it is easy to see that its semantics is a single p-obligation with positive trace set $s_1 \cup s_2$, i.e. $[\![\ \text{sendSMS3}-\ ]\!] = ((s_1 \cup s_2, \emptyset), \{1\})$. For sendSMS4-, we get a total of four p-obligations:

$$po_1 = ((s_1, s_2), [0.8, 1]) \qquad po_2 = ((s_2, s_1), [0, 0.2])$$
$$po_3 = ((s_1 \cup s_2, \emptyset), [0.8, 1]) \quad po_4 = ((s_1 \cup s_2, \emptyset), \{1\})$$

where po_1 and po_2 represent each of the two palt operands, resulting from using Definition (8), part a, with $N = 1$, po_3 represents their combination (using $N = 2$), while po_4 is the final combined p-obligation resulting from Definition (8), line (b).

As po_4 is identical to the single p-obligation for sendSMS3-, Definition (15) is satisfied for both $\rightsquigarrow_{pg}$ and $\rightsquigarrow_{png}$.

As a final example, we demonstrate how the diagram sendSMS6- in Figure 14 is a narrowing refinement of the diagram sendSMS4- in Figure 13. The semantics of the two diagrams are given in Figure 15. Here, $s1$ is the set of traces where

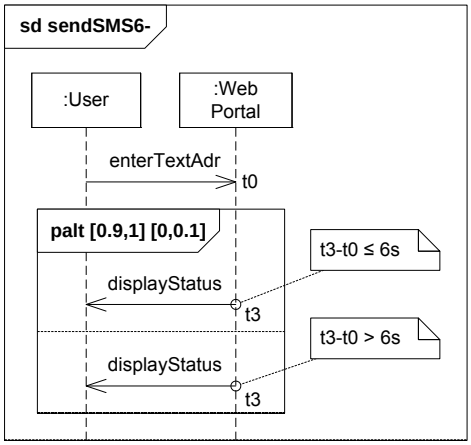

Figure 14: Simplified version of sendSMS6.

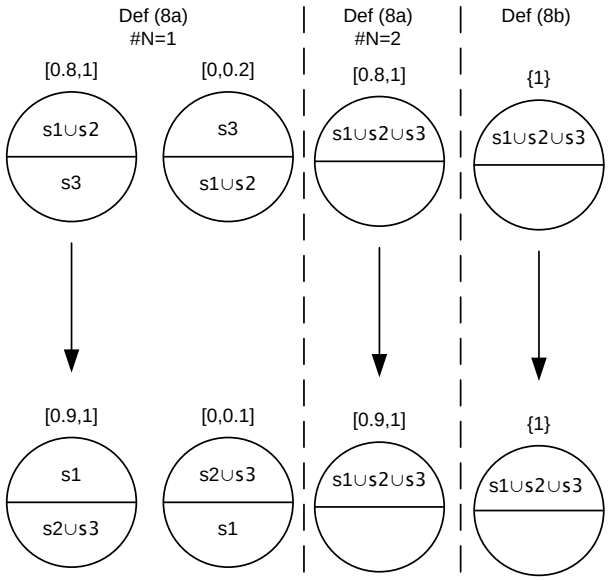

Figure 15: sendSMS4- is refined by sendSMS6-.

$t3 - t0 \leq 6$, $s2$ is the set of traces with $6 \leq t3 - t0 \leq 8$, and $s3$ the set of traces with $t3 - t0 \geq 8$. The upper row represents $[\![$ sendSMS4− $]\!]$ and the lower row represents $[\![$ sendSMS6− $]\!]$. Each p-obligation is illustrated by a circle representing the interaction obligation, with the probability set above the circle.

From Figure 15, we see that each p-obligation for sendSMS4- that does not include 0 in the its probability set, is refined by an p-obligation in sendSMS6-. In each case, it is a narrowing refinement, as the set of inconclusive traces remains the same in the refinement. This means that Definition (15) is fulfilled, and we conclude that sendSMS6- is a narrowing refinement of sendSMS4-.

4.3 Properties of refinement – theoretical results

In the refinement examples in Sections 4.1 and 4.2, we considered subparts of the diagrams from Section 2 and demonstrated how the local modifications constituted valid refinements of those subparts. In this section, we present essential properties of our refinement definitions, and show how these may be used to establish refinement of the complete diagrams, without calculating the full semantics for each diagram.

Since practical specifications may be large, it is important that different parts of a sequence diagram may be refined separately, without considering the rest of the diagram. This is the mathematical properties of monotonicity and modularity, which are studied in Section 4.3.1.

When performing a series of refinement steps, it is also important that the end result refines not only the previous specification, but also the original one. This is the property of transitivity, which is studied in Section 4.3.2.

4.3.1 Monotonicity and modularity

A binary operator *op* is monotonic with respect to refinement if the following holds: If d_1 is refined by d_1' and d_2 is refined by d_2' then d_1 *op* d_2 is refined by d_1' *op* d_2'. Formally, an operator *op* with n operands is monotonic with respect to a refinement relation $\rightsquigarrow$ if the following holds:

$$(\forall i \leq n : [\![\ d_i\ ]\!] \rightsquigarrow [\![\ d_i'\ ]\!]) \Rightarrow op(d_1, \ldots, d_n) \rightsquigarrow op(d_1', \ldots, d_n') \qquad (16)$$

Theorem 1 *The operators* refuse, veto, par, seq, alt, *and* tc[3] *are monotonic with respect to the refinement relations* $\rightsquigarrow_{pg}$ *and* $\rightsquigarrow_{png}$.

Proof For the operators refuse, veto, par, seq, and alt, the proofs are found in [42], while the proof for tc is available in Appendix B. □

In the case of palt, there exist some syntactically correct specifications that do not fulfill the monotonicity requirement, so we do not have full monotonicity. However, we have a slightly weaker modularity result that, under certain conditions that will normally hold for practical specifications, allows us to ensure that any implementation of the more concrete specification will also be an

[3]See Appendix A for descriptions and definitions of these operators.

implementation of the more abstract specification by considering the operands one by one.

For any given refinement relation $\rightsquigarrow_x$, we assume that a corresponding 'implements' relation $\mapsto_x$ is defined in such a way that $d \mapsto_x I$ holds if and only if

1. every p-obligation in $[\![d]\!]$ is implemented by I, and
2. for any p-obligations po, po', if $po \rightsquigarrow_y po'$ and po' is implemented by I then po is also implemented by I,

where $(x, y) \in \{(pg, pr), (png, pnr)\}$ and I is an implementation. Apart from these assumptions we do not define the 'implements' relations, as these will depend on the implementation model and is beyond the scope of this paper, which focuses on sequence diagram specifications.

We say that a sequence diagram operator op with n operands is modular with respect to the pair $(\rightsquigarrow, \mapsto)$ if the following holds:

$$(\forall i \leq n : [\![d_i]\!] \rightsquigarrow [\![d'_i]\!] \wedge d'_i \mapsto I_i) \Rightarrow op(d_1, \ldots, d_n) \mapsto I \tag{17}$$

where I is an implementation of $op(d'_1, \ldots, d'_n)$ composed of the implementations I_i, i.e. $op(d'_1, \ldots, d'_n) \mapsto I$ and $I = op_c(I_1, \ldots, I_n)$, where op_c is a composition operator for implementations corresponding to the sequence diagram operator op.

For all operators except from palt, modularity follows directly from monotonicity and the assumption that the 'implements' relation is preserved through abstraction. In the case of palt we have modularity under two conditions that from a practical point of view are entirely reasonable:

1. The palt is well-balanced.
2. Each operand d_i is safe in the sense that the positive behavior of d_i at infinite time is completely determined by the behavior of d_i at finite time.

Theorem 2 *The operator* palt *is modular with respect to* $(\rightsquigarrow_{pg}, \mapsto_{pg})$ *and* $(\rightsquigarrow_{png}, \mapsto_{png})$ *if it is well-balanced and each of its operands is safe.*

Proof See Appendix B. □

It is normally advisable to make sure that every palt is well-balanced and this condition may be imposed syntactically. The safety condition (formally defined in Appendix B) restricts us from expressing liveness properties. In a real-time notation like probabilistic STAIRS, progress may be expressed directly as time constraint. Hence, in practice there is no need to express liveness properties.

We now argue why each of the diagrams in Section 2 is a valid refinement of the previous one.

The diagram sendSMS1 in Figure 1 may be seen as consisting of three subdiagrams, the first (topmost) four messages, the next three messages (as given

by sendSMS1' in Figure 9), and the final message displayStatus, all composed by the implicit weak sequencing operator seq. Similarly, the diagram sendSMS2 in Figure 2 may be seen as a weak sequencing between the messages above the par operator, the par fragment itself (given as sendSMS2' in Figure 9), and the message displayStatus. In Sections 4.1.1 and 4.2.1, we showed that sendSMS2' is a valid refinement of sendSMS1'. By monotonicity of seq, it follows that sendSMS2 is a refinement of sendSMS1, as the other parts are the same in both diagrams.

Similarly, the part of sendSMS2 that is changed by sendSMS3 in Figure 3 is given by sendSMS2* and sendSMS3* in Figure 11. In Sections 4.1.2 and 4.2.2, sendSMS3* was shown to be a valid narrowing refinement of sendSMS2*. By monotonicity of seq, it follows that sendSMS3 is a valid narrowing refinement of sendSMS2.

By an argument similar to the one given for sendSMS3- and sendSMS4- in Section 4.2.2, it is possible to show that sendSMS4 is a valid narrowing refinement of sendSMS3. In this case, considering only parts of the diagrams and then applying the monotonicity results is not possible, as the time constraints introduced in sendSMS4 stretches across most of the diagram.

In order to show that sendSMS5 is a valid refinement of sendSMS4, it is again sufficient to prove that refinement holds for the affected subdiagram. This we argue in Sections 4.1.1 and 4.2.1 with respect to sendSMS4' and sendSMS5' in Figure 10.

The argument for why sendSMS6 is a narrowing refinement of sendSMS5, consists of two parts. First, changing opt to veto is a valid narrowing refinement as we argue in Sections 4.1.2 and 4.2.2 with respect to sendSMS5' in Figure 10 and sendSMS6' in Figure 12. The changes made to the palt-operator and associated time constraints are also a valid narrowing refinement by an argument similar to the one given in Section 4.2.2 regarding that sendSMS6- in Figure 14 is a narrowing refinement of sendSMS4- in Figure 13.

4.3.2 Transitivity

A refinement relation $\rightsquigarrow$ is transitive if the following holds: If d_1 is refined by d_2 and d_2 is refined by d_3, then d_1 is refined by d_3. Formally:

$$[\![d_1]\!] \rightsquigarrow [\![d_2]\!] \wedge [\![d_2]\!] \rightsquigarrow [\![d_3]\!] \Rightarrow [\![d_1]\!] \rightsquigarrow [\![d_3]\!] \tag{18}$$

Theorem 3 *The refinement relations $\rightsquigarrow_{pg}$ and $\rightsquigarrow_{png}$ are transitive.*

Proof See Theorems 12 and 15 in [42]. The proof for $\rightsquigarrow_{png}$ is also outlined in Appendix B. □

In the previous subsection, we argued why each of the example diagrams in Section 2 is a refinement of the previous one. By transitivity, we get that each diagram is a refinement of all of the previous diagrams, and in particular that the final specification in sendSMS6 is a refinement of the original specification in sendSMS1. Note that it follows trivially from the definitions that all refinement relations are reflexive, i.e. that $[\![d]\!] \rightsquigarrow [\![d]\!]$ for any sequence diagram d.

5 Related work

Most closely related to the work presented in this paper is the STAIRS approach [18, 45], on which probabilistic STAIRS is based. STAIRS assigns formal semantics to sequence diagrams and defines refinement relations similar to the ones presented here. Time is introduced in STAIRS in [19]. An operational semantics for STAIRS is given in [34, 33], equivalent to the denotational one. However, STAIRS does not have the expressive power to capture requirements that depend on probabilities. The purpose of probabilistic STAIRS is to extend STAIRS in order to be able to capture also soft real-time requirements and other kinds of probabilistic requirements in the formal specifications.

We now review some of the other work that is related to the approach presented in this paper. We first look at work on sequence diagrams (or similar notations) in general, as little work has been done on sequence diagrams with probability. Then we look at other kinds of specification languages for expressing probabilistic requirements.

Message Sequence Charts (MSC) [24] are very similar to UML 2.x sequence diagrams, and have influenced the development of the latter. Important composition operators such as seq, par, alt, opt, and loop are included in both languages, with similar intuitive interpretations. However, the MSC language contains no operators for specifying negative behavior. Proposals for formal semantics for MSC have been given in, for example, [35] and [43]. While the latest version of the MSC specification [24] gives only an informal description of the semantics of MSC, an official formal operational semantics for an earlier version [22] is provided in [23]. With respect to refinement, [24] explains instance decomposition as a form of refinement, but gives no requirements on behavioral refinement. However, an amendment [25] to [24] discusses various ways of comparing MSC specifications or comparing an MSC specification with an implementation based on trace inclusion.

A semantics for MSCs with time is given in [3, 2]. The events of a MSC are assigned timestamps using a timing function, and timing constraints are used to specify minimum and maximum time intervals between events. In addition, algorithms are given for checking the realizability of MSCs and the existence of a timing function that is consistent with the timing constraints of an MSC. In [55], a semantics with time for MSC is given, based on partially ordered sets. Time is represented by a function mapping each event in a diagram to a set of time values, giving the absolute time interval in which the event should occur. Relative timing constraints are expressed by a function mapping pairs of events to intervals of time values.

Live Sequence Charts (LSC) [8, 16] is an extension of MSC that allow a distinction between possible and necessary behavior, as well as give explicit conditions under which the requirements of the diagram applies. For possible behavior, the interpretation is that at least one run of the system should satisfy the diagram. Thus, this is an existential requirement. For necessary (mandatory) behavior, the interpretation is that all system runs must satisfy the diagram. Thus, this is a universal requirement. The condition for the re-

quirements to apply is given by an initial part of the diagram called a *pre-chart*; the interpretation is that if the behavior specified by the pre-chart takes place, then the (existential or universal) requirements in the rest (the body) of the diagram apply. The condition given by the pre-chart is global in the sense that the requirements given by the body diagram do not apply until the full behavior of the pre-chart has taken place. In [17], a time extension to LSCs is presented. Here, a clock variable *Time* is added to the formalism so that time can be treated as data and time constraints can be expressed by means of ordinary variables. [16] includes a construct for probabilistic choice (which they call nondeterministic choice), where an exact probability is assigned to each alternative. Refinement is not formally defined, but [8] gives an example of refinement where a system is described in increasing level of detail, and states that refinement can easily be defined from the semantics of LSC.

In [15], Modal Sequence Diagrams (MSD) is proposed as an extension of UML 2.0 sequence diagrams, with a semantics based on LSC. The neg operator is interpreted as an addition of a universal and unfulfillable (false) condition to the end of the neg operand, thereby ensuring that the system is not allowed to satisfy scenarios that include the behavior of the neg operand. In MSD, the pre-charts are omitted in order to support sequential composition. Instead, existential fragments within a universal diagram serve a purpose similar to pre-charts. The existential fragment does not have to be satisfied, but if it is satisfied, then the subsequent universal fragment has to be satisfied.

Another LSC-inspired semantics for UML 2.x sequence diagrams is given in [31]. The semantics is based on partially ordered sets that are used to build event structures. Both negative, must and may behavior is expressed using modal logic constraints over these event structures. A similar operational semantics is given in [6]. Choices have guards, and if more than one guard is true the uppermost operand will be chosen. Each lifeline is executed separately meaning that synchronization must occur at the entry of choices in order to ensure that all lifelines choose the same operand. If a negative fragment is executed, the execution aborts in the same way as in LSC.

In [30], a variant of MSCs is defined that is supported by formal definitions of the semantics, as well as refinement relations. A system is represented semantically by a set of streams, each consisting of a sequence of system channel valuations and a sequence of state valuations. Nondeterminism is indicated by the existence of more than one stream. In addition, four different interpretations of MSCs are given, referred to as the existential, universal, exact and negative interpretation. The existential, universal and negative interpretations require the MSC in question to be fulfilled by at least one, all or none of the executions, respectively. The exact interpretation does also require the MSC to be fulfilled by all executions, but in addition all other behaviors than the ones explicitly specified by the MSC are prohibited. Four different refinement relations are defined in [30]. The first is binding of references, which allows references to empty MSCs (which are interpreted as arbitrary behavior) to be bound to 'proper' MSCs. Property refinement reduces the set of possible behaviors of the system, for example by removing alternatives or strengthening guards. Message refine-

ment allows a single message to be replaced by a whole interaction sequence or protocol. Finally, structural refinement allows a single component (lifeline) to be replaced by a set of other components, thus allowing decomposition.

In Triggered Message Sequence Charts (TMSC) [49, 50], an initial part of a diagram can be designated as a trigger diagram, with the interpretation that if the behavior described by the trigger diagram takes place, then the behavior described by the rest of the diagram must subsequently take place. Unlike the pre-charts of LSC, however, the trigger condition applies locally to each lifeline. This means that, for any given lifeline, if the events on that lifeline described by the trigger diagram take place, then the following events on that lifeline must subsequently take place. As the fulfillment of the trigger condition is determined locally on each lifeline, there is no need for synchronization between the lifelines. A refinement relation is defined, with the intuitive interpretation that a specification S_1 is refined by a specification S_2 if S_2 is more deterministic than S_1.

A semantics for UML 2.x sequence diagrams based on translating the diagrams to Büchi automata is provided in [14]. Positive and negative behavior is interpreted as liveness and safety properties, respectively. The diagrams are composed by strict rather than weak sequencing, and hence all lifelines as implicitly synchronized on entering and leaving of a sub-diagram. Refinement is defined to be the same as language inclusion.

In [51, 52, 53], a denotational trace based semantics for UML 2.x sequence diagrams is defined that is quite similar to the STAIRS semantics, but without underspecification. Also, sequence diagrams are not allowed to be inconsistent, and the `neg` operator for specifying negative traces can indirectly also specify positive traces. A notion of refinement is defined, but more restricted than in STAIRS as there is no notion of underspecification in the semantics.

A trace-based formal semantics for UML 2.0 sequence diagrams is proposed in [7], with many similarities to STAIRS and Störrle. A notable difference is that they make a prefix closure of negative traces. Similar to Störrle, inconsistent sequence diagrams are not allowed. A compliance (implementation) relation is defined that requires the system to produce at least one of the positive traces of the sequence diagram, and none of the negative traces. Refinement is defined in terms of compliance: A specification S is refined by a specification S' if any system that complies with S' also complies with S. [7] shows that not all composition operators (such as `seq`) are monotonic with respect to refinement, and mentions that the formal semantics and/or the refinement and compliance relations therefore may be subject to further adjustments. Based on the approach presented in [7], [5] develops a semantics for UML 2.0 Interactions with support for (hard) real-time constraints. A refinement relation for constraints is defined, and constraint introduction is shown to be monotonic with respect to refinement. Soft real-time constraints or other forms of probabilistic requirements are not considered.

Performance Message Sequence Chart (PMSC) [13, 32] extends MSC with syntactic constructs for expressing performance requirements. The aim is to integrate performance characteristics, such as response time and throughput, in

functional specifications. Of particular interest to us is the new operator altprob for probabilistic choice that is introduced in [32]. This operator allows exact probabilities to be assigned to the alternatives represented by its operands. This means that underspecification with respect to probability can not be captured by this operator. Apart from mentioning instance decomposition, refinement is not discussed, and no definition is given of what it means for a system to comply with a PMSC specification. The semantics of PMSC is explained at a purely intuitive level.

The UML Profile for Schedulability, Performance, and Time Specification [39] extends UML by adding stereotypes and annotations for defining values for performance measures, and allows specification of probability-related requirements, such as soft real-time requirements. The profile is intended to be used to construct models that can be analyzed with respect to performance, such as Markov chains. [1] presents a technique for constructing a Markov chain model from a sequence diagram that is annotated with constructs from the profile, thereby creating a model that can be analyzed by a suitable performance analysis tool.

We are not aware of any approach where probabilities are fully integrated in a formal semantics of sequence diagrams. However, there are a number of other languages where probabilities are fully integrated into the semantics. We now review some of these, in particular with respect to how they deal with refinement/verification and underspecification with respect to probability.

Two probabilistic variants of the process algebra CSP (Communicating Sequential Processes) [21], called $PCSP_0$ and $PCSP$, are presented in [48]. For both $PCSP_0$ and $PCSP$, an axiomatic characterization of the operators is offered, thus supporting algebraic reasoning about processes. In addition, a satisfaction relation is defined between a specification expressed as a predicate R over traces and a process P expressed in $PGCL$, as follows: A process P satisfies a specification R if $R(t)$ holds for every trace t of P. In both $PCSP_0$ and $PCSP$, the operator $\sqcap$ of CSP is replaced with an operator ${}_p\!\sqcap$ for probabilistic choice, where p is the probability of choosing the left-hand process and $1 - p$ is the probability of choosing the right-hand process. Underspecification with respect to probabilities cannot be expressed, as only exact probabilities can be assigned to the operator for probabilistic choice, and there is no operator for letting the system choose nondeterministically between two different probabilistic choices.

The language $pGCL$ [37, 38] is based on Dijkstra's Guarded Command Language (GCL) [10]. GCL and $pGCL$ can be seen as programming languages where states are represented by variable assignments. Both languages contains an operator $\sqcap$ for nondeterministic (demonic) choice[4]. This operator allows the notion of abstraction, and therefore, also refinement, to be captured. A nondeterministic choice between alternatives gives the union of the alternatives, and

[4]An operator $\sqcup$ for angelic choice is also introduced. Intuitively, in a $pGCL$ program, a demonic choice is made by a demon who seeks to minimize the probability of reaching the state under consideration, while an angelic choice is made by an angel who seeks to maximize the same probability.

refinement is defined as reverse inclusion: $prog'$ refines $prog$ if the behavior of $prog'$ is included in the behavior of $prog$. Hence, the nondeterminism expressed by $\sqcap$ represents underspecification.

In addition to the GCL operators, $pGCL$ contains an operator ${}_p\oplus$ for probabilistic choice. Underspecification with respect to probability can be expressed by a nondeterministic choice between two probabilistic choices whose alternatives are identical, where the probability values represent the upper and lower bounds on the acceptable probability. A nice feature of $pGCL$ is that it also offers a program logic that allows us to discover properties of the system via syntactic manipulations on the $pGCL$ program. For example, if S is a set of desirable states, then we may find the probability that execution of the program will end in a state in S.

In [20] it is shown how probabilistic reasoning can be applied to predicative programs and specifications. The semantics of a standard (non-probabilistic) predicative program is given in terms of first-order logic. For example, the program statement `if` b `then` $x := H$ `else` $x := T$ is interpreted as $(b \wedge x = H) \vee (\neg b \wedge x = T)$. This approach is generalized to the probabilistic case by considering Booleans to be a subset of real numbers, where $\top = 1$ and $\bot = 0$. A probabilistic choice can then be expressed with the `if`...`then`...`else`... construct. For example, an unfair coin biased toward the tails outcome can be represented by the program statement `if` 0.4 `then` $x := H$ `else` $x := T$. Nondeterminism is disjunction, and equivalent to an `if`...`then`...`else`... construct where the condition is a variable of unknown value (probability); $P \vee Q$ is equivalent to $\exists p \in [0,1] :$ `if` p `then` P `else` Q. Nondeterminism gives freedom to the implementer, who is intuitively free to choose p. A nondeterministic choice can be refined either by a probabilistic choice (by ensuring that $0 < p < 1$) or by a deterministic choice (by ensuring that $p = 1$ or $p = 0$). As in [37], underspecification with respect to probabilities can be expressed by a nondeterministic choice between two probabilistic choices.

Probabilistic automata [46, 47] are extensions of labeled transition systems designed to address the problem of modeling and verification of randomized distributed algorithms. Unlike ordinary automata, probabilistic automata allow probabilistic choice to be represented in the form of probabilistic transitions. A probabilistic transition is a transition from a state to a discrete distribution over pairs consisting of a label and a state. Nondeterminism is represented by the fact that there may be several outgoing (probabilistic) transitions from any state. Underspecification with respect to probabilities can be represented by nondeterministic choices between probabilistic transitions that are identical, except for the probability values of the distribution. [46] proposes hierarchical verification techniques based on either preorders of trace distributions (set inclusion) or on simulation.

A trace-based model for systems with both probabilistic and nondeterministic choice is presented in [9]. A trace is a sequence of states, and a state is an assignment of values to a set of variables. Semantically, a system is represented by a set of probability distributions on traces, which are called bundles. The fact that the model contains a set of bundles instead of a single bundle is due to

nondeterministic choices. As in, for example, [46], nondeterminism is resolved by a scheduler, so underspecification with respect to probabilities can be expressed in a similar way as in [46]. However, unlike [46] and other earlier work, [9] allows multiple schedulers for the resolution of the nondeterminism within a system. This is done in order to achieve deep compositionality, which means that the semantics (set of trace bundles) of a composite system can be obtained from the semantics of its component systems. For each scheduler, the set of variables it may affect and the set of variables that is visible to the scheduler (the variables upon which the scheduler's choice may depend) must be specified by a so-called *atom*. As the atoms determine probabilistic dependence between variable values, merging of atoms may increase the behavior (bundles) of a system. Atoms form a part of the semantic representation and are taken into consideration (by taking their union) when composing systems. Atoms also play a role with respect to refinement; refinement is basically bundle containment, with the additional requirement that the concrete system (implementation) cannot exhibit more variable dependencies than the abstract system (specification).

In [28], a specification formalism in the form of probabilistic (unlabeled) transition systems is presented. A probabilistic transition system is a transition system where transitions are assigned sets of allowed probabilities. The use of sets of allowed probabilities instead of exact probabilities represents underspecification with respect to probability, and is motivated partly by the need to specify soft requirements, such as 'the probability of losing a message in a communication channel should be no more than 0.01'. Two different refinement relations between specifications are proposed. The stronger criterion is based on the idea of simulation between specifications. The idea is that a transition in one specification can be simulated by a set of transitions in the other specification, as long as the combined probability of the transitions in this set is an acceptable probability of the original transition. The weaker criterion views a specification as a definition of a set of processes. Refinement is then defined as set inclusion of processes. A specification S is refined by a specification S' if all the processes of S' are also processes of S. A process (unlike a specification) has exactly one probability assigned to each transition.

In [27, 29], labeled transition systems with both nondeterministic and probabilistic choice are used for specifying systems. Nondeterministic choice is used to represent underspecification, and refinement corresponds to restricting the possible behavior. Refinement relations are defined based on testing. A test is a labeled transition system with both nondeterministic and probabilistic choice, where a subset of the states is defined as success states. A testing system $P \parallel T$ is the parallel composition of a process P and a test T, and from this we can obtain the set of possible probabilities of reaching a success state. Based on this, the concepts of may-refinement and must-refinement are defined as follows: A process P_2 is a may-refinement of a process P_1 if for every test T the highest probability of reaching a success state in $P \parallel T_2$ is not higher than in $P \parallel T_2$. P_2 is a must-refinement of P_1 if for every test T the lowest probability of reaching a success state in $P \parallel T_2$ is not lower than in $P \parallel T_2$. P_2 is a refinement of P_1 if it is both a may-refinement and a must-refinement of P_1. Intuitively, this means

that P_2 refines P_1 if for every test, the interval of probabilities of reaching a success state is made smaller. In other words, a refinement step may reduce underspecification with respect to probability. [29] shows that the refinement relations are compositional in the sense that if P_1 is refined by P_2, then $P_1 \parallel P$ is refined by $P_2 \parallel P$ for any process P.

6 Conclusion

In this paper, we have presented probabilistic STAIRS (pSTAIRS), an approach to extend UML 2.x sequence diagrams to capture soft real-time requirements as well as probabilistic choice in general. Soft real-time requirements are expressed by the combined use of operators for probabilistic choice (palt) and real-time constraints (tc). Having separate operators for real-time constraints and probabilistic choice allows us to capture all kinds of probabilistic choice in the same manner, whether related to time or not. In order to obtain a simpler notation for soft real-time requirements, one could easily introduce a macro operator for the combination of palt and tc used to capture such requirements. We have chosen not to introduce such a macro operator, as we wish to emphasize the similarity between soft real-time requirements and general probabilistic choice in the underlying theory. Probabilistic choice is used to capture requirements where there for each alternative is given a set of probabilities for how often the alternative may occur. Soft real-time requirements are a special case, where the difference between the alternatives is the timing requirements.

Probabilistic STAIRS makes it possible to capture underspecification with respect to probability as well as with respect to behavior/traces. Underspecification with respect to probability is essential in order to capture real-time requirements. It is also highly useful for other kinds of probabilistic choices, as the specifier will typically be satisfied as long as a given alternative occurs with a probability within a given interval, rather than with an exact probability. Moreover, achieving an exact probability in the final implementation can be very hard, meaning that specifications requiring exact probabilities may be almost impossible to comply with. Underspecification with respect to behavior/traces can be captured independently of underspecification with respect to probability. This enables refining a (sub-)specification with respect to behavior/traces without worrying about probabilities and vice versa.

Taking into account the incomplete nature of sequence diagrams, a formal semantics consistent with the semi-formal trace semantics of UML 2.x sequence diagrams has been provided for pSTAIRS. Based on this formal semantics, two alternative refinement relations targeting different phases of a development process have been presented. The first refinement relation ($\rightsquigarrow_{pg}$), suitable early in the development process, is used to reduce the amount of incompleteness and/or underspecification in the specification. At a later stage, general narrowing refinement ($\rightsquigarrow_{png}$) may be more suitable, assuming that all relevant behavior is specified so that the only relevant refinement step is reducing underspecification with respect to behavior/traces and/or probabilities. Both refinement

relations have been shown to have the mathematical properties of transitivity and monotonicity, which makes it possible to develop and analyze specifications in a stepwise and modular manner. The practical use of the refinement relations and the exploitation of their mathematical properties in order to simplify the analysis have been demonstrated on a scenario from the telecom industry.

References

[1] A. A. Abdulatif and R. Pooley. A computer assisted state marking method for extracting performance models from design models. *International Journal of Simulation Systems, Science and Technology*, pages 36–46, September 2007.

[2] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of message sequence charts. *IEEE Transactions on Software Engineering*, 29(7):623–633, 2003.

[3] Rajeev Alur, Gerard J. Holzmann, and Doron Peled. An analyser for message sequence charts. In *Proc. Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop (TACAS '96)*, volume 1055 of *LNCS*, pages 35–48. Springer, 1996.

[4] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: FOCUS on Streams, Interfaces, and Refinement.* Monographs in Computer Science. Springer, 2001.

[5] D. Calegari, M. V. Cengarle, and N. Szasz. UML 2.0 interactions with OCL/RT constraints. In *Proc. Forum on specification and Design Languages (FDL)*, pages 167–172, 2008.

[6] Alessandra Cavarra and Juliana Küster Filipe. Formalizing liveness-enriched sequence diagrams using ASMs. In *Proc. Abstract State Machines 2004. Advances in Theory and Practice, 11th International Workshop (ASM 2004)*, volume 3052 of *LNCS*, pages 62–77. Springer, 2004.

[7] M. V. Cengarle and A. Knapp. UML 2.0 interactions: Semantics and refinement. In *Proc. 3rd International Workshop on Critical Systems Development with UML (CSDUML)*, Technical report TUM-I0415, pages 85–99. Institut für Informatik, Technische Universität München, 2004.

[8] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[9] L. de Alfaro, T. A. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In *Proc. 12th International Conference on Concurrency Theory (CONCUR)*, pages 351–365. Springer, 2001.

[10] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[11] B. Dobing and J. Parsons. How UML is used. *Communications of the ACM*, 49(5):109–113, 2006.

[12] ETSI. *Open Service Access (OSA); Parlay X Web Services; Part 4: Short Messaging (Parlay X 2)*, 2006.

[13] N. Faltin, L. Lambert, A. Mitschele-Thiel, and F. Slomka. An annotational extension of message sequence charts to support performance engineering. In *SDL'97, Time For Testing*, SDL Forum, pages 307–322. Elsevier, 1997.

[14] Radu Grosu and Scott A. Smolka. Safety-liveness semantics for UML 2.0 sequence diagrams. In *Fifth International Conference on Application of Concurrency to System Design (ACSD 2005)*, pages 6–14. IEEE Computer Society, 2005.

[15] D. Harel and S. Maoz. Assert and negate revisited: Modal semantics for UML sequence diagrams. *Journal of Software and Systems Modeling*, 2007. Status: Online first.

[16] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.

[17] D. Harel and P. S. Thiagarajan. Message sequence charts. In *UML for Real: Design of Embedded Real-Time Systems*, pages 77–105. Kluwer Academic Publishers, 2003.

[18] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. STAIRS towards formal design with sequence diagrams. *Journal of Software and Systems Modeling*, 22(4):349–458, 2005.

[19] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. In *Proc. Scenarios: Models, Transformations and Tools*, volume 3466 of *LNCS*, pages 1–25. Springer, 2005.

[20] E. C. R. Hehner. Probabilistic predicative programming. In *Proc. 7th International Conference on Mathematics of Program Construction*, number 3125 in LNCS, pages 169–185. Springer, 2004.

[21] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[22] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1996.

[23] International Telecommunication Union. *Recommendation Z.120 — Annex B: Formal semantics of Message Sequence Charts*, 1998.

[24] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 2004.

[25] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC), Amendment 2: Revised Appendix I — Application of MSC*, 2009.

[26] C. B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall, 1986.

[27] B. Jonsson, C. Ho-Stuart, and W. Yi. Testing and refinement for nondeterministic and probabilistic processes. In *Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863, pages 418–430. Springer, 1994.

[28] B. Jonsson and K. G. Larsen. Specification and refinement of probabilistic processes. In *Proc. 6th Annual IEEE Symposium on Logic in Computer Science*, pages 266–277, 1991.

[29] B. Jonsson and W. Yi. Compositional testing preorders for probabilistic processes. In *Proc. 10th Annual IEEE Symposium on Logic in Computer Science*, pages 431–443. IEEE Computer Society, 1995.

[30] I. H. Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.

[31] Juliana Küster-Filipe. Modelling concurrent interactions. In *Proc. Algebraic Methodology and Software Technology, 10th International Conference (AMAST 2004)*, volume 3116 of *LNCS*, pages 304–318. Springer, 2004.

[32] L. Lambert. PMSC for performance evaluation. In *Proc. 1. Workshop on Performance and Time in SDL/MSC*, pages 70–80, 1998.

[33] M. S. Lund. *Operational Analysis of Sequence Diagram Specifications*. PhD thesis, University of Oslo, 2008.

[34] M. S. Lund and K. Stølen. A fully operational semantics for UML 2.0 sequence diagrams with potential and mandatory choice. In *Proc. 14th International Symposium on Formal Methods*, volume 4085 of *LNCS*, pages 380–395. Springer, 2006.

[35] S. Mauw and M. A. Reniers. An algebraic semantics of basic message sequence charts. *The Computer Journal*, 37(4):269–277, 1994.

[36] S. Mauw, M. A. Reniers, and T. A. C. Willemse. Message sequence charts in the software engineering process. In *Handbook of Software Engineering and Knowledge Engineering*, pages 437–463. World Scientific, 2001.

[37] A. McIver and C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005.

[38] A. McIver and C. Morgan. Developing and reasoning about probabilistic programs in pGCL. In *Proc. First Pernambuco Summer School on Software Engineering, Revised Lectures (PSSE)*, volume 3167 of *LNCS*, pages 123–155. Springer, 2006.

[39] Object Management Group. *UML Profile for Schedulability, Performance, and Time Specification (Version 1.1)*, document: formal/05-01-02 edition, 2005.

[40] Object Management Group. *UML 2.1 Superstructure Specification*, document: ptc/06-04-02 edition, 2006.

[41] 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, 2007.

[42] A. Refsdal, R. K. Runde, and K. Stølen. Relating computer systems to sequence diagrams with underspecification, inherent nondeterminism and probabilistic choice, Part 2. Technical Report 347, Department of Informatics, University of Oslo, 2007.

[43] M. A. Reniers. *Message Sequence Chart: Syntax and Semantics*. PhD thesis, Eindhoven University of Technology, 1999.

[44] R. K. Runde, Ø. Haugen, and K. Stølen. How to transform UML neg into a useful construct. In *Proc. Norsk Informatikkonferanse*, pages 55–66. Tapir, 2005.

[45] R. K. Runde, Ø. Haugen, and K. Stølen. Refining UML interactions with underspecification and nondeterminism. *Nordic Journal of Computing*, 12(2):157–188, 2005.

[46] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.

[47] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[48] K. Seidel. Probabilistic communicating processes. *Theoretical Computer Science*, 152(2):219–249, 1995.

[49] B. Sengupta and R. Cleaveland. Triggered message sequence charts. *SIGSOFT Software Engineering Notes*, 27(6):167–176, 2002.

[50] Bikram Sengupta and Rance Cleaveland. Triggered message sequence charts. *IEEE Transactions on Software Engineering*, 32(8):587–607, 2006.

[51] Harald Störrle. Assert, negate and refinement in UML-2 interactions. In *Proc. 2. International Workshop on Critical Systems Development with UML (CSDUML'03)*, Technical report TUM-I0317, pages 79–93. Institut für Informatik, Technische Universität München, 2003.

[52] Harald Störrle. Semantics of interactions in UML 2.0. In *IEEE Symposium on Human Centric Computer Languages and Environments (HCC 2003)*, pages 129–136. IEEE Computer Society, 2003.

[53] Harald Störrle. Trace semantics of interactions in UML 2.0. Technical Report TR 0403, Institut für Informatik, der Ludwig-Maximilians-Universität München, 2004.

[54] T. Weigert. *UML 2.0 RFI Response Overview*. Object Management Group, document: ad/00-01-07 edition, 1999.

[55] Tong Zheng, Ferhat Khendek, and Loïc Hélouët. A semantics for timed MSC. *Electronic Notes in Theoretical Computer Science*, 65(7), 2002.

A Formal semantics of composition operators

In this section we give a more detailed explanation of the semantic domain of specifications, and provide formal definitions of the remaining composition operators.

A.1 Events and traces

A *trace* is a sequence of *events* representing a system run. An event is a triple (k, m, t) consisting of a kind k, a message m, and a timestamp tag t. The kind k can be either !, denoting a transmission event, or ?, denoting a reception event. A message is a triple (s, tr, re) consisting of a signal s, a transmitter lifeline tr, and a receiver lifeline re. Every timestamp tag is assigned a timestamp, which is a positive real number, to indicate the time of occurrence for the event. Constraints on the timing of events are imposed by the use of logical formulas with timestamp tags as free variables.

We consider only closed sequence diagrams in the sense that for each send event in a trace, its corresponding receive is also present in the trace, and vice versa. In addition, 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;
- no event occurs more than once in the trace;
- if event e_1 occurs before event e_2 in the trace, then the timestamp assigned to the timestamp of e_2 is greater than or equal to the timestamp assigned to the timestamp of e_1.

Given a set of available lifeline names and signal names, we let $\mathcal{H}$ denote the set of all traces that are well-formed with respect to some sequence diagram. In

other words, a trace t is a member of $\mathcal{H}$ if and only if there exists a sequence diagram d such that t is well-formed with respect to d. We let $\mathcal{E}$ denote the set of all events.

A.2 Semantics of composition operators

In this section we define the semantics of the composition operators. But first we present the abstract/textual syntax of probabilistic sequence diagrams.

A.2.1 Abstract/textual syntax

The set of syntactically correct sequence diagrams, $\mathcal{D}$, is defined inductively as the least set such that[5]:

- $\mathcal{E} \subset \mathcal{D}$
- $\mathsf{skip} \in \mathcal{D}$
- $d \in \mathcal{D} \Rightarrow \mathsf{refuse}\ d \in \mathcal{D} \wedge \mathsf{veto}\ d \in \mathcal{D} \wedge \mathsf{opt}\ d \in \mathcal{D}$
- $d_1, d_2 \in \mathcal{D} \Rightarrow d_1\ \mathsf{par}\ d_2 \in \mathcal{D} \wedge d_1\ \mathsf{seq}\ d_2 \in \mathcal{D} \wedge d_1\ \mathsf{alt}\ d_2 \in \mathcal{D}$
- $d_1, \ldots, d_m \in \mathcal{D} \wedge Q_1, \ldots, Q_m \subseteq [0...1] \Rightarrow \mathsf{palt}(d_1;Q_1, \ldots, d_m;Q_m) \in \mathcal{D} \wedge \mathsf{expalt}(d_1;Q_1, \ldots, d_m;Q_m) \in \mathcal{D}$
- $d \in \mathcal{D} \wedge C \in \mathbb{F}(tt.d) \Rightarrow d\ \mathsf{tc}\ C \in \mathcal{D}$

where $\mathbb{F}(tt.d)$ denotes the set of logical formulas whose free variables are contained in the set $tt.d$ of all timestamp tags occurring in the diagram d.

The first two cases imply that any event, as well as the empty diagram, is a sequence diagram. Any other sequence diagram is constructed by the use of operators for negative behavior (refuse and veto), optional behavior (opt), parallel execution (par), weak sequencing (seq), potential choice/underspecification (alt), probabilistic choice (palt and expalt), or time constraint (tc). The semantics of these operators will be explained in A.2. Note that the seq operator occurs implicitly in the graphical diagrams.

A.2.2 Parallel composition, sequential composition, underspecification, negative behavior, and time constraints

We now define the operators that allow us to express parallel composition, sequential composition, underspecification with respect to behavior, negative behavior, and time constraints. But first we need to introduce some basic operators on traces, trace sets, interaction obligations, and p-obligations.

Parallel composition ($\|$) 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

[5]Note that we sometimes use infix notation also for palt and expalt when there is only two operands.

that for events on the same lifeline, all events from the first trace is ordered before the events from the second trace. Formally:

$$s_1 \parallel s_2 \quad \stackrel{\mathsf{def}}{=} \quad \{h \in \mathcal{H} \mid \exists p \in \{1,2\}^\infty : \pi_2((\{1\} \times \mathcal{E}) \circledT (p,h)) \in s_1 \\ \wedge \pi_2((\{2\} \times \mathcal{E}) \circledT (p,h)) \in s_2\} \tag{19}$$

$$s_1 \gtrsim s_2 \quad \stackrel{\mathsf{def}}{=} \quad \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \\ \forall l \in \mathcal{L} : e.l \circledS h = e.l \circledS h_1 \frown e.l \circledS h_2\} \tag{20}$$

where $\mathcal{L}$ denotes the set of all lifelines; $e.l$ is the set of events that may take place on the lifeline l, i.e. all transmission events where l is the transmitter and all receive events where l is the receiver; π_2 is a projection operator returning the second element of a pair; and $\frown$ is the concatenation operator for sequences. $\circledS$ and $\circledT$ are filtering operators for traces and pairs of traces, respectively. $E \circledS h$ 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

$$\{e_1, e_3\} \circledS \langle e_1, e_1, e_2, e_1, e_3, e_2 \rangle = \langle e_1, e_1, e_1, e_3 \rangle$$

The operator $\circledT$ is a generalization of $\circledS$ filtering pairs of traces with respect to pairs of elements such that, for instance

$$\{(1, e_1), (1, e_2)\} \circledT (\langle 1,1,2,1,2 \rangle, \langle e_1, e_1, e_1, e_2, e_2 \rangle) = (\langle 1,1,1 \rangle, \langle e_1, e_1, e_2 \rangle)$$

For formal definitions of $\circledS$ and $\circledT$, see [4].

Time requirements are imposed by the use of a time constraint, denoted by $\wr C$, where C is a predicate over timestamp tags. When a time constraint is applied to a trace set, all traces not fulfilling the constraint are removed. Formally, time constraint for a trace set s is defined as

$$s \wr C \quad \stackrel{\mathsf{def}}{=} \quad \{h \in s \mid h \models C\} \tag{21}$$

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

$$h = \langle (k_1, m_1, t_1 \mapsto r_1), (k_2, m_2, t_2 \mapsto r_2), (k_3, m_3, t_3 \mapsto r_3) \rangle \text{ and } C = t_3 < t_1 + 5$$

where $t_i \mapsto r_j$ denotes that timestamp r_j is assigned to timestamp tag t_i. Then $h \models C$ if $r_3 < r_1 + 5$.

For interaction obligations, parallel composition ($\parallel$), sequential composition

($\gtrsim$), inner union ($\uplus$), refusal ($\dagger$), and time constraint ($\wr$) are defined by:

$$(p_1, n_1) \parallel (p_2, n_2) \stackrel{\mathsf{def}}{=} (p_1 \parallel p_2, (n_1 \parallel p_2) \cup (n_1 \parallel n_2) \cup (p_1 \parallel n_2)) \quad (22)$$

$$(p_1, n_1) \gtrsim (p_2, n_2) \stackrel{\mathsf{def}}{=} (p_1 \gtrsim p_2, \quad (23)$$
$$(n_1 \gtrsim p_2) \cup (n_1 \gtrsim n_2) \cup (p_1 \gtrsim n_2))$$

$$(p_1, n_1) \uplus (p_2, n_2) \stackrel{\mathsf{def}}{=} (p_1 \cup p_2, n_1 \cup n_2) \quad (24)$$

$$\dagger(p_1, n_1) \stackrel{\mathsf{def}}{=} (\emptyset, p_1 \cup n_1) \quad (25)$$

$$(p, n) \wr C \stackrel{\mathsf{def}}{=} (p \wr C, n \cup (p \wr \neg C)) \quad (26)$$

Notice that for $\parallel$ and $\gtrsim$, 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.

Inner union $\uplus$ represents underspecification with respect to behavior/traces. The idea is that two different interaction obligations represent behavior that from the specifier's point of view are equivalent with respect to their positive and negative traces. Hence the interaction obligations can be combined into a single interaction obligation, thus allowing us to introduce new positive or negative traces in an interaction obligation.

Finally, time constraint $\wr$ defines traces that do not fulfill the constraint as negative, while traces that fulfill the constraint are positive.

Definitions (22) to (26) for interaction obligations are extended to p-obligations as follows:

$$(o_1, Q_1) \parallel (o_2, Q_2) \stackrel{\mathsf{def}}{=} (o_1 \parallel o_2, Q_1 * Q_2) \quad (27)$$

$$(o_1, Q_1) \gtrsim (o_2, Q_2) \stackrel{\mathsf{def}}{=} (o_1 \gtrsim o_2, Q_1 * Q_2) \quad (28)$$

$$(o_1, Q_1) \uplus (o_2, Q_2) \stackrel{\mathsf{def}}{=} (o_1 \uplus o_2, Q_1 * Q_2) \quad (29)$$

$$\dagger(o, Q) \stackrel{\mathsf{def}}{=} (\dagger o, Q) \quad (30)$$

$$(o, Q) \wr C \stackrel{\mathsf{def}}{=} (o \wr C, Q) \quad (31)$$

where we write o_i for an interaction obligation (p_i, n_i). The multiplication of probability sets when composing two p-obligation with $\parallel$, $\gtrsim$ or $\uplus$ is motivated by the fact that such compositions always occur in the context of composing specifications represented by sets of p-obligations, where each p-obligation in the resulting composed specification is obtained by choosing independently one p-obligation from each set. In other words, the composition of two sets of p-obligations is the set we may obtain by choosing one p-obligation from each set and composing these two p-obligations. Thus, the definitions of parallel composition ($\|$), sequential composition ($\gtrsim$), inner union ($\uplus$), refusal ($\dagger$), and time constraint ($\wr$) are lifted from p-obligations to sets of p-obligations in a

straightforward manner:

$$O_1 \; op \; O_2 \quad \stackrel{\text{def}}{=} \quad \{po_1 \; op \; po_2 \mid po_1 \in O_1 \wedge po_2 \in O_2\} \tag{32}$$

$$\dagger O \quad \stackrel{\text{def}}{=} \quad \{\dagger po \mid po \in O\} \tag{33}$$

$$O \wr C \quad \stackrel{\text{def}}{=} \quad \{po \wr C \mid po \in O\} \tag{34}$$

where op is one of $\parallel$, $\gtrsim$ and $\uplus$.

We are now ready to define the semantics of the pSTAIRS operators. The semantics of an event $(k, m, t) \in \mathcal{E}$ is the interaction obligation whose positive set consists of infinitely many unary positive traces – one for each possible assignment of a timestamp to the timestamp tag of the event. The negative set is empty, and 1 is the only allowed probability:

$$[\![\; (k, m, t) \;]\!] \quad \stackrel{\text{def}}{=} \quad \{((\{\langle (k, m, t \mapsto r) \rangle \mid r \in \mathbb{R}\}, \emptyset), \{1\})\} \tag{35}$$

The empty diagram denotes an empty trace:

$$[\![\; \mathsf{skip} \;]\!] \quad \stackrel{\text{def}}{=} \quad \{((\{\langle\rangle\}, \emptyset), \{1\})\} \tag{36}$$

The operators seq, par, alt and refuse are defined as follows:

$$[\![\; d_1 \; \mathsf{seq} \; d_2 \;]\!] \quad \stackrel{\text{def}}{=} \quad [\![\; d_1 \;]\!] \gtrsim [\![\; d_2 \;]\!] \tag{37}$$

$$[\![\; d_1 \; \mathsf{par} \; d_2 \;]\!] \quad \stackrel{\text{def}}{=} \quad [\![\; d_1 \;]\!] \parallel [\![\; d_2 \;]\!] \tag{38}$$

$$[\![\; d_1 \; \mathsf{alt} \; d_2 \;]\!] \quad \stackrel{\text{def}}{=} \quad [\![\; d_1 \;]\!] \uplus [\![\; d_2 \;]\!] \tag{39}$$

$$[\![\; \mathsf{refuse} \; d \;]\!] \quad \stackrel{\text{def}}{=} \quad \dagger [\![\; d \;]\!] \tag{40}$$

$$[\![\; d \; \mathsf{tc} \; C \;]\!] \quad \stackrel{\text{def}}{=} \quad [\![\; d \;]\!] \wr C \tag{41}$$

The macro operators veto and opt are defined by:

$$\mathsf{veto} \; d \quad \stackrel{\text{def}}{=} \quad \mathsf{skip} \; \mathsf{alt} \; \mathsf{refuse} \; d \tag{42}$$

$$\mathsf{opt} \; d \quad \stackrel{\text{def}}{=} \quad \mathsf{skip} \; \mathsf{alt} \; d \tag{43}$$

Notice that the two operators refuse and veto are used instead of the UML operator neg. The reason for this is ambiguity in the UML standard, as further explained in [44].

A.2.3 Example

We now illustrate the definitions of weak sequencing and time constraints with an example. Consider the diagram example in Figure 16, where we have chosen to show the timestamp tags for all events explicitly. The semantics of this specification contains only a single p-obligation, as the palt operator must be employed to introduce more p-obligations (this is explained later). Hence, we get $[\![\; \mathsf{example} \;]\!] = \{((p, n), \{1\})\}$, where p is the set of traces that is described

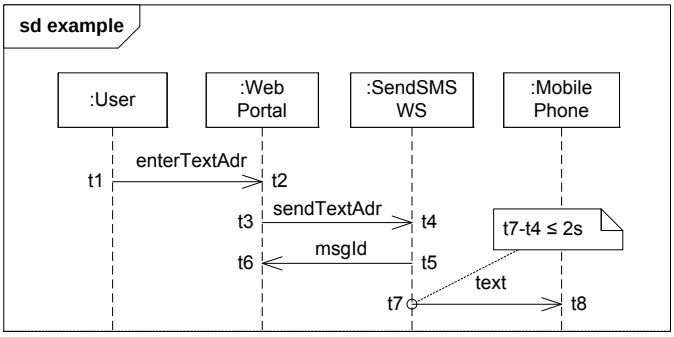

Figure 16: Example scenario for illustration of semantics.

as positive by example and n is the set of traces that is described as negative by example. We now explain which traces are contained in p and n.

The traces described by the diagram follows the restrictions on the ordering of events imposed by weak sequencing. The first event to occur can only be transmission of enterTextAdr on the :User lifeline, as the first event on all other lifelines are reception events that cannot occur before their corresponding transmission events. For the same reason, the second event can only be reception of enterTextAdr on the :WebPortal lifeline, followed by transmission of sendTextAdr on the same lifeline and reception of sendTextAdr on the :SendSMSWS lifeline. Again, there is only one event that can occur next at this point, namely transmission of msgId on the :SendSMSWS lifeline. However, after transmission of this message there are two possibilities for the next event: either the msgId message is received by the web portal or the text message is sent from the web service. As these events occur on different lifelines (and transmission of msgId has already occurred), weak sequencing does not impose a particular order between these two events. In the case where the transmission event of text occurs before the reception event of msgId, there are again two alternatives for the next event: either reception of msgId occurs before reception of text, or vice versa. This means that there are three alternative orderings that differ with respect to the following: 1) Reception of msgId occurs before transmission of text. 2) Transmission of text occurs before reception of msgId, and reception of msgId occurs before reception of text. 3) Transmission of text occurs before reception of msgId, and reception of text occurs before reception of msgId.

Before showing what the resulting traces look like we introduce the following shorthand notation for messages:

$$\begin{aligned} et &= (\mathsf{enterTextAdr}, \mathsf{User}, \mathsf{WebPortal}) \\ st &= (\mathsf{sendTextAdr}, \mathsf{WebPortal}, \mathsf{SendSMSWS}) \\ mi &= (\mathsf{msgId}, \mathsf{SendSMSWS}, \mathsf{WebPortal}) \\ te &= (\mathsf{text}, \mathsf{SendSMSWS}, \mathsf{MobilePhone}). \end{aligned}$$

Ignoring for the moment the difference between positive and negative traces, the set s of all traces described by example equals the union $s = s_1 \cup s_2 \cup s_3$ of the trace sets we get by ordering the events according to the alternatives described above. The trace sets s_1, s_2, s_3 are then given by the following:

$$\begin{aligned}
s_1 &= \{\langle (!, et, t_1 \mapsto r_1), (?, et, t_2 \mapsto r_2), (!, st, t_3 \mapsto r_3), (?, st, t_4 \mapsto r_4), \\
&\quad (!, mi, t_5 \mapsto r_5), (?, mi, t_6 \mapsto r_6), (!, te, t_7 \mapsto r_7), (?, te, t_8 \mapsto r_8)\rangle \\
&\quad \mid \forall i < 8 : r_i \leq r_{i+1}\} \\
s_2 &= \{\langle (!, et, t_1 \mapsto r_1), (?, et, t_2 \mapsto r_2), (!, st, t_3 \mapsto r_3), (?, st, t_4 \mapsto r_4), \\
&\quad (!, mi, t_5 \mapsto r_5), (!, te, t_7 \mapsto r_7), (?, mi, t_6 \mapsto r_6), (?, te, t_8 \mapsto r_8)\rangle \\
&\quad \mid \forall i < 8 : r_i \leq r_{i+1}\} \\
s_3 &= \{\langle (!, et, t_1 \mapsto r_1), (?, et, t_2 \mapsto r_2), (!, st, t_3 \mapsto r_3), (?, st, t_4 \mapsto r_4), \\
&\quad (!, mi, t_5 \mapsto r_5), (!, te, t_7 \mapsto r_7), (?, te, t_8 \mapsto r_8), (?, mi, t_6 \mapsto r_6)\rangle \\
&\quad \mid \forall i < 8 : r_i \leq r_{i+1}\}
\end{aligned}$$

Note that s_1, s_2, and s_3 are infinite sets due to the fact that there are infinitely many ways of assigning timestamps to the timestamp tags.

The positive traces according to example are the traces of s that also fulfills the time constraint attached to the transmission event of text, while the negative traces are those that do not fulfill this constraint. Hence, we have

$$\begin{aligned}
p &= \{h \in s \mid r_7 - r_4 \leq 2\} \\
n &= \{h \in s \mid \neg(r_7 - r_4 \leq 2)\}
\end{aligned}$$

Table 1: Overview over monotonicity proofs

	$\rightsquigarrow_{pg}$	$\rightsquigarrow_{png}$
refuse	T21 in [42]	T38 in [42]
seq	T3 in [41]	T39 in [42]
par	T4 in [41]	T40 in [42]
alt	T5 in [41]	T41 in [42]
palt	T6 in [41]	T8
tc	T7 in [41]	T7

Table 2: Overview over transitivity proofs

	$\rightsquigarrow_{pg}$	$\rightsquigarrow_{png}$
Transitivity	T1 in [41]	T14 in [42]

B Formal proofs

We now present proofs for the properties presented in Section 4.3. Many of the proofs are found in other technical reports ([41] and [42]), while other proofs are new for this report. Table 1 gives an overview over all monotonicity proofs, while Table 2 gives an overview over transitivity proofs. We use T as shorthand for Theorem, so that "T21 in [42]" means Theorem 21 in the technical report [42]. Entries containing only a theorem number, without reference to another report, refer to theorems in this report, which are found below.

The operators for real-time constraints tc and $\wr$ were not included in [42]. As the results presented in Section 4.3 rely partly on proofs from [42], we now explain how to extend the proofs in [42] to ensure that the results presented in Section 4.3 are valid.

As real-time was not included in [42], there was no timestamp tag or timestamp associated with events. The introduction of timestamps means that a single event in a diagram is represented semantically by an infinite number of events, one for each possible assignment of a timestamp to its timestamp tag, rather than by a single event. This semantical extension has no implication for the proofs on which the results in Section 4.3 rely, as these proofs do not make any assumptions about the cardinality of trace sets.

However, the proofs of Lemma 2 and Lemma 11 in [42] use induction on the syntactical structure of the diagram, which means that the addition of the operator tc needs to be taken into account. We therefore now present revised proofs for these lemmas. The following convention for notation of p-obligations has been used: $po_i = (o_i, Q_i) = ((p_i, n_i), Q_i)$.

Lemma 4 (Update of Lemma 2 in [42]) *Let $d \in \mathcal{D}$. Then*

$$\pi_2.\bar{\oplus}[\![\ d\ ]\!] \subseteq \{1\}$$

PROOF.

$\langle 1\rangle 1.\ \exists po \in [\![\ d\ ]\!] : Q \subseteq \{1\}$
  $\langle 2\rangle 1.$ CASE: d consists of a single event e or $d = \mathsf{skip}$
    $\langle 3\rangle 1.\ [\![\ d\ ]\!] = \{((\{\langle (k, m, t \mapsto r)\rangle \mid r \in \mathbb{R}\}, \emptyset), \{1\})\} \vee [\![\ d\ ]\!] = \{((\{\langle\rangle\}, \emptyset), \{1\})\}$
      PROOF: By assumption $\langle 2\rangle 1$
    $\langle 3\rangle 2.$ Q.E.D.
      PROOF: By $\langle 3\rangle 1$
  $\langle 2\rangle 2.$ CASE: d contains at least one operator
    $\langle 3\rangle 1.$ ASSUME: For every sequence diagram d' that occur in an operand of d the following holds:
      $\exists po' \in [\![\ d'\ ]\!] : Q' \subseteq \{1\}$ (ind. hyp.)
      PROVE: $\exists po \in [\![\ d\ ]\!] : Q \subseteq \{1\}$
      $\langle 4\rangle 1.$ CASE: $d = \mathsf{palt}(d_1;Q_1, \ldots d_n;Q_n)$
        $\langle 5\rangle 1.\ \exists po \in [\![\ d\ ]\!] : \pi_2.po = \{1\} \cap \sum_{i=1}^{n} Q_i$
          PROOF: By $\langle 4\rangle 1$ and definition (8)
        $\langle 5\rangle 2.$ Q.E.D.
          PROOF: By $\langle 5\rangle 1$ and definition 7
      $\langle 4\rangle 2.$ CASE: $d = d_1\ \mathsf{seq}\ d_2$
        $\langle 5\rangle 1.$ LET: $po_1 \in [\![\ d_1\ ]\!]$ s.t. $Q_1 \subseteq \{1\}$
          $po_2 \in [\![\ d_2\ ]\!]$ s.t. $Q_2 \subseteq \{1\}$
          PROOF: By assumption $\langle 3\rangle 1$
        $\langle 5\rangle 2.\ po_1 \gtrsim po_2 \in [\![\ d\ ]\!]$
          PROOF: By $\langle 5\rangle 1$ and assumption $\langle 4\rangle 2$
        $\langle 5\rangle 3.\ \pi_2.(po_1 \gtrsim po_2) \subseteq \{1\}$
          PROOF: By $\langle 5\rangle 1$
        $\langle 5\rangle 4.$ Q.E.D.
          PROOF: By $\langle 5\rangle 2$ and $\langle 5\rangle 3$
      $\langle 4\rangle 3.$ CASE: $d = d_1\ \mathsf{par}\ d_2$
        PROOF: Similar to $\langle 4\rangle 2$
      $\langle 4\rangle 4.$ CASE: $d = d_1\ \mathsf{alt}\ d_2$
        PROOF: Similar to $\langle 4\rangle 2$
      $\langle 4\rangle 5.$ CASE: $d = \mathsf{refuse}\ d_1$
        $\langle 5\rangle 1.$ LET: $(o, Q) \in [\![\ d_1\ ]\!]$ s.t. $Q \subseteq \{1\}$
          PROOF: By assumption $\langle 3\rangle 1$
        $\langle 5\rangle 2.\ (\dagger o, Q) \in [\![\ d\ ]\!]$
          PROOF: By $\langle 5\rangle 1$ and assumption $\langle 4\rangle 5$
        $\langle 5\rangle 3.$ Q.E.D.
          PROOF: By $\langle 5\rangle 1$ ($Q \subseteq \{1\}$) and $\langle 5\rangle 2$
      $\langle 4\rangle 6.$ CASE: $d = d_1 \mathsf{tc}\ C$
        $\langle 5\rangle 1.$ LET: $(o, Q) \in [\![\ d_1\ ]\!]$ s.t. $Q \subseteq \{1\}$
          PROOF: By assumption $\langle 3\rangle 1$
        $\langle 5\rangle 2.\ (o \wr C, Q) \in [\![\ d\ ]\!]$

PROOF: By $\langle 5\rangle 1$ and assumption $\langle 4\rangle 6$

$\langle 5\rangle 3$. Q.E.D.

PROOF: By $\langle 5\rangle 1$ ($Q \subseteq \{1\}$) and $\langle 5\rangle 2$

$\langle 4\rangle 7$. Q.E.D.

PROOF: The cases $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$ and $\langle 4\rangle 6$ are exhaustive (the remaining operators veto, opt and expalt are macro operators)

$\langle 3\rangle 2$. Q.E.D.

PROOF: Induction step

$\langle 2\rangle 3$. Q.E.D.

PROOF: Induction with $\langle 2\rangle 1$ as base case and $\langle 2\rangle 2$ as induction step

$\langle 1\rangle 2$. Q.E.D.

PROOF: By $\langle 1\rangle 1$, Definition (6) and Definition (7)

□

The following lemma is useful for showing Lemma 6, which is an update of Lemma 11 in [42]

Lemma 5 *Let O be a set of p-obligations. Then*

$$\bar{\oplus}(O \wr C) = (\bar{\oplus} O) \wr C$$

PROOF.

$\langle 1\rangle 1$. LET: $po_a = \bar{\oplus}(O \wr C)$
$po_b = (\bar{\oplus} O) \wr C$

$\langle 1\rangle 2$. $Q_a = Q_b$

PROOF: By $\langle 1\rangle 1$, Definition (6) and Definition (34) ($\wr C$ does not affect probability sets)

$\langle 1\rangle 3$. $p_a = \bigcup_{po \in O \wr C} p \cap \bigcap_{po \in O \wr C} p \cup n \wedge n_a = \bigcap_{po \in O \wr C} n$

PROOF: By $\langle 1\rangle 1$ and Definition (6)

$\langle 1\rangle 4$. $p_b = (\bigcup_{po \in O} p \cap \bigcap_{po \in O} p \cup n) \wr C \wedge n_b = \bigcap_{po \in O} n \cup ((\bigcup_{po \in O} p \cap \bigcap_{po \in O} p \cup n) \wr \neg C)$

PROOF: By $\langle 1\rangle 1$ and Definition (6)

$\langle 1\rangle 5$. $p_a = p_b$

$\langle 2\rangle 1$. $\bigcup_{po \in O \wr C} p = \bigcup_{po \in O} p \wr C$

PROOF: By Definition (34)

$\langle 2\rangle 2$. $\forall t \in \bigcup_{po \in O} p \wr C : t \models C$

PROOF: By Definition (21)

$\langle 2\rangle 3$. $\forall t \in \bigcup_{po \in O \wr C} p : t \models C$

PROOF: By $\langle 2\rangle 2$ and $\langle 2\rangle 1$

$\langle 2\rangle 4$. $\forall t \in \bigcup_{po \in O \wr C} p \cap \bigcap_{po \in O \wr C} p \cup n : t \models C$

PROOF: By $\langle 2\rangle 3$

$\langle 2\rangle 5$. $\bigcup_{po \in O \wr C} p \cap \bigcap_{po \in O \wr C} p \cup n = (\bigcup_{po \in O} p \cap \bigcap_{po \in O} p \cup n) \wr C$

PROOF: By $\langle 2\rangle 4$

$\langle 2\rangle 6$. Q.E.D.

PROOF: By $\langle 2\rangle 5$ $\langle 1\rangle 3$ and $\langle 1\rangle 4$

$\langle 1\rangle 6$. $n_a = n_b$

$\langle 2\rangle 1$. $n_a \subseteq n_b$

⟨3⟩1. ASSUME: $t \in n_a$
PROVE: $t \in n_b$
⟨4⟩1. $\forall po \in O \wr C : t \in n$
PROOF: By assumption ⟨3⟩1 and ⟨1⟩3
⟨4⟩2. $\forall po \in O : t \in n \cup (p \wr \neg C)$
PROOF: By assumption ⟨4⟩1 and Definition (34)
⟨4⟩3. CASE: $\forall po \in O : t \in n$
⟨5⟩1. $t \in \bigcup_{po \in O} n$
PROOF: By assumption ⟨4⟩3
⟨5⟩2. Q.E.D.
PROOF: By ⟨5⟩1
⟨4⟩4. CASE: $\exists po \in O : t \notin n$
⟨5⟩1. LET: $po' \in O$ such that $t \notin n'$
PROOF: By assumption ⟨4⟩4
⟨5⟩2. $t \in p' \wr \neg C$
PROOF: By ⟨5⟩1 and ⟨4⟩2
⟨5⟩3. $t \in (\bigcup_{po \in O} p) \wr \neg C$
PROOF: By ⟨5⟩1 and ⟨5⟩2
⟨5⟩4. $\neg(t \models C)$
PROOF: By ⟨5⟩2
⟨5⟩5. $t \in (\bigcap_{po \in O} p \cup n) \wr \neg C$
PROOF: By ⟨4⟩2 and ⟨5⟩4
⟨5⟩6. $t \in (\bigcup_{po \in O} p \cap \bigcap_{po \in O} p \cup n) \wr \neg C$
PROOF: By ⟨5⟩3 and ⟨5⟩5
⟨5⟩7. Q.E.D.
PROOF: By ⟨5⟩6 and ⟨1⟩4
⟨4⟩5. Q.E.D.
PROOF: The cases ⟨4⟩3 and ⟨4⟩4 are exhaustive
⟨3⟩2. Q.E.D.
PROOF: $\subseteq$-rule: ⟨3⟩1
⟨2⟩2. $n_b \subseteq n_a$
⟨3⟩1. ASSUME: $t \in n_b$
PROVE: $t \in n_a$
⟨4⟩1. $\forall po \in O \wr C : t \in n$
⟨5⟩1. ASSUME: $po_1 \in O \wr C$
PROVE: $t \in n_1$
⟨6⟩1. LET: $po_2 \in O$ such that $po_1 = po_2 \wr C$
PROOF: By assumption ⟨5⟩1
⟨6⟩2. $n_1 = n_2 \cup (p_2 \wr C)$
PROOF: By ⟨6⟩1 and Definition (31)
⟨6⟩3. CASE: $t \in \bigcap_{po \in O} n$
⟨7⟩1. $t \in n_2$
PROOF: By ⟨6⟩1 and assumption ⟨6⟩3
⟨7⟩2. Q.E.D.
PROOF: By ⟨7⟩1 and ⟨6⟩2
⟨6⟩4. CASE: $t \in (\bigcup_{po \in O} p \cap \bigcap_{po \in O} p \cup n) \wr \neg C$

⟨7⟩1. $t \in p_2 \cup n_2$
PROOF: By ⟨6⟩1 and assumption ⟨6⟩4
⟨7⟩2. $t \models \neg C$
PROOF: By assumption ⟨6⟩4 and Definition (21)
⟨7⟩3. $t \in n_2 \cup (p_2 \wr \neg C)$
PROOF: By ⟨7⟩1 and ⟨7⟩2
⟨7⟩4. Q.E.D.
PROOF: By ⟨7⟩3 and ⟨6⟩2
⟨6⟩5. Q.E.D.
PROOF: By ⟨1⟩4 (right conjunct) and assumption ⟨3⟩1 the cases ⟨6⟩3 and ⟨6⟩4 are exhaustive
⟨5⟩2. Q.E.D.
PROOF: ∀-rule: ⟨5⟩1
⟨4⟩2. Q.E.D.
PROOF: By ⟨4⟩1 and ⟨1⟩3 (right conjunct)
⟨3⟩2. Q.E.D.
PROOF: ⊆-rule: ⟨3⟩1
⟨2⟩3. Q.E.D.
PROOF: By ⟨2⟩1 and ⟨2⟩2
⟨1⟩7. Q.E.D.
PROOF: By ⟨1⟩5 and ⟨1⟩6

□

Lemma 6 (Update of Lemma 11 in [42]) *Let $d \in \mathcal{D}$. Then*

$$\exists po \in [\![d]\!] : po \rightsquigarrow_{pr} \bar{\oplus}[\![d]\!] \wedge Q \subseteq \{1\}$$

PROOF.

⟨1⟩1. CASE: d consists of a single event e or $d = \mathsf{skip}$
⟨2⟩1. $[\![d]\!] = \{((\{\langle (k, m, t \mapsto r)\rangle \mid r \in \mathbb{R}\}, \emptyset), \{1\})\} = \{\bar{\oplus}[\![d]\!]\} \vee$
$[\![d]\!] = \{((\{\langle\rangle\}, \emptyset), \{1\})\} = \{\bar{\oplus}[\![d]\!]\}$
PROOF: By assumption ⟨1⟩1
⟨2⟩2. Q.E.D.
PROOF: By ⟨2⟩1 and reflexivity of $\rightsquigarrow_{pr}$
⟨1⟩2. CASE: d contains at least one operator
⟨2⟩1. ASSUME: For every sequence diagram d' that occur in an operand of an operator in d the following holds:
$\exists po' \in [\![d']\!] : po' \rightsquigarrow_{pr} \bar{\oplus}[\![d']\!] \wedge Q' \subseteq \{1\}$ (ind. hyp.).
PROVE: $\exists po \in [\![d]\!] : po \rightsquigarrow_{pr} \bar{\oplus}[\![d]\!] \wedge Q \subseteq \{1\}$
⟨3⟩1. CASE: $d = \mathsf{palt}(d_1;Q_1, \ldots, d_n;Q_n)$
⟨4⟩1. LET: $po_a = (\oplus \bigcup_{i=1}^{n} [\![d_i;Q_i]\!], \{1\} \cap \sum_{i=1}^{n} Q_i)$
⟨4⟩2. $po_a \in [\![d]\!]$
PROOF: By assumption ⟨3⟩1 and definition (8)
⟨4⟩3. $Q_a \subseteq \{1\}$
PROOF: By ⟨4⟩1
⟨4⟩4. $po_a \rightsquigarrow_{pr} \bar{\oplus}[\![d]\!]$

⟨5⟩1. $\pi_2.\bar{\oplus}[\![\ d\ ]\!] \subseteq Q_a$
⟨6⟩1. $\pi_2.\bar{\oplus}[\![\ d\ ]\!] \subseteq \{1\}$
PROOF: By Lemma 4
⟨6⟩2. CASE: $\pi_2.\bar{\oplus}[\![\ d\ ]\!] = \{1\}$
⟨7⟩1. $1 \in \sum_{i=1}^{n} Q_i$
⟨8⟩1. ASSUME: $1 \notin \sum_{i=1}^{n} Q_i$
PROVE: $\bot$
⟨9⟩1. $Q_a = \emptyset$
PROOF: By ⟨4⟩1 and assumption ⟨8⟩1
⟨9⟩2. $\exists po \in [\![\ d\ ]\!] : Q = \emptyset$
PROOF: By ⟨9⟩1 and ⟨4⟩2
⟨9⟩3. $\pi_2.\bar{\oplus}[\![\ d\ ]\!] = \emptyset$
PROOF: By ⟨9⟩2
⟨9⟩4. Q.E.D.
PROOF: By ⟨9⟩3 and assumption ⟨6⟩2
⟨8⟩2. Q.E.D.
PROOF: $\bot$-rule
⟨7⟩2. $1 \in Q_a$
PROOF: By ⟨7⟩1 and ⟨4⟩1
⟨7⟩3. Q.E.D.
PROOF: By ⟨7⟩2 and assumption ⟨6⟩2
⟨6⟩3. CASE: $\pi_2.\bar{\oplus}[\![\ d\ ]\!] = \emptyset$
PROOF: By assumption ⟨6⟩3
⟨6⟩4. Q.E.D.
PROOF: By ⟨6⟩1, the cases ⟨6⟩2 and ⟨6⟩3 are exhaustive
⟨5⟩2. $o_a \rightsquigarrow_r \oplus[\![\ d\ ]\!]$
⟨6⟩1. $\oplus[\![\ d\ ]\!] = \oplus \bigcup_{i=1}^{n} [\![\ d_i;Q_i\ ]\!]$
PROOF: By assumption ⟨3⟩1
⟨6⟩2. $o_a = \oplus \bigcup_{i=1}^{n} [\![\ d_i;Q_i\ ]\!]$
PROOF: By ⟨4⟩1
⟨6⟩3. Q.E.D.
PROOF: By ⟨6⟩1 and ⟨6⟩2
⟨5⟩3. Q.E.D.
PROOF: By ⟨5⟩1 and ⟨5⟩2
⟨4⟩5. Q.E.D.
PROOF: By ⟨4⟩2, ⟨4⟩3 and ⟨4⟩4; po_a is the po we are looking for.
⟨3⟩2. CASE: $d = d_1$ seq d_2
⟨4⟩1. LET: $po_1 \in [\![\ d_1\ ]\!]$ such that $po_1 \rightsquigarrow_{pr} \bar{\oplus}[\![\ d_1\ ]\!] \wedge Q_1 \subseteq \{1\}$
$po_2 \in [\![\ d_2\ ]\!]$ such that $po_2 \rightsquigarrow_{pr} \bar{\oplus}[\![\ d_2\ ]\!] \wedge Q_2 \subseteq \{1\}$
PROOF: By assumption ⟨2⟩1
⟨4⟩2. $po_1 \succsim po_2 \in [\![\ d\ ]\!]$
PROOF: By ⟨4⟩1 and assumption ⟨3⟩2
⟨4⟩3. $\pi_2.(po_1 \succsim po_2) \subseteq \{1\}$
PROOF: By ⟨4⟩1
⟨4⟩4. $po_1 \succsim po_2 \rightsquigarrow_{pr} \bar{\oplus}[\![\ d_1\ ]\!] \succsim \bar{\oplus}[\![\ d_2\ ]\!]$
PROOF: By ⟨4⟩1 and Lemma 3 in [42]

⟨4⟩5. $\bar{\oplus}[\![\ d_1\ ]\!] \succsim \bar{\oplus}[\![\ d_2\ ]\!] \rightsquigarrow_{pr} \bar{\oplus}([\![\ d_1\ ]\!] \succsim [\![\ d_2\ ]\!])$
⟨5⟩1. $\oplus[\![\ d_1\ ]\!] \succsim \oplus[\![\ d_2\ ]\!] \rightsquigarrow_{r} \oplus([\![\ d_1\ ]\!] \succsim [\![\ d_2\ ]\!])$
PROOF: By Lemma 4 in [42]
⟨5⟩2. $\pi_2.\bar{\oplus}([\![\ d_1\ ]\!] \succsim [\![\ d_2\ ]\!]) \subseteq \pi_2.(\bar{\oplus}[\![\ d_1\ ]\!] \succsim \bar{\oplus}[\![\ d_2\ ]\!])$
PROOF: By Lemma 7 in [42]
⟨5⟩3. Q.E.D.
PROOF: By ⟨5⟩1 and ⟨5⟩2
⟨4⟩6. $po_1 \succsim po_2 \rightsquigarrow_{pr} \bar{\oplus}([\![\ d_1\ ]\!] \succsim [\![\ d_2\ ]\!])$
PROOF: By ⟨4⟩4, ⟨4⟩5 and transitivity of $\rightsquigarrow_{pr}$
⟨4⟩7. Q.E.D.
PROOF: By ⟨4⟩2, ⟨4⟩3 and ⟨4⟩6; $po_1 \succsim po_2$ is the *po* we are looking for.
⟨3⟩3. CASE: $d = d_1$ par d_2
PROOF: Similar to case ⟨3⟩2, with $\succsim$ replaced by $\|$, and the reference to Lemma 4 in [42] replaced by a reference to Lemma 5 in [42].
⟨3⟩4. CASE: $d = d_1$alt d_2
⟨4⟩1. LET: $po_1 \in [\![\ d_1\ ]\!]$ such that $po_1 \rightsquigarrow_{pr} \bar{\oplus}[\![\ d_1\ ]\!] \wedge Q_1 \subseteq \{1\}$
$po_2 \in [\![\ d_2\ ]\!]$ such that $po_2 \rightsquigarrow_{pr} \bar{\oplus}[\![\ d_2\ ]\!] \wedge Q_1 \subseteq \{1\}$
PROOF: By assumption ⟨2⟩1
⟨4⟩2. $po_1 \uplus po_2 \in [\![\ d\ ]\!]$
PROOF: By ⟨4⟩1 and assumption ⟨3⟩4
⟨4⟩3. $\pi_2.(po_1 \uplus po_2) \subseteq \{1\}$
PROOF: By ⟨4⟩1
⟨4⟩4. $po_1 \uplus po_2 \rightsquigarrow_{pr} \bar{\oplus}[\![\ d_1\ ]\!] \uplus \bar{\oplus}[\![\ d_2\ ]\!]$
PROOF: By ⟨4⟩1 and Lemma 3 in [42]
⟨4⟩5. $\bar{\oplus}[\![\ d_1\ ]\!] \uplus \bar{\oplus}[\![\ d_2\ ]\!] \rightsquigarrow_{pr} \bar{\oplus}([\![\ d_1\ ]\!] \uplus [\![\ d_2\ ]\!])$
⟨5⟩1. $\oplus[\![\ d_1\ ]\!] \uplus \oplus[\![\ d_2\ ]\!] \rightsquigarrow_{r} \oplus([\![\ d_1\ ]\!] \uplus [\![\ d_2\ ]\!])$
PROOF: By Lemma 6 in [42]
⟨5⟩2. $\pi_2.\bar{\oplus}([\![\ d_1\ ]\!] \uplus [\![\ d_2\ ]\!]) \subseteq \pi_2.(\bar{\oplus}[\![\ d_1\ ]\!] \uplus \bar{\oplus}[\![\ d_2\ ]\!])$
PROOF: By Lemma 7 in [42]
⟨5⟩3. Q.E.D.
PROOF: By ⟨5⟩1 and ⟨5⟩2
⟨4⟩6. $po_1 \uplus po_2 \rightsquigarrow_{pr} \bar{\oplus}([\![\ d_1\ ]\!] \uplus [\![\ d_2\ ]\!])$
PROOF: By ⟨4⟩4, ⟨4⟩5 and transitivity of $\rightsquigarrow_{pr}$
⟨4⟩7. Q.E.D.
PROOF: By ⟨4⟩2, ⟨4⟩3 and ⟨4⟩6; $po_1 \uplus po_2$ is the *po* we are looking for.
⟨3⟩5. CASE: $d =$ refuse d_1
⟨4⟩1. LET: $po_1 \in [\![\ d_1\ ]\!]$ such that $po_1 \rightsquigarrow_{pr} \bar{\oplus}[\![\ d_1\ ]\!] \wedge Q_1 \subseteq \{1\}$
PROOF: By assumption ⟨2⟩1
⟨4⟩2. $\dagger po_1 \in [\![\ d\ ]\!]$
PROOF: By ⟨4⟩1 and assumption ⟨3⟩5
⟨4⟩3. $\pi_2.(\dagger po_1) \subseteq \{1\}$
PROOF: By ⟨4⟩1
⟨4⟩4. $\dagger po_1 \rightsquigarrow_{pr} \dagger\bar{\oplus}[\![\ d_1\ ]\!]$
PROOF: By ⟨4⟩1 and Lemma 9 in [42]
⟨4⟩5. $\dagger po_1 \rightsquigarrow_{pr} \bar{\oplus}\dagger[\![\ d_1\ ]\!]$
PROOF: By ⟨4⟩4 and Lemma 8 in [42]

⟨4⟩6. Q.E.D.
PROOF: By ⟨4⟩2, ⟨4⟩3 and ⟨4⟩5; $\dagger po_1$ is the po we are looking for.
⟨3⟩6. CASE: $d = d_1 \mathsf{tc}\ C$
⟨4⟩1. LET: $po_1 \in [\![\ d_1\ ]\!]$ such that $po_1 \rightsquigarrow_{pr} \bar{\oplus}[\![\ d_1\ ]\!] \wedge Q_1 \subseteq \{1\}$
PROOF: By assumption ⟨2⟩1
⟨4⟩2. $po_1 \wr C \in [\![\ d\ ]\!]$
PROOF: By ⟨4⟩1 and assumption ⟨3⟩6
⟨4⟩3. $\pi_2.(po_1 \wr C) \subseteq \{1\}$
PROOF: By ⟨4⟩1
⟨4⟩4. $po_1 \wr C \rightsquigarrow_{pr} (\bar{\oplus}[\![\ d_1\ ]\!]) \wr C$
PROOF: By ⟨4⟩1 and Lemma 8 in [41], which states that $po \rightsquigarrow_{pr} po' \Rightarrow po \wr C \rightsquigarrow_{pr} po' \wr C$)
⟨4⟩5. $po_1 \wr C \rightsquigarrow_{pr} \bar{\oplus}([\![\ d_1\ ]\!] \wr C)$
PROOF: By ⟨4⟩4 and Lemma 5
⟨4⟩6. Q.E.D.
PROOF: By ⟨4⟩2, ⟨4⟩3 and ⟨4⟩5; $\dagger po_1$ is the po we are looking for.
⟨3⟩7. Q.E.D.
PROOF: The cases ⟨3⟩1, ⟨3⟩2, ⟨3⟩3, ⟨3⟩4, ⟨3⟩5 and ⟨3⟩6 are exhaustive.
⟨2⟩2. Q.E.D.
PROOF: Induction step
⟨1⟩3. Q.E.D.
PROOF: By induction with ⟨1⟩1 as basis and ⟨1⟩2 as induction step

□

Time was not considered in [42] and the refinement relation $\rightsquigarrow_{png}$ was not introduced in [41], therefore there was no proof of monotonicity of tc with respect to $\rightsquigarrow_{png}$ in either report. We therefore now present this proof.

Theorem 7 (Monotonicity of tc w.r.t. $\rightsquigarrow_{png}$) *Let $d, d' \in \mathcal{D}$. Then*

$$[\![\ d\ ]\!] \rightsquigarrow_{png} [\![\ d'\ ]\!] \Rightarrow [\![\ d\ \mathsf{tc}\ C\ ]\!] \rightsquigarrow_{png} [\![\ d'\ \mathsf{tc}\ C\ ]\!]$$

PROOF.

⟨1⟩1. ASSUME: $[\![\ d\ ]\!] \rightsquigarrow_{png} [\![\ d'\ ]\!]$
PROVE: $[\![\ d\ \mathsf{tc}\ C\ ]\!] \rightsquigarrow_{png} [\![\ d'\ \mathsf{tc}\ C\ ]\!]$
⟨2⟩1. $\forall po \in [\![\ d\ ]\!] \wr C : 0 \notin \pi_2.po \Rightarrow \exists po' \in [\![\ d'\ ]\!] \wr C : po \rightsquigarrow_{pnr} po'$
⟨3⟩1. ASSUME: $po_1 \in [\![\ d\ ]\!] \wr C$
PROVE: $0 \notin \pi_2.po_1 \Rightarrow \exists po' \in [\![\ d'\ ]\!] \wr C : po_1 \rightsquigarrow_{pnr} po'$
⟨4⟩1. ASSUME: $0 \notin \pi_2.po_1$
PROVE: $\exists po' \in [\![\ d'\ ]\!] \wr C : po_1 \rightsquigarrow_{pnr} po'$
⟨5⟩1. LET: $po'_1 \in [\![\ d\ ]\!]$ such that $po_1 = po'_1 \wr C$
PROOF: By assumption ⟨3⟩1
⟨5⟩2. LET: $po'_2 \in [\![\ d'\ ]\!]$ such that $po'_1 \rightsquigarrow_{pnr} po'_2$
PROOF: By ⟨5⟩1, assumption ⟨1⟩1 and assumption ⟨4⟩1
⟨5⟩3. $po'_1 \wr C \rightsquigarrow_{pr} po'_2 \wr C$
PROOF: By ⟨5⟩2 and Lemma 8 in [41]
⟨5⟩4. $p'_1 \wr C \cup n'_1 \cup p'_1 \wr \neg C = p'_2 \wr C \cup n'_2 \cup p'_2 \wr \neg C$

⟨6⟩1. $p'_1 \cup n'_1 = p'_2 \cup n'_2$
PROOF: By ⟨5⟩2
⟨6⟩2. Q.E.D.
PROOF: By ⟨6⟩1
⟨5⟩5. $po'_1 \wr C \rightsquigarrow_{pnr} po'_2 \wr C$
PROOF: By ⟨5⟩3 and ⟨5⟩4
⟨5⟩6. $po'_2 \wr C \in [\![d']\!] \wr C$
PROOF: By ⟨5⟩2
⟨5⟩7. Q.E.D.
PROOF: By ⟨5⟩5 and ⟨5⟩6; $po'_2 \wr C$ is the po' we are looking for
⟨4⟩2. Q.E.D.
PROOF: ⇒-rule: ⟨4⟩1
⟨3⟩2. Q.E.D.
PROOF: ∀-rule: ⟨3⟩1
⟨2⟩2. Q.E.D.
PROOF: By ⟨2⟩1
⟨1⟩2. Q.E.D.
PROOF: ⇒-rule: ⟨1⟩1

□

Finally, neither [41] nor [42] included a monotonicity proof for palt w.r.t. $\rightsquigarrow_{png}$, therefore we include this proof here.

Theorem 8 (Monotonicity of palt w.r.t $\rightsquigarrow_{png}$) *Let $d_1, \ldots, d_n, d'_1, \ldots, d'_n \in \mathcal{D}$. Furthermore, let $d = \mathsf{palt}(d_1;Q_1, \ldots, d_n;Q_n)$ and $d' = \mathsf{palt}(d'_1;Q'_1, \ldots, d'_n;Q'_n)$. Then*

$$\forall i \leq n : [\![d_i]\!] \rightsquigarrow_{png} [\![d'_i]\!] \wedge Q'_i \subseteq Q_i \wedge \oplus [\![d_i]\!] \rightsquigarrow_{nr} \oplus [\![d'_i]\!] \Rightarrow [\![d]\!] \rightsquigarrow_{png} [\![d']\!]$$

PROOF.

⟨1⟩1. ASSUME: 1. $\forall i \leq n : [\![d_i]\!] \rightsquigarrow_{png} [\![d'_i]\!]$
2. $\forall i \leq n : Q'_i \subseteq Q_i$
3. $\forall i \leq n : \oplus [\![d_i]\!] \rightsquigarrow_{nr} \oplus [\![d'_i]\!]$
PROVE: $[\![d]\!] \rightsquigarrow_{png} [\![d']\!]$
⟨2⟩1. $\forall po \in [\![d]\!] : 0 \notin \pi_2.po \Rightarrow \exists po' \in [\![d']\!] : po \rightsquigarrow_{pnr} po'$
⟨3⟩1. ASSUME: $po_a \in [\![d]\!]$
PROVE: $0 \notin \pi_2.po_a \Rightarrow \exists po' \in [\![d']\!] : po_a \rightsquigarrow_{pnr} po'$
⟨4⟩1. ASSUME: $0 \notin \pi_2.po_a$
PROVE: $\exists po' \in [\![d']\!] : po_a \rightsquigarrow_{pnr} po'$
⟨5⟩1. CASE: $po_a \in \{\bar{\oplus}(\{po_i\}_{i \in N}) \mid N \subseteq \{1, \ldots, n\} \wedge N \neq \emptyset \wedge \forall i \in N : po_i \in [\![d_i;Q_i]\!]\}$
⟨6⟩1. LET: $N \subseteq \{1, \ldots, n\}, po_i \in [\![d_i;Q_i]\!]$ for each $i \leq n$ such that $N \neq \emptyset \wedge po_a = \bar{\oplus}(\{po_i\}_{i \in N})$
PROOF: By assumption ⟨5⟩1
⟨6⟩2. LET: $po'_i \in [\![d'_i;Q'_i]\!]$ such that $po_i \rightsquigarrow_{pnr} po'_i$ for all $i \in N$
PROOF: By assumption ⟨1⟩1,1 and 2
⟨6⟩3. $\bar{\oplus}(\{po_i\}_{i \in N}) \rightsquigarrow_{pnr} \bar{\oplus}(\{po'_i\}_{i \in N})$

⟨7⟩1. $\forall i \in N : \oplus\{po_i\} \rightsquigarrow_{nr} \oplus\{po'_i\}$
PROOF: By ⟨6⟩2 and Definition (4)
⟨7⟩2. $\oplus(\{po_i\}_{i\in N}) \rightsquigarrow_{nr} \oplus(\{po'_i\}_{i\in N})$
PROOF: By ⟨7⟩1 and Lemma 20 in [42]
⟨7⟩3. $\sum_{i\in N} \pi_2.po'_i \subseteq \sum_{i\in N} \pi_2.po_i$
PROOF: By ⟨6⟩2
⟨7⟩4. Q.E.D.
PROOF: By ⟨7⟩2 and ⟨7⟩3
⟨6⟩4. $\bar{\oplus}(\{po'_i\}_{i\in N}) \in [\![\ d'\ ]\!]$
PROOF: By ⟨6⟩2 and Definition (8)
⟨6⟩5. Q.E.D.
PROOF: By ⟨6⟩3 and ⟨6⟩4; $\bar{\oplus}(\{po'_i\}_{i\in N})$ is the po' we are looking for.
⟨5⟩2. CASE: $po_a = (\oplus\bigcup_{i=1}^{n}[\![\ d_i;Q_i\ ]\!], \{1\} \cap \sum_{i=1}^{n} Q_i)$
⟨6⟩1. $\{1\} \cap \sum_{i=1}^{n} Q'_i \subseteq \{1\} \cap \sum_{i=1}^{n} Q_i$
PROOF: By assumption ⟨1⟩1,2
⟨6⟩2. $\oplus\bigcup_{i=1}^{n}[\![\ d_i;Q_i\ ]\!] \rightsquigarrow_{nr} \oplus\bigcup_{i=1}^{n}[\![\ d'_i;Q'_i\ ]\!]$
PROOF: By assumption ⟨1⟩1,3 and Lemma 20 in [42]
⟨6⟩3. $po_a \rightsquigarrow_{pnr} (\oplus\bigcup_{i=1}^{n}[\![\ d'_i;Q'_i\ ]\!], \{1\} \cap \sum_{i=1}^{n} Q'_i)$
PROOF: By assumption ⟨5⟩2, ⟨6⟩1 and ⟨6⟩2
⟨6⟩4. $(\oplus\bigcup_{i=1}^{n}[\![\ d'_i;Q'_i\ ]\!], \{1\} \cap \sum_{i=1}^{n} Q'_i) \in [\![\ d'\ ]\!]$
PROOF: By definition (8)
⟨6⟩5. Q.E.D.
PROOF: By ⟨6⟩3 and ⟨6⟩4; $(\oplus\bigcup_{i=1}^{n}[\![\ d'_i;Q'_i\ ]\!], \{1\} \cap \sum_{i=1}^{n} Q'_i)$ is the po' we are looking for
⟨5⟩3. Q.E.D.
PROOF: By Definition (8) the cases ⟨5⟩1 and ⟨5⟩2 are exhaustive
⟨4⟩2. Q.E.D.
PROOF: ⇒-rule
⟨3⟩2. Q.E.D.
PROOF: ∀-rule
⟨2⟩2. Q.E.D.
PROOF: By ⟨2⟩1
⟨1⟩2. Q.E.D.
PROOF: ⇒-rule

□

We now prove the modularity result (Theorem 2 from Section 4.3.1), which shows that the extra requirement for monotonicity of refinement w.r.t. palt (as represented by 3 above, and similarly for $\rightsquigarrow_r$) does not have significant practical consequences as explained in Section 4.3.1. Note that Theorem 2 from Section 4.3.1 follows immediately from Theorem 9 and Theorem 10 below.

Before we present the proofs we need to define formally what it means for a specification to be safe. A *chain* is an infinite sequence of traces c such that $\forall j \in \mathbb{N} : c[j] \sqsubseteq c[j+1]$, where $c[j]$ denotes the j'th element of c and $\sqsubseteq$ is the standard prefix operator on traces. Any chain has a least upper bound denoted

by $\sqcup c$. We let *chains* denote the set of all chains. A specification d is *safe* iff the following holds for any chain c:

$$(\forall j \in \mathbb{N} : \exists t \in \mathcal{H} \setminus \pi_2. \oplus [\![d]\!] : c[j] \sqsubseteq t \wedge \#\sqcup c = \infty) \Rightarrow \sqcup c \in \mathcal{H} \setminus \pi_2. \oplus [\![d]\!] \quad (44)$$

where $\# \sqcup c$ denotes the length of $\sqcup c$.

Theorem 9 (Modularity of palt w.r.t. $(\rightsquigarrow_{pg}, \mapsto_{pg})$) *Let*

$$d = \mathsf{palt}(d_1;Q_1, \ldots, d_k;Q_k) \quad (45)$$
$$d' = \mathsf{palt}(d'_1;Q'_1, \ldots, d'_k;Q'_k) \quad (46)$$

Assume that

$$\forall j \leq k : [\![d_j]\!] \rightsquigarrow_{pg} [\![d'_j]\!] \wedge Q'_j \subseteq Q_j \quad (47)$$
$$\forall j \leq k : d_j \text{ is safe} \quad (48)$$
$$d' \text{ is well-balanced} \quad (49)$$
$$d' \mapsto_{pg} I \wedge \forall j \leq k : d'_j \mapsto I_j \quad (50)$$

where the implementation I is composed of the implementations I_j, i.e. $I = op(I_1, \ldots, I_k)$ for some suitable composition operator op so that the traces produced by I is the union of the traces produced by each I_j. Then the following holds:

$$d \mapsto_{pg} I \quad (51)$$

Proof If $[\![d]\!] \rightsquigarrow_{pg} [\![d']\!]$ then (51) follows immediately from the first conjunct of assumption 50 together with preservement of $\mapsto_{pg}$ through abstraction. In the following we therefore assume

$$[\![d]\!] \not\rightsquigarrow_{pg} [\![d']\!] \quad (52)$$

This means that there exists a p-obligation $((p_1, n_1), Q_1) \in [\![d]\!]$ such that

$$\forall ((p', n'), Q') \in [\![d']\!] : ((p_1, n_1), Q_1) \not\rightsquigarrow_{pr} ((p', n'), Q') \quad (53)$$

As $d = \mathsf{palt}(d_1;Q_1, \ldots, d_k;Q_k)$, it is clear that the p-obligation $((p_1, n_1), Q_1)$ comes either from line (a) or line (b) of Definition (8). Assume it comes from line (a). This means that $((p_1, n_1), Q_1) = \bar{\oplus} M$, where M is a set of p-obligations obtained by selecting at most one p-obligation po_j from each operand $d_j;Q_j$ of the palt. Then we can obtain a set of p-obligations M' by selecting corresponding p-obligations po'_j from the corresponding operands of $[\![d']\!]$ such that $\forall j \leq k : po_j \rightsquigarrow_{pr} po'_j$. But this means that $\bar{\oplus} M' \in [\![d']\!]$ and $((p_1, n_1), Q_1) \rightsquigarrow_{pr} \bar{\oplus} M'$, which contradicts (53). Hence, $((p_1, n_1), Q_1)$ must come from line (b) of Definition (8), which means that

$$((p_1, n_1), Q_1) = (\oplus \bigcup_{j=1}^{k} [\![d_j;Q_j]\!], \{1\} \cap \sum_{j=1}^{k} Q_j) \quad (54)$$

Since p-obligations with an empty probability set are not implementable, it follows from the first conjunct of assumption (50) that

$$\forall((p,n),Q) \in [\![\ d'\ ]\!] : Q \neq \emptyset \tag{55}$$

From this together with assumption (47) it then follows that

$$\forall((p,n),Q) \in [\![\ d\ ]\!] : Q \neq \emptyset \tag{56}$$

From this, (54) and the fact that $\oplus \bigcup_{j=1}^{k} [\![\ d_j;Q_j\ ]\!] = \oplus [\![\ d\ ]\!]$ it then follows that

$$((p_1,n_1),Q_1) = (\oplus [\![\ d\ ]\!], \{1\}) \tag{57}$$

From (55) and Definition (8) it follows that

$$(\oplus [\![\ d'\ ]\!], \{1\}) \in [\![\ d'\ ]\!] \tag{58}$$

Together with (53), this implies that

$$\oplus [\![\ d\ ]\!] \not\rightsquigarrow_r \oplus [\![\ d'\ ]\!] \tag{59}$$

This means that we have either $\pi_2.\oplus [\![\ d\ ]\!] \not\subseteq \pi_2.\oplus [\![\ d'\ ]\!]$ or $\pi_1.\oplus [\![\ d\ ]\!] \not\subseteq \pi_1.\oplus [\![\ d'\ ]\!] \cup \pi_2.\oplus [\![\ d'\ ]\!]$. From assumptions (47) (first conjunct) and (49) it follows that the latter alternative is not possible, as this would mean that there exists a trace that is included in all p-obligations in $[\![\ d\ ]\!]$, but not in all p-obligations in $[\![\ d'\ ]\!]$. As d' is well-balanced, this would imply that this trace is not included in any p-obligation in $[\![\ d'\ ]\!]$, which would contradict the first conjunct of assumption (47). Hence, the first alternative must hold, i.e. $\pi_2.\oplus [\![\ d\ ]\!] \not\subseteq \pi_2.\oplus [\![\ d'\ ]\!]$. This means that there exists a trace t such that

$$t \in \pi_2.\oplus [\![\ d\ ]\!] \wedge t \in \pi_1.\oplus [\![\ d'\ ]\!] \tag{60}$$

This means that there exists an $i \leq k$ such that

$$t \in \pi_2.\oplus [\![\ d_i\ ]\!] \wedge t \notin \pi_2.\oplus [\![\ d'_i\ ]\!] \tag{61}$$

From (56) it follows that there exists a p-obligation $((p,n),\{1\})$ such that

$$((p,n),\{1\}) \in [\![\ d_i\ ]\!] \tag{62}$$

Together, (62), (55) and (47) imply that there exists a p-obligation $((p',n'),\{1\})$ such that

$$((p',n'),\{1\}) \in [\![\ d'_i\ ]\!] \wedge (p,n) \rightsquigarrow_r (p',n') \tag{63}$$

From the first conjunct of (61) together with (62) we get

$$t \in n \tag{64}$$

Together, (64) and (63) imply that

$$t \in n' \tag{65}$$

From (63) and (65) we have that t is negative in a p-obligation in $[\![\ d_i'\ ]\!]$ with 1 as the only allowed probability. This means that in any implementation of d_i', t can only be produced with probability 0. Now assume for contradicion that I_i is able to produce t. Since any trace in the semantics of a probabilistic STAIRS specification by definition is either infinite or represents an execution that terminates, and no execution that terminates can occur with probability 0 it follows that

$$\#t = \infty \tag{66}$$

From the first conjunt of (61) it follows that

$$t \notin \mathcal{H} \setminus \pi_2.\oplus [\![\ d_i\ ]\!] \tag{67}$$

From (48) it follows that d_i is safe. Together with (66), (67) and definition (44), this means that there exists $m \in \mathbb{N}$ such that

$$\forall t' \in \mathcal{H} \setminus \pi_2.\oplus [\![\ d_i\ ]\!] : t|_m \not\sqsubseteq t' \tag{68}$$

To see this, assume that (68) did not hold, i.e. that $\forall j \in \mathbb{N} : \exists t' \in \mathcal{H} \setminus \pi_2.\oplus [\![\ d_i\ ]\!] : t|_j \sqsubseteq t'$. As t is the least upper bound for the chain defined by $\forall j \in \mathbb{N} : c[j] = t|_j$, (48) would then imply that $t \in \mathcal{H} \setminus \pi_2.\oplus [\![\ d_i\ ]\!]$, which contradicts (67).

Let S be the set of all traces with $t|_m$ as a prefix, i.e. $S = \{t' \in \mathcal{H} \mid t|_m \sqsubseteq t'\}$. Note that since $t|_m$ is finite it follows that either I_i does not produce any traces in S at all, or the probability that I_i will produce a trace in S is greater than 0. From (68) it follows that

$$(\mathcal{H} \setminus \pi_2.\oplus [\![\ d_i\ ]\!]) \cap S = \emptyset \tag{69}$$

From (62) it follows that

$$\pi_2.\oplus [\![\ d_i\ ]\!] \subseteq n \tag{70}$$

Together with the second conjunct of (63), (70) implies that

$$\pi_2.\oplus [\![\ d_i\ ]\!] \subseteq n' \tag{71}$$

From (69) we get

$$S \subseteq \pi_2.\oplus [\![\ d_i\ ]\!] \tag{72}$$

Together, (72) and (71) imply that

$$S \subseteq n' \tag{73}$$

From the definition of S it follows that $t \in S$, which means that I_i is able to produce a trace in S, and hence that the probability that I_i produces a trace in S is greater than 0. But together with (73) this means that the probability

of producing n' is greater than 0, which contradicts the first conjunct of (63). Hence, the assumption that I_i is able to produce t cannot hold.

From the left conjunct of (60) it follows that $t \in \pi_2.\oplus[\![\ d_j\ ]\!]$ for any $j \leq k$. Hence, what we have shown from (59) and onwards is essentially that for any trace t that breaks the condition for $\oplus[\![\ d\ ]\!] \rightsquigarrow_r \oplus[\![\ d\ ]\!]$ to hold there is no $j \leq k$ such that t is produced by I_j, even in the cases where $t \notin \pi_2.\oplus[\![\ d'_j\ ]\!]$. Hence, t is not produced by I. But this means that if I implements $(\oplus[\![\ d'\ ]\!], \{1\})$, it also implements $(\oplus[\![\ d\ ]\!], \{1\})$. Since it follows from (58) and the first conjunct of (50) that I implements $(\oplus[\![\ d'\ ]\!], \{1\})$, it then follows that I implements $(\oplus[\![\ d\ ]\!], \{1\})$. As this is the only p-obligation in $[\![\ d\ ]\!]$ not refined by any p-obligation in $[\![\ d'\ ]\!]$ (which is clear from the fact that (57) could be derived from (53)), this means that (51) must also hold. □

Theorem 10 (Modularity of palt w.r.t. $(\rightsquigarrow_{png}, \mapsto_{png})$) *Let*

$$d = \mathsf{palt}(d_1;Q_1,\ldots,d_k;Q_k) \tag{74}$$
$$d' = \mathsf{palt}(d'_1;Q'_1,\ldots,d'_k;Q'_k) \tag{75}$$

Assume that

$$\forall j \leq k : [\![\ d_j\ ]\!] \rightsquigarrow_{png} [\![\ d'_j\ ]\!] \wedge Q'_j \subseteq Q_j \tag{76}$$
$$\forall j \leq k : d_j \textit{ is safe} \tag{77}$$
$$d' \textit{ is well-balanced} \tag{78}$$
$$d' \mapsto_{png} I \wedge \forall j \leq k : d'_j \mapsto_{png} I_j \tag{79}$$

where the implementation I is composed of the implementations I_j, i.e. $I = op(I_1,\ldots,I_k)$ for some suitable composition operator op so that the traces produced by I is the union of the traces produced by each I_j. Then the following holds:

$$d \mapsto_{png} I \tag{80}$$

Proof The proof is similar to the proof for Theorem 9, with $\rightsquigarrow_{pg}$ everywhere replaced by $\rightsquigarrow_{png}$, $\rightsquigarrow_r$ everywhere replaced by $\rightsquigarrow_{nr}$ and $\mapsto_{pg}$ everywhere replaced by $\mapsto_{png}$. Apart from this, the only difference is that there is an extra alternative $\pi_1.\oplus[\![\ d\ ]\!] \cup \pi_2.\oplus[\![\ d\ ]\!] \neq \pi_1.\oplus[\![\ d'\ ]\!] \cup \pi_2.\oplus[\![\ d'\ ]\!]$ that must be considered after step (59). However, from the assumptions that d' is well-balanced and that $\forall j \leq k : [\![\ d_j\ ]\!] \rightsquigarrow_{png} [\![\ d_j\ ]\!]$, it is clear that $\pi_1.\oplus[\![\ d\ ]\!] \cup \pi_2.\oplus[\![\ d\ ]\!] \neq \pi_1.\oplus[\![\ d'\ ]\!] \cup \pi_2.\oplus[\![\ d'\ ]\!]$ cannot hold. We may therefore continue the proof from step (60) in the same manner as for Theorem 9. □

Technology for a better society

www.sintef.no