ELSEVIER

Science of Computer Programming 26 (1996) 255–272

Science of Computer Programming

Refinement principles supporting the transition from asynchronous to synchronous communication

Ketil Stølen *

Institut für Informatik, TU München, D-80290 München, Germany

Abstract

We present three refinement principles supporting the transition from system specifications based on (unbounded) asynchronous communication to system specifications based on (bounded) synchronous communication. We refer to these principles as partial, total and conditional refinement, respectively. We distinguish between two synchronization techniques, namely synchronization by hand-shake and synchronization by real-time constraints. Partial refinement supports synchronization by hand-shake with respect to safety properties. Total refinement supports synchronization by hand-shake with respect to both safety and liveness properties. Finally, conditional refinement supports both synchronization by hand-shake and by real-time constraints. We discuss, relate and show the use of these principles in a number of small examples.

1. Introduction

Any method for system development, which depends on that boundedness constraints – constraints imposing upper bounds on the memory available for some data structure, component or channel – are imposed already in the requirement specification, is not a very useful method from a practical point of view.

Firstly, such boundedness constraints may have a very complicating effect and thereby lead to a reduced understanding of the system to be developed. Boundedness constraints also complicate formal reasoning and design. Thus, it seems sensible to avoid imposing these constraints as long as possible – in other words, to impose these boundedness constraints only in the later phases of a system development.

Secondly, the exact nature of these constraints is often not known when the requirement specification is written. For example, in the requirement engineering phase of a system development, it is often not clear in what programming language(s) the system is to be implemented or on what sort of architecture the system is supposed to run. Thus, in that case, it is known that some boundedness constraints are to be imposed, but not exactly what these are.

* Email: stoelen@informatik.tu-muenchen.de.

0167-6423/96/$15.00 © 1996 Elsevier Science B.V. All rights reserved
SSDI 0167-6423(95)00031-3

On the other hand, since any computer system has only a bounded amount of memory, it is clear that at some point in a system development such boundedness constraints have to be imposed. Thus, in a system development it must be possible to move from system specifications based on unbounded resources to system specifications based on bounded resources. Unfortunately, the usual principles of behavioral and interface refinement do not always support this type of refinements.

In this paper we concentrate on a particular aspect of this problem, namely the transition from system specifications based on (unbounded) asynchronous communication to system specifications based on (bounded) synchronous communication. We distinguish between two synchronization techniques, namely synchronization by *hand-shake* and synchronization by *real-time constraints*. By synchronization by hand-shake we mean all sorts of time-independent, demand driven or acknowledgment-based synchronization.

We propose three refinement principles, namely *partial*, *total* and *conditional* refinement. Partial and total refinement support synchronization by hand-shake. Partial refinement is restricted to specifications which only impose safety properties. Total refinement preserves both safety and liveness properties, but is not as general as we would have liked. Conditional refinement supports both synchronization by hand-shake and by real-time constraints.

The rest of this paper is split into five sections. In Section 2 we introduce the underlying semantics. In Section 3 we explain what we mean by a specification, and we define the usual principle of behavioral refinement. In Section 4 we introduce the three refinement principles, namely partial, total and conditional refinement, and show how they can be used to support synchronization by hand-shake. In Section 5 we show how conditional refinement can be used to support synchronization by real-time constraints. Finally, there is a conclusion giving a brief summary and a comparison to approaches known from the literature.

2. Semantic model

We represent the communication histories of channels by *timed streams*. A timed stream is a finite or infinite sequence of messages and time ticks. A time tick is represented by $\surd$. The interval between two consecutive ticks represents the least unit of time. A tick occurs in a stream at the end of each time unit.

An infinite timed stream represents a complete communication history; a finite timed stream represents a partial communication history. Since time never halts, any infinite timed stream is required to have infinitely many ticks. We do not want timed streams to end in the middle of a time unit. Thus, we insist that a timed stream is either empty, infinite or ends with a tick.

Given a set of messages M, by M^{∞}, M^{*} and M^{ω} we denote, respectively, the set of all *infinite* timed streams over M, the set of all *finite* timed streams over M, and the set of all *finite* and *infinite* timed streams over M. We use N to denote the set of natural numbers, and N_{∞} to denote $\mathsf{N} \cup \{\infty\}$. Given $s \in M^{\omega}$ and $j \in \mathsf{N}_{\infty}$, $s{\downarrow_j}$

denotes the prefix of s characterizing the behavior until time j, i.e., $s{\downarrow_j}$ denotes s if j is greater than the number of ticks in s, and the shortest prefix of s containing j ticks, otherwise. Note that $s{\downarrow_\infty} = s$. This operator is overloaded to tuples of timed streams in a point-wise style, i.e., $t{\downarrow_j}$ denotes the tuple we get by applying $\downarrow_j$ to each component of t.

A *named stream tuple* is a mapping $\alpha \in a \to M^\omega$ from a set of channel identifiers to timed streams. Intuitively, α assigns a (possibly partial) communication history to each channel named by the channel identifiers in a. The operator $\downarrow$ is overloaded to named stream tuples in the same point-wise style as for tuples of timed streams.

Given two named stream tuples $\alpha \in a \to M^\omega$, $\beta \in b \to M^\omega$ such that $a \cap b = \emptyset$; by $\alpha \uplus \beta$ we denote their *disjoint union*, i.e., the element of $a \cup b \to M^\omega$ such that

$$c \in a \Rightarrow (\alpha \uplus \beta)(c) = \alpha(c), \qquad c \in b \Rightarrow (\alpha \uplus \beta)(c) = \beta(c).$$

Moreover, for any set of identifiers b, $\alpha|_b$ denotes the *projection* of α on b, i.e., $\alpha|_b$ is the element of $a \cap b \to M^\omega$ such that

$$c \in a \cap b \Rightarrow (\alpha|_b)(c) = \alpha(c).$$

A function

$$\tau \in (i \to M^\infty) \to (o \to M^\infty)$$

mapping named stream tuples to named stream tuples is *pulse-driven* iff

$$\forall \alpha, \beta \in i \to M^\infty : j \in \mathsf{N} : \alpha{\downarrow_j} = \beta{\downarrow_j} \Rightarrow \tau(\alpha){\downarrow_{(j+1)}} = \tau(\beta){\downarrow_{(j+1)}}.$$

Pulse-drivenness means that the input until time j completely determines the output until time $j+1$. In other words, a pulse-driven function imposes a delay of at least one time unit between input and output and is in addition "lazy" in the sense that the function can be (partially) computed based on partial input. We use the arrow $\xrightarrow{p}$ to distinguish pulse-driven functions from functions that are not pulse-driven.

We model specifications by sets of pulse-driven functions. Each function or subset of functions contained in such a set represents one possible implementation. For example, a specification of a component, whose input and output channels are named by i and o, respectively, is modeled by a set of pulse-driven functions F such that $F \subseteq (i \to M^\infty) \xrightarrow{p} (o \to M^\infty)$.

Pulse-driven functions can be composed into *networks* of functions – networks which themselves behave as pulse-driven functions. For this purpose we introduce a composition operator $\otimes$. It can be understood as a *parallel operator with hiding*. For example, the network pictured in Fig. 1 consisting of the two functions

$$\tau_1 \in (i_1 \to M^\infty) \xrightarrow{p} (o_1 \to M^\infty), \qquad \tau_2 \in (i_2 \to M^\infty) \xrightarrow{p} (o_2 \to M^\infty),$$

where $i_1 \cap i_2 = o_1 \cap o_2 = i_1 \cap o_1 = i_2 \cap o_2 = \emptyset$, is characterized by $\tau_1 \otimes \tau_2$. Informally speaking, any output channel of τ_1 and input channel of τ_2, and any output channel of

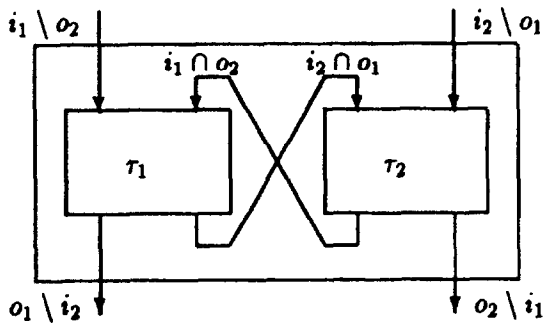

Fig. 1. Network characterized by $\tau_1 \otimes \tau_2$.

τ_2 and input channel of τ_1, whose names are *identical*, are connected and hidden in the sense that they cannot be observed from the outside.

Given that

$$i = (i_1 \setminus o_2) \cup (i_2 \setminus o_1), \qquad o = (o_1 \setminus i_2) \cup (o_2 \setminus i_1),$$

for any $\alpha \in i \rightarrow M^{\infty}$, we define

$$(\tau_1 \otimes \tau_2)(\alpha) = \psi|_o \uplus \theta|_o \quad \textbf{where} \quad \psi = \tau_1(\alpha|_{i_1} \uplus \theta|_{i_1}), \quad \theta = \tau_2(\alpha|_{i_2} \uplus \psi|_{i_2}).$$

Note that the pulse-drivenness of τ_1 and τ_2 implies [1] that for any α there are unique ψ, θ such that

$$\psi = \tau_1(\alpha|_{i_1} \uplus \theta|_{i_1}), \qquad \theta = \tau_2(\alpha|_{i_2} \uplus \psi|_{i_2}).$$

Thus, $\tau_1 \otimes \tau_2$ is well-defined. It is also easy to prove that $\tau_1 \otimes \tau_2$ is pulse-driven. As will be shown below, the composition operator $\otimes$ can be lifted from functions to specifications in a straightforward way.

3. Specification and refinement

We now explain what we mean by a specification. In fact, we introduce two different specification formats, namely formats for *time-dependent* and *time-independent* specifications. The former format differs from the latter in that it allows real-time constraints to be imposed. We also introduce the usual principle of behavioral refinement. However, first we define some useful operators on streams.

3.1. Operators on streams

We also use streams without ticks. We refer to such streams as *untimed*. Given a set of messages M, then $M^{\overline{\infty}}$, $M^{\overline{*}}$ and $M^{\overline{\omega}}$ denote, respectively, the set of all *infinite*

[1] As a consequence of Banach's fix-point theorem [2], since pulse-driven functions can be understood as contracting functions in a complete metric space.

untimed streams over M, the set of all *finite* untimed streams over M, and the set of all *finite* and *infinite* untimed streams over M.

Given $A \subseteq M \cup \{\surd\}$, (timed or untimed) streams r and s over M, and integer j:

- $\#r$ denotes the *length* of r, i.e. ∞ if r is infinite, and the number of elements in r otherwise. Note that time ticks are counted.
- $\langle a_1, a_2, .., a_n \rangle$ denotes the stream of length n whose first element is a_1, whose second element is a_2, and so on. $\langle\rangle$ denotes the *empty* stream.
- $A ⓒ r$ denotes the result of *filtering* away all messages (ticks included) not in A. If $A = \{d\}$ we write $d ⓒ r$ instead of $\{d\} ⓒ r$. For example

$$\{a, b\} ⓒ \langle a, b, \surd, c, \surd, a, \surd \rangle = \langle a, b, a \rangle.$$

- $r|_j$ denotes $\langle\rangle$ if $j \leqslant 0$, the prefix of r of length j if $0 < j < \#r$, and r otherwise. We define $r|_\infty = r$. This operator is overloaded to stream tuples in a point-wise way. Note the way $\downarrow$ differs from this operator.
- $r \frown s$ denotes the result of *concatenating* r and s. Thus, $\langle a, b \rangle \frown \langle c, d \rangle = \langle a, b, c, d \rangle$. If r is infinite we have that $r \frown s = r$.
- $\overline{r}$ denotes the result of *removing* all ticks in r. Thus, $\overline{\langle a, \surd, b, \surd \rangle} = \langle a, b \rangle$.

3.2. Specification formats

We write *time-dependent* specifications in the following form:

$$S \equiv (i \rhd o) \overset{\mathsf{td}}{::} R$$

S is the specification's name, and i and o are finite, repetition free lists of identifiers. The identifiers in i name the input channels, and the identifiers in o name the output channels. The lists are not allowed to have identifiers in common. We refer to the elements of these lists as the *input* and *output* identifiers, respectively. The label **td** is used to distinguish time-dependent specifications from time-independent specifications. As we will see below, the latter are labeled by **ti**. R is a formula in predicate logic with the identifiers of i and o as its only free variables. In R each of these identifiers represents a timed infinite stream modeling the complete communication history of the channel named by the identifier. Thus, i and o name the input and output channels, respectively, and R characterizes the relationship between their communication histories. We will often refer to R as the *i/o-relation* and to $(i \rhd o)$ as the *syntactic interface*.

For any mapping $\alpha \in C \to D$ and formula P, whose free variables are contained in C and vary over D, $\alpha \models P$ holds iff P evaluates to true when each free variable c in P is interpreted as $\alpha(c)$.

Since there is an injective mapping from repetition free lists to totally ordered sets, we will often treat such lists as if they were sets. The *denotation* of a time-dependent specification $S \equiv (i \rhd o) \overset{\mathsf{td}}{::} R$ can then be defined as follows:

$$[\![S]\!] \overset{\mathsf{def}}{=} \{\tau \in (i \to M^\infty) \xrightarrow{p} (o \to M^\infty) \mid \forall \alpha : (\alpha \uplus \tau(\alpha)) \models R\}.$$

A *time-independent* specification can only be used to specify the time-independent behavior of a component. A time-independent specification has almost the same syntactic structure as a time-dependent specification

$$S \equiv (i \rhd o) \overset{\mathsf{ti}}{::} R$$

The only difference is that the label td has been replaced by ti and that the input and output identifiers occurring in R now vary over arbitrary untimed streams. We allow these streams to be finite since a timed infinite stream with only finitely many ordinary messages degenerates to a finite stream when the ticks are removed.

Given a named stream tuple $\alpha \in a \to M^{\infty}$, by $\overline{\alpha}$ we denote the element of $a \to M^{\overline{\omega}}$ such that $\forall c \in a : \overline{\alpha}(c) = \overline{\alpha(c)}$. The *denotation* of a time-independent specification $S \equiv (i \rhd o) \overset{\mathsf{ti}}{::} R$ can then be defined follows:

$$[\![S]\!] \overset{\mathsf{def}}{=} \{\tau \in (i \to M^{\infty}) \xrightarrow{p} (o \to M^{\infty}) \mid \forall \alpha : \overline{(\alpha \uplus \tau(\alpha))} \models R\}.$$

The *composition* operator $\otimes$ can be lifted from pulse-driven functions to specifications in a straightforward way. Let S_1 and S_2 be two specifications whose syntactic interfaces are characterized by $(i_1 \rhd o_1)$ and $(i_2 \rhd o_2)$, respectively. If $i_1 \cap i_2 = o_1 \cap o_2 = \emptyset$, $i = (i_1 \setminus o_2) \cup (i_2 \setminus o_1)$ and $o = (o_1 \setminus i_2) \cup (o_2 \setminus i_1)$, then $S_1 \otimes S_2$ denotes the network pictured in Fig. 1 with τ_1 and τ_2 replaced by S_1 and S_2, respectively. We define $[\![S_1 \otimes S_2]\!]$ to be the set of all

$$\tau \in (i \to M^{\infty}) \xrightarrow{p} (o \to M^{\infty})$$

such that

$$\forall \alpha \in (i \to M^{\infty}) : \exists \tau_1 \in [\![S_1]\!], \tau_2 \in [\![S_2]\!] : \tau(\alpha) = (\tau_1 \otimes \tau_2)(\alpha).$$

Note that this definition is not equivalent to the point-wise composition of the functions in $[\![S_1]\!]$ and $[\![S_2]\!]$. However, this alternative denotation based on point-wise composition, obtained by moving the two existential quantifiers ahead of the universal, is of course contained in $[\![S_1 \otimes S_2]\!]$. In fact, the way $[\![\quad]\!]$ is defined implies that for any specification S and function τ, if

$$\forall \theta : \exists \tau' \in [\![S]\!] : \tau(\theta) = \tau'(\theta)$$

then $\tau \in [\![S]\!]$. Thus, the denotation of a specification is always closed in this sense. This closure property makes our model fully abstract [5] and simplifies the definitions of refinement.

In the sequel we distinguish between basic and composite specifications. The latter differ from the former in that they consist of several specifications composed by $\otimes$.

A time-independent specification

$$S \equiv (i \rhd o) \overset{\mathsf{ti}}{::} R$$

is said to be *safe* if it only imposes safety properties. To formally characterize what this means, we introduce some helpful notations. For any formula P, repetition free list

of identifiers a, and list of expressions c of the same length as a, by $P[{}^{a}_{c}]$ we denote the result of replacing each occurrence of an element of a in P by the corresponding element of c. Moreover, for any repetition free list of identifiers a, we use $a \in T$ to declare each element of a to be of type T. Finally, for any lists of expressions a and c of the same length, $a \sqsubseteq c$ holds iff each element of a is a prefix of the corresponding element of c. We may then formally define S to be safe iff

$$\forall i \in M^{\overline{\omega}}, o \in M^{\overline{\omega}} : R \Leftrightarrow \forall o' \in M^{\overline{*}} : o' \sqsubseteq o \Rightarrow R[{}^{o}_{o'}].$$

3.3. Behavioral refinement

We represent the usual principle of *behavioral refinement* by $\rightsquigarrow$. It holds only for specifications whose syntactic interfaces are identical. Given two specifications S_1 and S_2, then $S_1 \rightsquigarrow S_2$ iff $[\![S_2]\!] \subseteq [\![S_1]\!]$. Thus, S_2 is a behavioral refinement of S_1 iff any pulse-driven function which satisfies S_2 also satisfies S_1.

Clearly, $\rightsquigarrow$ characterizes a *reflexive* and *transitive* relation on specifications. Moreover, it is also a *congruence* modulo $\otimes$ in the sense that

$$S_1 \rightsquigarrow \tilde{S}_1 \wedge S_2 \rightsquigarrow \tilde{S}_2 \Rightarrow S_1 \otimes S_2 \rightsquigarrow \tilde{S}_1 \otimes \tilde{S}_2.$$

4. Synchronization by hand-shake

As already mentioned, in this paper we consider two synchronization techniques, namely synchronization by *hand-shake* and synchronization by *real-time constraints*. In this section we propose refinement principles supporting the former.

The close relationship between specification formalisms based on hand-shake communication and purely asynchronous communication is well-documented in the literature. For example, [6] shows how the process algebra of CSP can be extended to handle asynchronous communication by representing each asynchronous communication channel by a separate process. A similar technique allows different types of hand-shake communication to be introduced in a system specification based on purely asynchronous communication: each asynchronous channel is refined into a network of two components which internally communicate in a synchronous manner, and which externally behave like the identity component.

Consider a network consisting of two time-independent specifications S_1 and S_2 communicating *purely asynchronously* via an internal channel y, as indicated by Network 1 of Fig. 2. We want to refine Network 1 into a network of two specifications $\tilde{S}_1$ and $\tilde{S}_2$ communicating in a *synchronous* manner employing some sort of *hand-shake* protocol – in other words, into a network of the same form as Network 4 of Fig. 2.

Using the technique proposed above, we may move from Network 1 to Network 4 in three steps, employing the usual principle of behavioral refinement:

Step 1: Insert an identity specification I between S_1 and S_2 of Network 1, as indicated by Network 2 of Fig. 2. It follows trivially that

$$S_1 \otimes S_2 \rightsquigarrow S_1 \otimes I \otimes S_2.$$

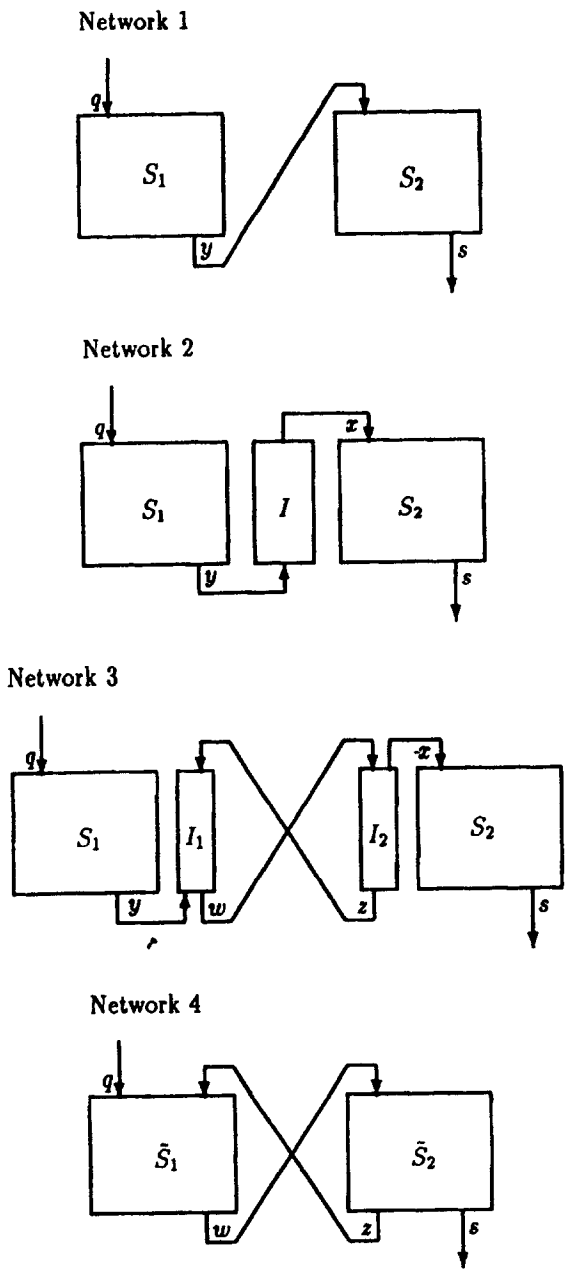

Fig. 2. Introducing synchronization by hand-shake.

Thus, Network 2 is a behavioral refinement of Network 1.

Step 2: Refine the identity specification into two sub-specifications I_1 and I_2 which communicate in accordance with the desired protocol. We then get Network 3 of Fig. 2. Clearly, it must be shown that

$$I \rightsquigarrow I_1 \otimes I_2,$$

in which case it follows by transitivity and congruence of $\rightsquigarrow$ that Network 3 is a behavioral refinement of Network 1.

Step 3: Finally, if we can show that

$$S_1 \otimes I_1 \rightsquigarrow \tilde{S}_1, \qquad I_2 \otimes S_2 \rightsquigarrow \tilde{S}_2,$$

we have that

$$S_1 \otimes S_2 \rightsquigarrow \tilde{S}_1 \otimes \tilde{S}_2$$

by transitivity and congruence of $\rightsquigarrow$. Thus, in that case, Network 4 is a behavioral refinement of Network 1.

Unfortunately, this strategy is rather tedious, and more importantly: it can only be employed to internal channels. To handle external channels accordingly, a more general refinement principle than behavioral refinement is needed. This refinement principle must allow for the introduction of additional feedback loops. For example, without this generality it is not possible to synchronize the communication between S_1 and S_2 in Network 1, using a hand-shake protocol. Of course, one may argue that the synchronization could be conducted via the environment, but this is not what we want. Thus, with respect to our example, this generality is needed in order to build up a connection from S_2 to S_1 allowing S_2 to communicate acknowledgments or demands.

4.1. Partial refinement

Consider two time-independent specifications S and $\tilde{S}$ such that $(i \triangleright o)$ is the syntactic interface of S. In the previous section we have seen that a refinement principle supporting synchronization by hand-shake must allow for the introduction of additional feedback loops. This implies that if $\tilde{S}$ is a refinement of S in this sense, $\tilde{S}$ must be allowed to have additional input and output channels. Thus, given that $(\tilde{i} \triangleright \tilde{o})$ is the syntactic interface of $\tilde{S}$, we assume that $i \subseteq \tilde{i}$ and $o \subseteq \tilde{o}$.

We now want to characterize what it means for $\tilde{S}$ to refine S. If only the "old" channels are considered, one might expect this to be equivalent to insisting that for any function $\tilde{\tau}$ satisfying $\tilde{S}$ and any input history there is a function τ satisfying S which behaves in the same way as $\tilde{\tau}$ with respect to this input history. However, due to the synchronization conducted via the new channels, the computation of $\tilde{\tau}$ can be halted too early because the required acknowledgments or demands are not received. Thus, in the general case, unless we make certain assumptions about the environment's behavior, this requirement is too strong. On the other hand, since a safety property only

says something about what a component is not allowed to do, and nothing about what it has to do, the possibility that the computation of $\tilde{\tau}$ is halted too early is not a problem if S is safe. Thus, the proposed definition is adequate if we are only interested in safety properties. Formally, given that S and $\tilde{S}$ are safe, we say that $\tilde{S}$ is a *partial refinement* of S, written $S \overset{p}{\rightsquigarrow} \tilde{S}$, iff

$$\forall \tilde{\tau} \in [\![\tilde{S}]\!], \alpha \in \tilde{i} \to M^{\infty} : \exists \tau \in [\![S]\!] : \tilde{\tau}(\alpha)|_o = \tau(\alpha|_i).$$

Note that if $i = \tilde{i}$ and $o = \tilde{o}$ then $\overset{p}{\rightsquigarrow}$ degenerates to $\rightsquigarrow$ with respect to safe specifications. It is straightforward to prove that $\overset{p}{\rightsquigarrow}$ characterizes a reflexive and transitive relation. Moreover, it is also easy to prove that $\overset{p}{\rightsquigarrow}$ is a congruence with respect to $\otimes$ in the same sense as $\rightsquigarrow$. Thus, partial refinement has the same nice properties as behavioral refinement and is therefore equally well suited as a refinement principle for modular system development. Unfortunately, most specifications are not safe – they also impose liveness constraints. Thus, a more powerful refinement principle is needed.

4.2. Total refinement

Consider once more the two time-independent specifications of the previous section. As already argued, since the computation of a component satisfying $\tilde{S}$ can be halted too early because a required acknowledgment or demand is not received, the definition of partial refinement is too strong if S also imposes liveness properties. In that case, the relation has to be weakened by some sort of environment assumption – an assumption constraining the communication histories of the "new" input channels. Let $\alpha \in \tilde{i} \to M^{\infty}$ and assume $\hat{i}$ is the set of new input channels, i.e. $\hat{i} = \tilde{i} \setminus i$. For many synchronization protocols it is enough to require that on each *new* input channel infinitely many messages are received. We use $\inf(\alpha, \hat{i})$ to denote this environment assumption. Formally,

$$\inf(\alpha, \hat{i}) \overset{\text{def}}{=} \forall c \in \hat{i} : \#\overline{\alpha(c)} = \infty.$$

Based on this environment assumption, we define $\tilde{S}$ to be a *total refinement* of S, written $S \overset{t}{\rightsquigarrow} \tilde{S}$, iff

$$\forall \tilde{\tau} \in [\![\tilde{S}]\!], \alpha \in \tilde{i} \to M^{\infty} : \exists \tau \in [\![S]\!] : \inf(\alpha, \hat{i}) \Rightarrow \tilde{\tau}(\alpha)|_o = \tau(\alpha|_i).$$

It is easy to see that total refinement degenerates to behavioral refinement if $i = \tilde{i}$ and $o = \tilde{o}$. Moreover, due to the pulse-drivenness constraint imposed on the functions characterizing the denotation of a time-independent specification, it follows that total refinement implies partial refinement if the specifications are safe. It is also easy to prove that $\overset{t}{\rightsquigarrow}$ characterizes a reflexive and transitive relation on time-independent specifications. Unfortunately, $\overset{t}{\rightsquigarrow}$ is not a congruence with respect to $\otimes$. In the following, we use R_S to represent the i/o-relation of a basic specification S.

Example 1 (*Total refinement is not a congruence*). To see that total refinement is not a congruence with respect to $\otimes$, consider the four time-independent specifications $S_1, S_2, \tilde{S}_1, \tilde{S}_2$, whose syntactic interfaces are characterized by $(q \rhd z)$, $(z \rhd k)$, $(q, x \rhd z)$, $(z \rhd x, k)$, respectively, and whose i/o-relations are defined as below

$$R_{S_1} \stackrel{\text{def}}{=} z = q, \quad R_{\tilde{S}_1} \stackrel{\text{def}}{=} z = q|_{\#x+1},$$
$$R_{S_2} \stackrel{\text{def}}{=} k = z, \quad R_{\tilde{S}_2} \stackrel{\text{def}}{=} k = z \wedge x = z|_{\#z-1}.$$

Note that x is a new feedback channel from $\tilde{S}_2$ to $\tilde{S}_1$. Clearly, $S_1 \overset{t}{\rightsquigarrow} \tilde{S}_1$ and $S_2 \overset{t}{\rightsquigarrow} \tilde{S}_2$. Since

$$R_{S_1} \wedge R_{S_2} \Rightarrow k = q,$$

it follows that $S_1 \otimes S_2$ behaves as an identity component. On the other hand, by inspecting $\tilde{S}_1$ and $\tilde{S}_2$, it is clear that any correct implementation of $\tilde{S}_1$ can send a second message along z only after having received at least one acknowledgment along x. Moreover, it is also clear that any correct implementation of $\tilde{S}_2$ can output the first acknowledgment along x only after having received at least two messages along z. These causality constraints are semantically imposed via the pulse-drivenness[2]. Thus, any correct implementation of $\tilde{S}_1 \otimes \tilde{S}_2$ will never output more than one message along k. Since both $S_1 \otimes S_2$ and $\tilde{S}_1 \otimes \tilde{S}_2$ have q as their only input channel, and since it may be the case that $\#q > 1$, it follows that

$$S_1 \otimes S_2 \overset{t}{\not\rightsquigarrow} \tilde{S}_1 \otimes \tilde{S}_2.$$

The problem observed in Example 1 can be understood as *deadlock* caused by an erroneous synchronization protocol. What is required is some proof obligation, more explicitly – some freedom from deadlock test, characterizing under what conditions total refinement is a "congruence" with respect to $\otimes$. Firstly, we want a proof obligation which takes advantage of the fact that we have already proved that $S_1 \overset{t}{\rightsquigarrow} \tilde{S}_1$ and $S_2 \overset{t}{\rightsquigarrow} \tilde{S}_2$. This suggests it should be independent of S_1 and S_2. Secondly, to allow systems to be developed in a top-down style, this proof obligation must be checkable based on the information available at the point in time where the refinement step is carried out. For example, it should not require knowledge about how $\tilde{S}_1$ and $\tilde{S}_2$ are implemented.

With respect to a network as in Example 1, it is enough to check that, when the computation halts, then the output along z will not be extended if additional input is received on the feedback channel x. This is equivalent to verifying the proof obligation below

$$R_{\tilde{S}_1} \wedge R_{\tilde{S}_2} \Rightarrow R_{\tilde{S}_1}[{}^{x}_{x^\frown x'}],$$

[2] Remember that also time-independent specifications are interpreted in terms of pulse-driven functions and timed streams.

where x' is a new identifier. We now prove that this proof obligation guarantees $S_1 \otimes S_2 \overset{t}{\rightsquigarrow} \tilde{S}_1 \otimes \tilde{S}_2$, given that $S_1 \overset{t}{\rightsquigarrow} \tilde{S}_1$, $S_2 \overset{t}{\rightsquigarrow} \tilde{S}_2$ and that the syntactic interfaces are as in Example 1.

Let $\tilde{\tau} \in [\![\tilde{S}_1 \otimes \tilde{S}_2]\!]$, $\alpha \in \{q\} \to M^\infty$. The definition of $\otimes$ implies there are $\tilde{\tau}_1 \in [\![\tilde{S}_1]\!]$, $\tilde{\tau}_2 \in [\![\tilde{S}_2]\!]$ such that $\tilde{\tau}(\alpha) = (\tilde{\tau}_1 \otimes \tilde{\tau}_2)(\alpha)$. This means there are $\beta \in \{k\} \to M^\infty$, $\delta \in \{z\} \to M^\infty$, $\sigma \in \{x\} \to M^\infty$ such that

$$\tilde{\tau}_1(\alpha \uplus \sigma) = \delta, \quad \tilde{\tau}_2(\delta) = \sigma \uplus \beta, \quad (\tilde{\tau}_1 \otimes \tilde{\tau}_2)(\alpha) = \beta.$$

There are two cases to consider:

- If $\inf(\sigma, x)$ then $S_1 \overset{t}{\rightsquigarrow} \tilde{S}_1$ and $S_2 \overset{t}{\rightsquigarrow} \tilde{S}_2$ imply there are $\tau_1 \in [\![S_1]\!]$, $\tau_2 \in [\![S_2]\!]$ such that $\tau_1(\alpha) = \delta$ and $\tau_2(\delta) = \beta$.
- If $\neg\inf(\sigma, x)$ then there is a $j \in \mathbb{N}$ such that $\sigma(x)\downarrow_j {}^\frown \surd^\infty = \sigma(x)$, where $\surd^\infty$ denotes an infinite stream of ticks. Let $\tilde{\tau}_1'$ be the function such that

$$(\alpha \uplus \sigma)\downarrow_j \not\sqsubseteq w \Rightarrow \tilde{\tau}_1'(w) = \tilde{\tau}_1(w),$$

$$(\alpha \uplus \sigma\downarrow_j) \sqsubseteq w \Rightarrow \tilde{\tau}_1'(w) = \tilde{\tau}_1(\alpha \uplus \sigma),$$

$$\alpha\downarrow_k \sqsubseteq v \wedge \alpha\downarrow_{(k+1)} \not\sqsubseteq v \wedge k \geqslant j \Rightarrow \tilde{\tau}_1'(v \uplus (\sigma\downarrow_j {}^\frown u)) = \tilde{\tau}_1(v \uplus (\sigma\downarrow_k {}^\frown u)).$$

  The concatenation and prefix operators are here overloaded to named stream tuples in the obvious point-wise way. $\tilde{\tau}_1'$ is clearly well-defined and pulse-driven. Moreover, it follows straightforwardly from the proof obligation that $\tilde{\tau}_1' \in [\![\tilde{S}_1]\!]$. Clearly,

$$\tilde{\tau}_1'(\alpha \uplus (\sigma\downarrow_j {}^\frown u)) = \tilde{\tau}_1'(\alpha \uplus \sigma) = \tilde{\tau}_1(\alpha \uplus \sigma).$$

  Then $S_1 \overset{t}{\rightsquigarrow} \tilde{S}_1$ and $S_2 \overset{t}{\rightsquigarrow} \tilde{S}_2$ imply there are $\tau_1 \in [\![S_1]\!]$, $\tau_2 \in [\![S_2]\!]$ such that $\tau_1(\alpha) = \delta$ and $\tau_2(\delta) = \beta$.

Since $\tau_1 \in [\![S_1]\!]$, $\tau_2 \in [\![S_2]\!]$ imply $\tau_1 \otimes \tau_2 \in [\![S_1 \otimes S_2]\!]$, it then follows that $S_1 \otimes S_2 \overset{t}{\rightsquigarrow} \tilde{S}_1 \otimes \tilde{S}_2$, which is what we wanted to prove.

With respect to Example 1, if the i/o-relation of $\tilde{S}_2$ is redefined as below

$$R_{\tilde{S}_2} \overset{\text{def}}{=} k = z \wedge x = z,$$

it follows by the proposed proof obligation that $S_1 \otimes S_2 \overset{t}{\rightsquigarrow} \tilde{S}_1 \otimes \tilde{S}_2$.

We now show how this proof obligation can be generalized to handle arbitrary composition modulo $\otimes$.

Example 2 (*Handling additional external input channels*). To indicate the weakness of the test we have already formulated, we go through another example. Let S_1, S_2 and $\tilde{S}_1$ be as in Example 1, and let

$$\tilde{S}_2 \equiv (z, i \rhd x, k) :: k = z \wedge x = z|_{\#i}.$$

The behavior of $\tilde{S}_2$ now depends on an additional input channel i. As before, $S_1 \overset{t}{\rightsquigarrow} \tilde{S}_1$ and $S_2 \overset{t}{\rightsquigarrow} \tilde{S}_2$. Moreover, it is also clear that $S_1 \otimes S_2 \overset{t}{\rightsquigarrow} \tilde{S}_1 \otimes \tilde{S}_2$. Unfortunately, our proof obligation does not hold. For example, we have that

$$\#q > 1 \wedge i = \langle\rangle \wedge \#z = 1 \Rightarrow R_{\tilde{S}_1} \wedge R_{\tilde{S}_2}, \quad R_{\tilde{S}_1}[{}^{x}_{x^\frown x'}] \wedge \#x' = \infty \Rightarrow z = q.$$

Thus, since x' does not occur in the antecedent of the proof obligation, it follows that it is falsified by at least this instantiation.

With respect to Example 2, the problem is that our proof obligation does not take the new channel i into account. Since i is not an output channel of $\tilde{S}_1$, but connected to the overall environment, the implicit environment assumption built into the definition of total refinement implies we only have to consider the situation that infinitely many messages are received on i. Thus, the proof obligation can be weakened as below:

$$\forall i \in M^{\overline{\infty}} : R_{\tilde{S}_1} \wedge R_{\tilde{S}_2} \Rightarrow R_{\tilde{S}_1}[{}^{x}_{x^\frown x'}].$$

Note that this proof obligation is satisfied by the refinement step considered in Example 2.

It is now straightforward to formulate a general proof obligation. Let x, y, i be lists consisting of, respectively, the new input channels of $\tilde{S}_1$ connected to $\tilde{S}_2$, the new input channels of $\tilde{S}_2$ connected to $\tilde{S}_1$, and the new input channels of $\tilde{S}_1$ and $\tilde{S}_2$ connected to the overall environment. Then we get the following refinement rule:

$$S_1 \overset{t}{\rightsquigarrow} \tilde{S}_1$$

$$S_2 \overset{t}{\rightsquigarrow} \tilde{S}_2$$

$$\underline{\forall i \in M^{\overline{\infty}} : R_{\tilde{S}_1} \wedge R_{\tilde{S}_2} \Rightarrow R_{\tilde{S}_1}[{}^{x}_{x^\frown x'}] \wedge R_{\tilde{S}_2}[{}^{y}_{y^\frown y'}]}$$

$$S_1 \otimes S_2 \overset{t}{\rightsquigarrow} \tilde{S}_1 \otimes \tilde{S}_2$$

It is assumed that the specifications are basic. The rule can easily be generalized to deal with $n > 2$ specifications.

The proof that this rule is sound is a straightforward generalization of the proof for the restricted case given above. See [11] for details. Note that this rule does not require proof work conducted earlier in the development process to be redone. The two first premises can be checked locally; the third premise is a co-existence check making sure that no deadlock has been introduced.

As already mentioned, although total refinement is sufficient for many hand-shake protocols, this principle is not as general as we would have liked. The problem is that certain synchronization protocols impose *fairness* constraints on the distribution of acknowledgments or demands sent along a channel.

Example 3 (*Lack of generality*). To see the lack of generality, let S_1 and S_2 be defined as in Example 1. Moreover, assume that $\tilde{S}_1$ and $\tilde{S}_2$ have the same syntactic interfaces as in Example 1, and that their i/o-relations are redefined as below

$$R_{\tilde{S}_1} \stackrel{\text{def}}{=} z = q|_{\#(1©x)+1}, \quad R_{\tilde{S}_2} \stackrel{\text{def}}{=} k = z \wedge \#(1©x) = \#z.$$

Clearly, $S_2 \overset{t}{\rightsquigarrow} \tilde{S}_2$. Moreover, we also have that $S_1 \otimes S_2 \overset{t}{\rightsquigarrow} \tilde{S}_1 \otimes \tilde{S}_2$. However, it does not hold that $S_1 \overset{t}{\rightsquigarrow} \tilde{S}_1$. The reason is of course that the implicit environment assumption of total refinement, namely that infinitely many messages are received on x, does not guarantee that the required number of 1's are received.

4.3. Conditional refinement

Consider once more the two time-independent specifications S and $\tilde{S}$ of the two previous sections. As already argued, for certain hand-shake protocols the implicit environment assumption of total refinement is too weak. One way to deal with this problem is to make the environment assumption explicit and let the user himself specify the required assumption. More explicitly, let B be a formula whose free variables are contained in $\tilde{\imath} \cup \tilde{o}$ and vary over untimed streams, we say that $\tilde{S}$ is a *conditional refinement* of S with respect to B, written $S \rightsquigarrow_B \tilde{S}$, iff

$$\forall \tilde{\tau} \in [\![\tilde{S}]\!], \alpha \in \tilde{\imath} \to M^\infty : \exists \tau \in [\![S]\!] : (\overline{\alpha \uplus \tilde{\tau}(\alpha)}) \models B \Rightarrow \tilde{\tau}(\alpha)|_o = \tau(\alpha|_i).$$

Note that the condition B may also refer to the output behavior. This is in some cases necessary since the correct input behavior at some point in time may depend on what has already been output.

It is clear that if $i = \tilde{\imath}$ and $o = \tilde{o}$ then $\rightsquigarrow_{\mathsf{true}}$ corresponds to behavioral refinement. It is also easy to see that for any time-independent specification S and condition B, we have that $S \rightsquigarrow_B S$. Thus, conditional refinement has the required "reflexivity" property. It is also "transitive" in a certain sense

$$S_1 \rightsquigarrow_{B_1} S_2 \wedge S_2 \rightsquigarrow_{B_2} S_3 \Rightarrow S_1 \rightsquigarrow_{B_1 \wedge B_2} S_3.$$

Conditional refinement is not a congruence modulo $\otimes$ in the general case. However, the following refinement rule is valid:

$$\frac{\begin{array}{l} S_1 \rightsquigarrow_{B_1} \tilde{S}_1 \\ S_2 \rightsquigarrow_{B_2} \tilde{S}_2 \\ B \wedge R_{\tilde{S}_1} \wedge R_{\tilde{S}_2} \Rightarrow B_1 \wedge B_2 \end{array}}{S_1 \otimes S_2 \rightsquigarrow_B \tilde{S}_1 \otimes \tilde{S}_2}$$

It is assumed that the specifications are basic. The rule can easily be generalized to deal with $n > 2$ specifications.

Also this rule has the nice property that proof work conducted earlier in the development process does not have to be redone. The two first premises are local constraints; the third is a co-existence check making sure that no deadlock has been introduced.

We now prove that the rule is sound. Let $\tilde{i}_1, \tilde{i}_2, \tilde{o}_1, \tilde{o}_2, \tilde{x}, \tilde{y}$ be mutually disjoint lists of identifiers, and let $\cdot$ be a concatenation operator for such lists. Moreover, let $(i_1 \cdot x \rhd o_1 \cdot y)$, $(y \cdot i_2 \rhd x \cdot o_2)$, $(\tilde{i}_1 \cdot \tilde{x} \rhd \tilde{o}_1 \cdot \tilde{y})$ and $(\tilde{y} \cdot \tilde{i}_2 \rhd \tilde{x} \cdot \tilde{o}_2)$ be the syntactic interfaces of S_1, S_2, $\tilde{S}_1$ and $\tilde{S}_2$, respectively. Assume that $i_1 \subseteq \tilde{i}_1$, $o_1 \subseteq \tilde{o}_1$, $i_2 \subseteq \tilde{i}_2$, $o_2 \subseteq \tilde{o}_2$, $x \subseteq \tilde{x}$, $y \subseteq \tilde{y}$, and that the three premises hold.

Let $\tilde{\tau} \in [\![\tilde{S}_1 \otimes \tilde{S}_2]\!]$, $\alpha_1 \in \tilde{i}_1 \to M^\infty$, $\alpha_2 \in \tilde{i}_2 \to M^\infty$ be such that

$$\overline{(\alpha_1 \uplus \alpha_2 \uplus \tilde{\tau}(\alpha_1 \uplus \alpha_2))} \models B.$$

The definition of $\otimes$ implies there are $\tilde{\tau}_1 \in [\![\tilde{S}_1]\!]$, $\tilde{\tau}_2 \in [\![\tilde{S}_2]\!]$ such that $\tilde{\tau}(\alpha_1 \uplus \alpha_2) = (\tilde{\tau}_1 \otimes \tilde{\tau}_2)(\alpha_1 \uplus \alpha_2)$. It follows there are $\beta_1 \in \tilde{o}_1 \to M^\infty, \beta_2 \in \tilde{o}_2 \to M^\infty, \delta \in \tilde{x} \to M^\infty, \sigma \in \tilde{y} \to M^\infty$, such that

$$\tilde{\tau}_1(\alpha_1 \uplus \delta) = (\beta_1 \uplus \sigma), \quad \tilde{\tau}_2(\sigma \uplus \alpha_2) = (\delta \uplus \beta_2).$$

It follows straightforwardly that $\overline{(\alpha_1 \uplus \alpha_2 \uplus \beta_1 \uplus \beta_2 \uplus \delta \uplus \sigma))} \models B \wedge R_{\tilde{S}_1} \wedge R_{\tilde{S}_2}$, in which case the third premise implies $\overline{(\alpha_1 \uplus \alpha_2 \uplus \beta_1 \uplus \beta_2 \uplus \delta \uplus \sigma))} \models B_1 \wedge B_2$. This and the two first premises imply there are $\tau_1 \in [\![S_1]\!]$, $\tau_2 \in [\![S_2]\!]$ such that

$$\tilde{\tau}_1(\alpha_1 \uplus \delta)|_{o_1 \cup y} = (\beta_1|_{o_1} \uplus \sigma|_y) = \tau_1(\alpha_1|_{i_1} \uplus \delta|_x),$$
$$\tilde{\tau}_2(\sigma \uplus \alpha_2)|_{x \cup o_2} = (\delta|_x \uplus \beta_2|_{o_2}) = \tau_1(\sigma|_y \uplus \alpha_2|_{i_2}).$$

The way this was deduced, the definition of $\otimes$ and the fact we have unique fix-points imply the conclusion. Thus, the soundness of the refinement rule has been verified.

Example 4 (*The refinement step of Example 2*). The correctness of the refinement step of Example 2 follows straightforwardly by the rule proposed above if the three conditions are defined as below

$$B \stackrel{\text{def}}{=} \#i \geqslant \#q, \qquad B_1 \stackrel{\text{def}}{=} \#x \geqslant \#q - 1, \qquad B_2 \stackrel{\text{def}}{=} \mathsf{true}.$$

Example 5 (*The refinement step of Example 3*). The correctness of the refinement step of Example 3 follows straightforwardly by the rule proposed above if the three conditions are defined as below

$$B \stackrel{\text{def}}{=} \mathsf{true}, \qquad B_1 \stackrel{\text{def}}{=} \#(1 \copyright x) \geqslant \#q - 1, \qquad B_2 \stackrel{\text{def}}{=} \mathsf{true}.$$

5. Synchronization by real-time constraints

Above we have shown how partial, total and conditional refinement can be used to support synchronization by hand-shake. In this section we show that conditional refinement also supports synchronization by *real-time constraints*. Timed streams capture

real-time in the sense that the interval between each pair of consecutive time ticks represents the same least unit of time. Consider two time-dependent specifications S and $\tilde{S}$. For simplicity, since we in this section do not consider synchronization by handshake, we assume that both specifications have the same syntactic interface ($i \triangleright o$). Let B be a formula whose free variables are contained in $i \cup o$ and vary over infinite timed streams. We say $\tilde{S}$ is a *conditional refinement* of S with respect to B, iff

$$\forall \tilde{\tau} \in [\![\tilde{S}]\!], \alpha \in i \to M^{\infty} : \exists \tau \in [\![S]\!] : (\alpha \uplus \tilde{\tau}(\alpha)) \models B \Rightarrow \tilde{\tau}(\alpha) = \tau(\alpha).$$

Example 6 (*By real-time constraints*). Consider the two time-dependent specifications defined below

$$S_1 \equiv (q \triangleright y) \overset{\text{td}}{::} \overline{y} = \overline{q}, \qquad S_2 \equiv (y \triangleright s) \overset{\text{td}}{::} \overline{s} = \overline{y}.$$

Each correct implementation of $S_1 \otimes S_2$ requires an unbounded amount of internal memory. The reason is that the overall environment may send arbitrarily many messages between two time ticks along q. Since, due to the pulse-drivenness, any correct implementation delays the output with at least one time unit, S_1 must be able to store arbitrarily many messages. Consider the auxiliary predicate

$$\text{bnd}(i,k) \overset{\text{def}}{=} \forall j \in \mathbf{N} : \#(i\downarrow_{(j+1)}) - \#(i\downarrow_j) \leqslant k.$$

It holds for an infinite timed stream i if the maximum number of messages received between two consecutive time ticks in i is less than k. We may use this predicate to synchronize the communication as below

$$\tilde{S}_1 \equiv (q \triangleright y) \overset{\text{td}}{::} \text{bnd}(q,k) \Rightarrow \overline{y} = \overline{q} \wedge \text{bnd}(y,k),$$

$$\tilde{S}_2 \equiv (y \triangleright s) \overset{\text{td}}{::} \text{bnd}(y,k) \Rightarrow \overline{s} = \overline{y} \wedge \text{bnd}(s,k).$$

In the case of $\tilde{S}_1 \otimes \tilde{S}_2$ we may find an implementation requiring an internal memory capable of storing maximum m messages, where m depends on k and how fast the chosen architecture allows input messages to be forwarded along the output channels. Clearly,

$$S_1 \otimes S_2 \not\rightsquigarrow \tilde{S}_1 \otimes \tilde{S}_2.$$

The reason is that $\tilde{S}_1 \otimes \tilde{S}_2$ may behave arbitrarily as soon as the environment falsifies $\text{bnd}(q,k)$. On the other hand, it is clear that

$$S_1 \otimes S_2 \rightsquigarrow_{\text{bnd}(q,k)} \tilde{S}_1 \otimes \tilde{S}_2.$$

This follows easily since

$$\text{bnd}(q,k) \wedge R_{\tilde{S}_1} \wedge R_{\tilde{S}_2} \Rightarrow \text{bnd}(q,k) \wedge \text{bnd}(y,k).$$

Even if $S \rightsquigarrow_B \tilde{S}$ holds, it may be the case that $\tilde{S}$ allows an implementation which itself breaks the condition B or forces the environment to break the condition B. To avoid such refinements it is enough to impose *well-formedness* conditions on B. One may also formulate well-formedness conditions making sure that the predicate B is

only constraining the behavior related to synchronization. For example, with respect to hand-shake synchronization, one may introduce a well-formedness condition making sure that the condition B only constrains what is received on the "new" feedback channels. However, a detailed discussion of well-formedness conditions is beyond the scope of this paper.

6. Conclusions

In this paper we have introduced three principles of refinement. Their properties can be summed up as below. Partial refinement supports synchronization by hand-shake with respect to safety properties and is a congruence modulo $\otimes$, but does not support synchronization by real-time constraints. Total refinement supports synchronization by hand-shake with respect to both safety and liveness properties and allows modular top-down design, but is not very general and does not support synchronization by real-time constraints. Conditional refinement supports both synchronization by hand-shake and by real-time constraints with respect to both safety and liveness properties and allows modular top-down design. As we see it, the main contribution of this paper is that we have shown how refinement principles based on explicit or implicit environment assumptions can be used to support the transition from system specifications based on purely asynchronous communication to system specifications based on synchronous communication. However, in particular conditional refinement seems to have a much broader application area. We refer to [11] for detailed proofs of the different claims made in this paper.

As explained in [4], behavioral refinement can be generalized to interface refinement by relating the concrete and abstract interface by a representation function in the style of [7]. The three refinement principles proposed above can be generalized accordingly.

We refer to [5] for a detailed investigation of the underlying semantic model.

The principles of partial and total refinement were defined in [12], but in a less general setting. Conditional refinement is a straightforward generalization of behavioral refinement – so straightforward that it seems unlikely that this idea is new. For example, what [1] refers to as conditional implementation is closely related. Moreover, the decomposition theorem of [1] seems to allow related refinements with respect to complete systems. Contrary to us, their co-existence proof is formulated with respect to the more abstract specifications. An attempt to tackle the transition from unbounded to bounded resources in the context of algebraic specifications can be found in [3].

With respect to conditional refinement, instead of using explicit conditions one may calculate the weakest conditions under which the concrete specifications refine the abstract specifications. However, we find the use of explicit conditions more practical.

The refinement principles proposed above can of course be reformulated in other settings. For example, if the refinement principle of the rely/guarantee method [8] is

weakened along the lines proposed in this paper some of the problems reported in [13] seem to disappear.

The proposed refinement principles have not been justified with respect to some sort of observation language as for example advocated in [9]. Instead, the well-suitedness of behavioral refinement as defined in [4] has been taken for granted. Both total and conditional refinement characterize behavioral refinement in the sense of [4] modulo certain assumptions about the environment.

In practice, specifications are often written in an assumption/commitment form. Some of the proof-obligations proposed above can then be replaced by more sophisticated rules. See [10] for assumption/commitment rules with respect to the semantic setting of this paper.

Acknowledgements

The author has benefited from many discussions with Manfred Broy on this and related topics. Pierre Collette, Bernhard Möller and Oscar Slotosch have read erlier drafts of this paper and provided many helpful comments. Financial support has been received from the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen".

References

[1] M. Abadi and L. Lamport, Conjoining specifications, Tech. Report 118, Digital, SRC, Palo Alto, 1993.

[2] P. America, J. de Bakker, J.N. Kok and J. Rutten, Denotational semantics of a parallel object-oriented language, *Inform. Comput.* **83** (1989) 152–205.

[3] M. Breu, Endliche Implementierung algebraischer Spezifikationen, Ph.D. Thesis, Technische Universität München, 1991; Also available as Tech. Report TUM-I9111, Technische Universität München.

[4] M. Broy, Compositional refinement of interactive systems, Tech. Report 89, Digital, SRC, Palo Alto, 1992.

[5] R. Grosu and K. Stølen, A denotational model for mobile point-to-point dataflow networks, Tech. Report SFB 342/14/95 A, Technische Universität München, 1995.

[6] J. He, M. Josephs and C.A.R Hoare, A theory of synchrony and asynchrony, in: *Proc. IFIP WG 2.2/2.3 Working Conf. on Programming Concepts and Methods* (North-Holland, Amsterdam, 1990) 459–478.

[7] C.A.R. Hoare, Proof of correctness of data representations, *Acta Inform.* **1** (1972) 271–282.

[8] C.B. Jones, Specification and design of (parallel) programs, in: *Proc. Information Processing 83* (North-Holland, Amsterdam, 1983) 321–331.

[9] T. Nipkow, Non-deterministic data types: models and implementations, *Acta Inform.* **22** (1986) 629–661.

[10] K. Stølen, Assumption/commitment rules for dataflow networks – with an emphasis on completeness, *Proc. ESOP'96*, 1996, To appear.

[11] K. Stølen, Refinement principles supporting the transition from asynchronous to synchronous communication, Tech. Report SFB 342/20/95 A, Technische Universität München, 1995.

[12] K. Stølen, A refinement relation supporting the transition from unbounded to bounded communication buffers, in: *Proc. MPC'95,* Lecture Notes in Computer Science 947 (Springer, Berlin, 1995) 423–451.

[13] J.C.P. Woodcock and B. Dickinson, Using VDM with rely and guarantee-conditions. Experiences from a real project, in: *Proc. VDM'88,* Lecture Notes in Computer Science 328 (Springer, Berlin, 1988) 434–458.