Specification and Refinement of Finite Dataflow Networks — a Relational Approach*

Manfred Broy, Ketil Stølen

Abstract

We specify the black box behavior of dataflow components by characterizing the relation between the input and the output histories. We distinguish between three main classes of such specifications, namely time independent specifications, weakly time dependent specifications and strongly time dependent specifications. Dataflow components are semantically modeled by sets of timed stream processing functions. Specifications describe such sets by logical formulas. We emphasize the treatment of the well-known fair merge problem and the Brock/Ackermann anomaly. We give refinement rules which allow specifications to be decomposed into networks of specifications.

1 Introduction

Dataflow components can be specified by formulas with a free variable ranging over domains of continuous functions — so-called stream processing functions [Kel78], [BDD$^+$92]. Both time independent and time dependent components can be described this way. In the latter case, the functions are timed in the sense that the input/output streams may have occurrences of a special message representing a time signal. For such specifications elegant refinement calculi [Bro92a], [Bro93] can be formulated, which allow dataflow networks to be developed in the same way as the methods suggested in [Jon90], [Mor90] allow for the development of sequential programs.

The set of functions characterized by a component is called the component's denotation. Components modeled by timed stream processing functions can be classified as *time independent*, *weakly time dependent* or *strongly time dependent*. A component is time independent if at most the timing of the output depends upon the timing of the input. Weakly time dependent components are always nondeterministic, and their output (not only the timing of the output) depends upon the timing of the input. However, due to the nondeterminism this time dependency is hidden in the sense that for inputs, which are identical apart from the timing, we get the same set of outputs, when the time signals are abstracted away. A strongly time dependent component is a component that is neither time independent nor weakly time dependent.

*This work is supported by the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen".

In the case of weakly time dependent components explicit timing is not really needed in order to specify the black box behavior. A famous example of such a component is an agent which outputs a fair merge of the messages it receives on two input channels.

It is well-known that, because of the continuity constraint imposed on stream processing functions, a fair merge component cannot be modeled by a set of (monotonic) untimed stream processing functions [Kel78]. On the other hand, to specify components of this type in terms of timed stream processing functions is a bit like shooting sparrows with a shotgun, since explicit timing is not needed in order to characterize their behavior.

In an attempt to abstract from unnecessary time-dependency, this paper advocates a technique, where the black box behavior of dataflow networks is specified by characterizing the relation between the input and the output histories. We distinguish between three main classes of such specifications, namely *time independent specifications*, *weakly time dependent specifications* and *strongly time dependent specifications* — from now on shortened to *ti-specifications*, *wtd-specifications* and *std-specifications*.

Although ti-, wtd- and std-specifications are mainly intended for the specification of time independent, weakly time dependent and strongly time dependent components, respectively, a wtd-specification may also be used to specify a time independent component, and a std-specification may also be used to specify a time independent or a weakly time dependent component. In fact, as we will see later, in some sense a wtd-specification is a special case of an std-specification, and a ti-specification is a special case of a wtd-specification and an std-specification.

Specifications are semantically modeled by sets of timed stream processing functions. For each specification class refinement rules are given, which allow specifications to be decomposed into networks of specifications. Rules, which allow a specification of one class to be translated into a specification of another class, are also given.

Finally, it is explained how the well-known Brock/Ackermann anomaly [BA81] can be overcome by distinguishing between *simple* and *general* specifications.

Section 2 describes the underlying formalism. Then we introduce ti-specifications, wtd-specifications and std-specifications in Sections 3, 4 and 5, respectively. In Section 6 the three main types of specifications are divided into simple and general specifications. A brief summary and discussion can be found in Section 7. Finally, there is an appendix containing some soundness and completeness proofs.

2 Underlying Formalism

N denotes the set of natural numbers, and N^+ denotes $\mathsf{N} \setminus \{0\}$. We assume the availability of the standard logical operators. As usual, $\Rightarrow$ binds weaker than $\wedge, \vee, \neg$ which again bind weaker than all other operators and function symbols.

A stream is a finite or infinite sequence of messages. It models the history of a communication channel by representing the sequence of messages sent along the channel. Given a set of messages D, D^* denotes the set of all finite streams generated from D; D^∞ denotes the set of all infinite streams generated from D, and D^ω denotes $D^* \cup D^\infty$.

Let $d \in D$, $r, s \in D^\omega$, and j be a natural number, then:

- ϵ denotes the empty stream;
- $\langle d_1, \ldots, d_n \rangle$ denotes a stream of length n, whose first message is d_1, whose second message is d_2, etc. ;
- $\mathsf{ft}(r)$ denotes the first element of r if r is not empty;
- $\# r$ denotes the length of r;
- d^n, where $n \in \mathsf{N} \cup \{\infty\}$, denotes a stream of length n consisting of only d's;
- $r|_j$ denotes the prefix of r of length j if $j < \# r$, and r otherwise;
- $d \,\&\, s$ denotes the result of appending d to s;
- $r \frown s$ denotes r if r is infinite and the result of concatenating r with s, otherwise;
- $r \sqsubseteq s$ holds if r is a prefix of s.

Some of the stream operators defined above are overloaded to tuples of streams in a straightforward way. ϵ will also be used to denote tuples of empty streams when the size of the tuple is clear from the context. If d is an n-tuple of messages, and r, s are n-tuples of streams, then $\# r$ denotes the length of the shortest stream in r; $d \,\&\, s$ denotes the result of applying $\&$ pointwisely to the components of d and s; $r \frown s$ and $r \sqsubseteq s$ are generalized in the same pointwise way.

If s, s' and r are stream tuples such that $s = s' \frown r$ then $s' \wr s = r$. For any stream s, and natural number n, $[s]^n$ denotes an n-tuple consisting of n copies of s.

A chain c is an infinite sequence of stream tuples $c_1, c_2, \ldots$ such that for all $j \geq 1$, $c_j \sqsubseteq c_{j+1}$. $\sqcup c$ denotes c's least upper bound. Since streams may be infinite such least upper bounds always exist.

A Boolean function $P : (D^\omega)^n \rightarrow \mathsf{B}$ is admissible iff whenever it yields true for each element of a chain, then it yields true for the least upper bound of the chain. We write $\mathsf{adm}(P)$ iff P is admissible.

A Boolean function $P : (D^\omega)^n \rightarrow \mathsf{B}$ is prefix-closed iff whenever it yields true for a stream tuple, then it also yields true for any prefix of this stream tuple.

A Boolean function $P : (D^\omega)^n \rightarrow \mathsf{B}$ is safe iff it is admissible and prefix-closed. We write $\mathsf{safe}(P)$ iff P is safe.

For formulas we need a substitution operator. Given a variable a and term t, then $P[^a_t]$ denotes the result of substituting t for every free occurrence of a in P. The operator is generalized in an obvious way in the case that a and t are lists.

A function $\tau \in (D^\omega)^n \rightarrow (D^\omega)^m$ is called a stream processing function iff it is prefix continuous:

$$\text{for all chains } c \text{ generated from } (D^\omega)^n : \;\; \tau(\sqcup c) = \sqcup\{\tau(c_j) | j \in \mathsf{N}^+\}.$$

That a function is prefix continuous implies first of all that the function's behavior for infinite inputs is completely determined by its behavior for finite inputs. Secondly, prefix continuity implies prefix monotonicity which basically means that if the input is increased then the output may at most be increased. Thus what has already been output can never be removed later on.

A stream processing function $\tau \in (D^\omega)^n \rightarrow (D^\omega)^m$ is pulse-driven iff:

$$\text{for all stream tuples } i \text{ in } (D^\omega)^n: \ \#i \neq \infty \Rightarrow \#\tau(i) < \#i.$$

That a function is pulse-driven means that the length of the shortest output stream is infinite or greater than the shortest input stream. This property is interesting in the context of feedback constructs because it guarantees that the least fixpoint is always infinite for infinite input streams. For a more detailed discussion, see [Bro92c].

The arrows $\rightarrow$, $\stackrel{c}{\rightarrow}$ and $\stackrel{cp}{\rightarrow}$ are used to tag domains of ordinary functions, domains of continuous functions, and domains of continuous, pulse-driven functions, respectively.

To model timeouts we need a special message $\surd$, called "tick". There are several ways to interpret streams with ticks. In this paper, all messages should be understood to represent the same time interval — the least observable time unit. $\surd$ occurs in a stream whenever no ordinary message is sent within a time unit. A stream or a stream tuple with occurrences of $\surd$'s are said to be timed. Similarly, a stream processing function is said to be timed when it operates on domains of timed streams. Observe that in the case of a timed, pulse-driven, stream processing function the output during the first $n+1$ time intervals is completely determined by the input during the first n time intervals.

For any stream or stream tuple i, $\diamond i$ denotes the result of removing all occurrences of $\surd$ in i. For example:

$$\diamond(a \,\&\, b \,\&\, \surd \,\&\, a \,\&\, s) = a \,\&\, b \,\&\, a \,\&\, \diamond\, s.$$

In the more theoretical parts of this paper, to avoid unnecessary complications, we distinguish between only two sets of messages, namely the set D denoting the set of all messages minus $\surd$, and T denoting $D \cup \{\surd\}$. However, the proposed formalism can easily be generalized to deal with general sorting, and this is exploited in the examples.

We use two additional functions in our examples: a filter function Ⓒ and a function $\propto$ removing consecutive repetitions. More formally, if A is a set of n-tuples of messages, d, e are n-tuples of messages, and r is an n-tuple of streams, then AⒸ and $\propto$ are stream processing functions such that the following axioms hold:

$$d \in A \Rightarrow A \text{Ⓒ} d \,\&\, r = d \,\&\, A \text{Ⓒ} r,$$
$$d \notin A \Rightarrow A \text{Ⓒ} d \,\&\, r = A \text{Ⓒ} r,$$

$$\propto(\langle d \rangle) = \langle d \rangle,$$
$$d = e \Rightarrow \propto(d \,\&\, e \,\&\, r) = \propto(d \,\&\, r),$$
$$d \neq e \Rightarrow \propto(d \,\&\, e \,\&\, r) = d \,\&\, \propto(e \,\&\, r).$$

When $A = \{d\}$ we write dⒸr instead of $\{d\}$Ⓒr.

3 Time Independent Specifications

A *ti-specification* of a component with n input channels and m output channels is written in the form

$$S\ (i\!:\!o) \equiv R,$$

where S is the specification's name; i and o are disjoint, repetition free lists of identifiers representing n respectively m streams; R is a formula with the elements of i and o as its only free variables. The formula R characterizes the input/output relation and is therefore referred to as such. The denotation of the specification S is the set of all timed stream processing functions, which fulfill R when time signals are abstracted away:

$$[\![\ S\ (i\!:\!o)\ ]\!] \overset{\mathsf{def}}{=} \{\tau \in (T^\omega)^n \xrightarrow{cp} (T^\omega)^m | \forall r \in (T^\omega)^n : R[^{i}_{\diamond r}\ ^{o}_{\diamond \tau(r)}]\}.$$

Strictly speaking, the following denotation

$$[\![\ S\ (i\!:\!o)\ ]\!] \overset{\mathsf{def}}{=} \{\tau \in (D^\omega)^n \xrightarrow{c} (D^\omega)^m | \forall r \in (D^\omega)^n : R[^{i}_{r}\ ^{o}_{\tau(r)}]\}.$$

is equivalent in the sense that it allows the same set of implementations. Thus timed functions are not really required in order to model ti-specifications. However, in this paper we stick to the former alternative because it is then easier to relate the different classes of specifications.

For any specification S, R_S represents its input/output relation.

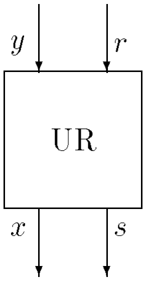

Figure 1: The Unreliable Receiver.

Example 1 Unreliable Receiver:
We specify a receiver UR that is unreliable in the sense that received data elements can be lost. The receiver has two input channels and two output channels, as pictured in Figure 1. It is assumed to communicate with a sender SND as indicated in Figure 4, on Page 11. The sender is formally specified in Example 3. The data elements to be received are input from the channel y.

The channel r models interference, which may cause a data element to be lost. When the n-th message input from r is a fail, it means that the n'th data element input from y is lost; on the other hand, if the n'th message input from r is an ok, it means that the n'th data element input from y is properly received.

It is assumed that infinitely many copies of ok are sent along r. As indicated in Figure 4, the agent SND has no access to r. This means that UR must forward the information input on r to the sender, so that the sender knows whether a data element sent along y was received or not. The output channel x is used for this purpose. Finally, any properly received data element is required to be output along s. More formally, given that $K = \{\mathsf{ok}, \mathsf{fail}\}$, UR is specified by:

$$\mathrm{UR}\ (y \in D^\omega, r \in K^\omega : x \in K^\omega, s \in D^\omega) \equiv$$

$$\#\mathsf{ok}\copyright r = \infty$$
$$\Rightarrow$$
$$\#x = \#y \wedge x \sqsubseteq r \wedge \#s = \#\mathsf{ok}\copyright(r|_{\#y}) \wedge (\epsilon, s) \sqsubseteq \{(\mathsf{ok}, d) | d \in D\}\copyright(r, y)$$

The antecedent states the environment assumption that infinitely many copies of ok are sent along r. The first conjunct of the consequence requires that x is of the same length as y; the second that x is a prefix of r; the third that s is of the same length as the stream of properly received data elements; the fourth that the stream of data elements sent along s is a prefix of the stream of properly received data elements. □

The operator $\overset{n\ m}{\otimes}$ is used to compose two specifications by connecting the n last output channels of the first specification with n first input channels of the second specification, and by connecting the m first output channels of the second specification with the m last input channels of the first specification. It can be seen as an operator for parallel composition. For example, if $\overset{n\ m}{\otimes}$ is used to compose $S_1\ (i, x : o, y)$ and $S_2\ (y, r : x, s)$, and y and x represent n and m channels, respectively, we get the network pictured in Figure 2. The channels represented by x and y are now hidden in the sense that they represent local channels.

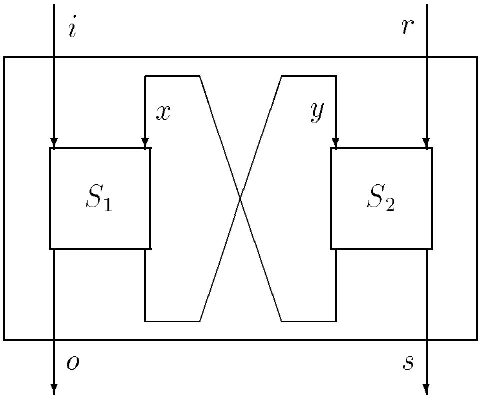

Figure 2: $S_1\ (i, x : o, y) \overset{n\ m}{\otimes} S_2\ (y, r : x, s)$.

More formally,

$$[\![\ S_1\ (i\,,x\!:\!o\,,y) \overset{n\ m}{\otimes} S_2\ (y\,,r\!:\!x\,,s)\ ]\!] \stackrel{\mathsf{def}}{=}$$
$$\{\tau_1 \overset{n\ m}{\otimes} \tau_2 | \tau_1 \in [\![\ S_1\ (i\,,x\!:\!o\,,y)\ ]\!] \wedge \tau_2 \in [\![\ S_2\ (y\,,r\!:\!x\,,s)\ ]\!]\},$$

where for any pair of timed stream tuples i and r, $(\tau_1 \overset{n\ m}{\otimes} \tau_2)(i, r) \stackrel{\mathsf{def}}{=} (o, s)$ iff (o, s) is the least fixpoint solution with respect to i and r. This is logically expressed by the following formula:

$$\begin{array}{l} \exists x, y : \forall o', y', x', s' : \\ \quad \tau_1(i, x) = (o, y) \wedge \tau_2(y, r) = (x, s)\ \wedge \qquad (1) \\ \quad \tau_1(i, x') = (o', y') \wedge \tau_2(y', r) = (x', s') \Rightarrow (o, y, x, s) \sqsubseteq (o', y', x', s'). \qquad (2) \end{array}$$

(1) requires (i, r, o, s) to represent a fixpoint; (2) requires this fixpoint to be the least.

When using $\overset{n\ m}{\otimes}$ to build networks of specifications, one will often experience that the operator needed is not $\overset{n\ m}{\otimes}$, but a slight modification of $\overset{n\ m}{\otimes}$, where for example the channels represented by x are not hidden, or the order of the channels represented by y and r is permuted in such a way that not all of the y-channels come before all the r-channels. The $\overset{n\ m}{\otimes}$-rules given below are valid for all these modified versions if the identifier lists in the specifications are updated accordingly. Instead of introducing operators and rules for each of these variations, we overload and use $\overset{n\ m}{\otimes}$ for all of them — with one exception: to simplify the discussions of the $\overset{n\ m}{\otimes}$-operator, we write

$$\mu^n S\ (i\,,x\!:\!o)$$

as a short-hand for

$$S_1\ (i\,,x\!:\!o\,,y) \overset{n\ n}{\otimes} S_2\ (y\!:\!x),$$

where $R_{S_1} \stackrel{\mathsf{def}}{=} R_S \wedge y = o$ and $R_{S_2} \stackrel{\mathsf{def}}{=} y = x$, as indicated in Figure 3. Clearly S_2 characterizes the identity component. When n and m are fixed by the context, we just write $\otimes$ and μ instead of $\overset{n\ m}{\otimes}$ and μ^n, respectively.

A specification $S_2\ (i\!:\!o)$ *refines* another specification $S_1\ (i\!:\!o)$, written

$$S_1\ (i\!:\!o) \leadsto S_2\ (i\!:\!o),$$

iff the behaviors specified by S_2 form a subset of the behaviors specified by S_1, formally:

$$[\![\ S_2\ (i\!:\!o)\ ]\!] \subseteq [\![\ S_1\ (i\!:\!o)\ ]\!].$$

Given a requirement specification $S\ (i\!:\!o)$, the goal of a system development is to construct a network of components C such that $S\ (i\!:\!o) \leadsto C$ holds. The refinement relation $\leadsto$ is reflexive, transitive and a congruence with respect to the composition operators. Hence,

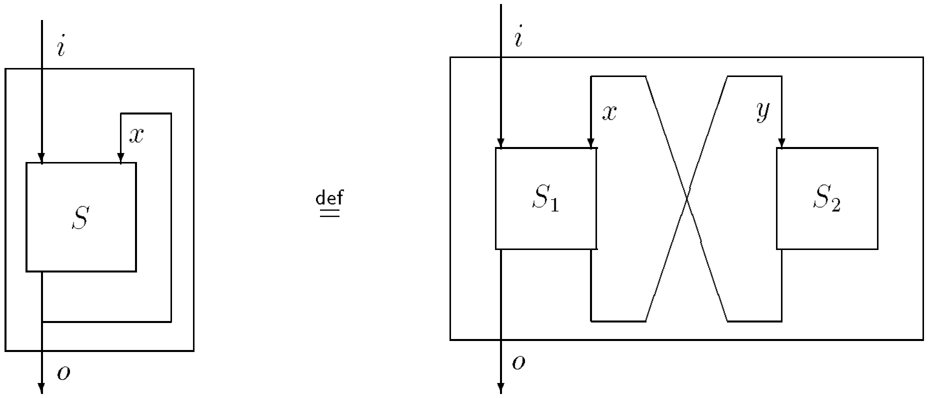

Figure 3: $\mu^n S\ (i\,, x\!:\!o) \stackrel{\mathsf{def}}{=} S_1\ (i\,, x\!:\!o\,, y) \overset{n\ n}{\otimes} S_2\ (y\!:\!x)$.

$\rightsquigarrow$ allows compositional system development: once a specification is decomposed into a network of subspecifications, each of these subspecifications can be further refined in isolation.

Based on this definition it is clear that a specification with an empty denotation refines any specification. Since specifications with empty denotations are inconsistent in the sense that there is no program that fulfills them, such refinements are undesirable. Thus, when formulating a specification $S\ (i\!:\!o)$, one should always check whether it is *consistent*. Consistency is logically expressed by:

$$\exists \tau \in (T^\omega)^n \xrightarrow{cp} (T^\omega)^m : \forall r \in (T^\omega)^n : R_S[\begin{smallmatrix} i & o \\ \diamond r & \diamond \tau(r) \end{smallmatrix}].$$

Or equivalently:

$$\exists \tau \in (D^\omega)^n \xrightarrow{c} (D^\omega)^m : \forall r \in (D^\omega)^n : R_S[\begin{smallmatrix} i & o \\ r & \tau(r) \end{smallmatrix}].$$

In other words, a specification is consistent iff its denotation is nonempty. Note, the consistency of a specification does not guarantee that there is a program that refines the specification. There are namely non-computable stream processing functions that cannot be expressed in any algorithmic language. It may therefore be argued that instead of proving consistency one should prove that a specification is implementable by a program. However, from a practical point of view, it is generally accepted that it does not make much sense to formally check the implementability of a specification. The reason is that to prove implementability it is often necessary to construct a program, which fulfills the specification, and that is of course the goal of the whole program refinement exercise.

We have explained what we mean by refinement. The next step is to explain how refinements can be proved correct. We give three rules for ti-specifications. The first one is a straightforward consequence rule:

Rule 1 :

$$\frac{R_{S_2} \Rightarrow R_{S_1}}{S_1\ (i:o) \rightsquigarrow S_2\ (i:o)}$$

Its correctness should be obvious. Rule 2 and 3, which allow for decomposition modulo the μ- and the $\otimes$-operators, respectively, are both based on fixpoint induction. In fact they are also closely related to the while-rule of Hoare-logic.

Rule 2 :

$$\frac{\begin{array}{l}\mathsf{adm}(\lambda x : I)\\ I[{}^{x}_{\epsilon}]\\ I \wedge R_{S_2} \Rightarrow I[{}^{x}_{o}]\\ I[{}^{x}_{o}] \wedge R_{S_2}[{}^{x}_{o}] \Rightarrow R_{S_1}\end{array}}{S_1\ (i:o) \rightsquigarrow \mu S_2\ (i\,, x:o)}$$

In this rule the stream tuples are named in accordance with Figure 3. The lambda notation in the first premise is used to express that I only has to be admissible with respect to x when i is kept constant. It is a well-known result that the least fixpoint of a feedback construct is equal to the least upper bound of the corresponding Kleene-chain [Kle52]. This is what fixpoint induction is based on, and this is also the idea behind Rule 2. The formula I can be thought of as an invariant in the sense of Hoare-logic and has the elements of i and x as its only free variables. The second premise implies that the invariant holds for the first element of the Kleene-chain. Then the third implies that the invariant holds for each element of the Kleene-chain, in which case it is a consequence of the first premise that it holds for the least upper bound of the Kleene-chain. Thus the conclusion can be deduced from the fourth premise.

The following rule

$$\frac{R_{S_2}[{}^{x}_{o}] \Rightarrow R_{S_1}}{S_1\ (i:o) \rightsquigarrow \mu S_2\ (i\,, x:o)}$$

is of course also sound. We refer to this rule as the degenerated version of Rule 2. One may ask, is it generally so that any decomposition provable by Rule 2 is also provable by its degenerated version? The answer is "no". With the degenerated version we can only prove properties that hold for all fixpoints. Properties which hold only for the least fixpoints can not be shown. In some sense the invariant of Rule 2 is used to characterize the least fixpoint solutions. We now look at a simple example where the inductive nature of Rule 2 is really needed.

Example 2 Elimination of Operationally Unfeasible Fixpoints:
Consider the following specification:

$$S_2\ (x:o) \equiv x = o.$$

It is clear that the result of applying the μ-operator to this specification is a network, which deadlocks in the sense that it never produces any output, i.e. a network which satisfies:

$$S_1\ (\,:o) \equiv o = \epsilon.$$

Mathematically expressed, it should be possible to prove that:

$$S_1\ (\,:o) \rightsquigarrow \mu S_2\ (x:o). \qquad (*)$$

However,

$$R_{S_2}\left[{}^{x}_{o}\right] \Rightarrow o = \epsilon$$

does not hold. This demonstrates that the degenerated version of Rule 2 is too weak. On the other hand, with

$$I \overset{\mathsf{def}}{=} x = \epsilon,$$

as invariant, it is straightforward to deduce $(*)$ using Rule 2.
□

The rule for the $\otimes$-operator is a straightforward generalization of Rule 2:

$$\begin{array}{l}
\textsf{Rule 3}: \\
\quad \textsf{adm}(\lambda x : I_1) \vee \textsf{adm}(\lambda y : I_2) \\
\quad I_1\left[{}^{x}_{\epsilon}\right] \vee I_2\left[{}^{y}_{\epsilon}\right] \\
\quad I_1 \wedge R_{S_1} \Rightarrow I_2 \\
\quad I_2 \wedge R_{S_2} \Rightarrow I_1 \\
\quad I_1 \wedge R_{S_1} \wedge I_2 \wedge R_{S_2} \Rightarrow R_S \\
\hline
\quad S\ (i\,, r:o\,, s) \rightsquigarrow S_1\ (i\,, x:o\,, y) \otimes S_2\ (y\,, r:x\,, s)
\end{array}$$

I_1 and I_2 are formulas with the elements of i, r, x and i, r, y as their only free variables, respectively. The third and the fourth premise implies that when one of the invariants holds for one element of the Kleene-chain then the other invariant holds for the next element of the Kleene-chain. The second premise then imply that both invariants hold for infinitely many elements of the Kleene-chain, in which case the first premise can be used to infer that at least one of the invariants hold for the least upper bound of the Kleene-chain. It then follows from premises three and four that both invariants hold for the least upper bound of the Kleene-chain. Thus, the conclusion can be deduced from premise five. See Proposition 1 in the appendix for a more detailed proof.

Example 3 Sender for the Unreliable Receiver:
The sender SND is supposed to hide the unreliability of UR, specified in Example 1. The unreliability can of course be hidden only under the assumption that infinitely many

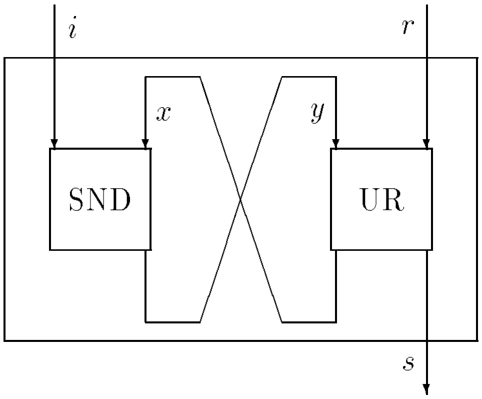

Figure 4: SND $(i\,, x : y) \otimes$ UR $(y\,, r : x\,, s)$.

ok's are sent along r. In fact, we can only hope that the resulting network satisfies the specification RR, which is specified as follows:

$$\mathrm{RR}\ (i \in D^{\omega}\,, r \in K^{\omega} : s \in D^{\omega}) \equiv \#\mathsf{ok}\copyright r = \infty \Rightarrow s = i$$

Clearly, SND must repeatedly send the same data element until an ok is received on x. Formally, SND is specified as follows:

$$\mathrm{SND}\ (i \in D^{\omega}\,, x \in K^{\omega} : y \in D^{\omega}) \equiv$$

$$\begin{array}{l} \mathsf{let} \\ \qquad n = \#\mathsf{ok}\copyright x \\ \mathsf{in} \\ \qquad n \leq \#i \\ \qquad\qquad \Rightarrow \\ \qquad \{(d, \mathsf{ok}) | d \in D\}\copyright(y, x) \sqsubseteq (i, \mathsf{ok}^{\infty}) \wedge \\ \qquad ((n = \#i \wedge \#y = \#x) \vee (n < \#i \wedge \#y = \#x + 1)) \end{array}$$

The antecedent states the environment assumption, namely that the number of ok's received on x is less than or equal to the number of data elements received on i. This is a sensible assumption, since UR only sends an acknowledgement along x for each properly received data element. The first conjunct of the consequence states that the stream of data elements sent along y, for which an ok is received on x, is a prefix of i. This means that for every received ok a fresh data element input from i is output. The second conjunct requires that either the length of i is equal to the number of ok's received on x and the length of y is equal to the length of x, or the length of i is greater than the number of ok's received on x and the length of y is one greater than the length of x. Operationally this means that the sender postpones outputting the next data element until it receives a positive or negative acknowledgement for the previous one.

To prove that the network SND $(i\,, x : y) \otimes$ UR $(y\,, r : x\,, s)$ is a refinement of RR $(i\,, r : s)$, it must be shown that

$$\mathrm{RR}\ (i\,, r : s) \rightsquigarrow \mathrm{SND}\ (i\,, x : y) \otimes \mathrm{UR}\ (y\,, r : x\,, s). \qquad (**)$$

Let

$$I_1 \stackrel{\mathsf{def}}{=} (\#\mathsf{ok}\copyright x \leq \#i \wedge x \sqsubseteq r) \vee \#\mathsf{ok}\copyright r \neq \infty,$$

$$I_2 \stackrel{\mathsf{def}}{=} \#\mathsf{ok}\copyright(r|_{\#y}) \leq \#i \vee \#\mathsf{ok}\copyright r \neq \infty,$$

then if

$$\mathsf{adm}(\lambda x : I_1) \vee \mathsf{adm}(\lambda y : I_2),$$

$$I_1[^x_\epsilon] \vee I_2[^y_\epsilon],$$

$$I_1 \wedge R_{\mathrm{SND}} \Rightarrow I_2,$$

$$I_2 \wedge R_{\mathrm{UR}} \Rightarrow I_1,$$

$$I_1 \wedge R_{\mathrm{SND}} \wedge I_2 \wedge R_{\mathrm{UR}} \Rightarrow R_{\mathrm{RR}},$$

it follows by Rule 3 that $(**)$ holds. The two first premises hold trivially. The remaining three can proved using straightforward predicate calculus.

□

Although Rule 2 and 3 are stronger than their degenerated versions, they are not as strong as desired. This will be discussed in detail in Section 6.

Nevertheless, a restricted completeness result may be stated. For any timed, pulse-driven, stream processing function $\tau \in (T^\omega)^n \stackrel{cp}{\to} (T^\omega)^m$, let

$$ti(\tau)\ (i : o)$$

be a ti-specification, whose input/output relation is characterized by the following equivalence:

$$R_{ti(\tau)} \Leftrightarrow \exists r' \in (T^\omega)^n : i = \diamond r' \wedge o = \diamond\tau(r').$$

Note, $ti(\tau)$ is the strongest ti-specification whose denotation contains τ.

For any specification $S\ (i : o)$ and timed, pulse-driven, stream processing function τ such that

$$[\![\ \mu\, ti(\tau)\ (i\,, x : o)\ ]\!] \subseteq [\![\ S\ (i : o)\ ]\!], \qquad (*)$$

it follows by Rule 2 that

$$S\ (i:o) \rightsquigarrow \mu\, ti(\tau)\ (i\,, x:o),$$

under the assumption that any valid formula of the base-logic (in which the premises are formulated) can be proved, and that the base-logic allows $ti(\tau)\ (i\,, x:o)$ and the invariant

$$\exists x' : x \sqsubseteq x' \wedge \mu\tau(i) = x',$$

where μ is overloaded to stream processing functions in a straightforward way, to be syntactically expressed (this corresponds to relative, semantic completeness with respect to deterministic components). The definition of the invariant implies that the three first premises of Rule 2 are valid, which means that the invariant is satisfied by the least fixpoint. Moreover, since the invariant holds for the least fixpoint only, it follows from the assumption $(*)$ that also the fourth premise is valid. A similar result holds in the case of Rule 3.

When writing ti-specifications one has to be very careful because of the strong monotonicity constraint imposed on their denotations. For example, consider the straightforward specification of fair merge (not necessarily order preserving) given below:

$$\text{RFM}\ (i\,, r:o) \equiv \forall d \in D : \#\{d\}©i + \#\{d\}©r = \#\{d\}©o.$$

This specification is inconsistent due to the monotonicity constraint. To see this, assume that there is a function τ which fulfills the specification. This means that

$$\tau(a^\infty, \epsilon) = a^\infty,$$
$$b©\tau(a^\infty, b^\infty) = b^\infty.$$

Clearly,

$$(a^\infty, \epsilon) \sqsubseteq (a^\infty, b^\infty),$$
$$\tau(a^\infty, \epsilon) \not\sqsubseteq \tau(a^\infty, b^\infty),$$

which means that τ is not monotonic and therefore not continuous. This contradicts the assumption. Thus the specification is inconsistent.

The cause of this problem is that a ti-specification makes no distinction between the behavior of a function for partial (finite) input and the behavior of a function for complete (infinite) input. More precisely, since

$$\diamond(a^\infty, \surd^\infty) = \diamond(a^\infty, \epsilon) = (a^\infty, \epsilon),$$

the specification above requires that

$$\diamond\tau(a^\infty, \surd^\infty) = \diamond\tau(a^\infty, \epsilon) = a^\infty,$$

although strictly speaking we only want to specify that

$$\diamond\tau(a^{\infty}, \epsilon) \sqsubseteq a^{\infty},$$
$$\diamond\tau(a^{\infty}, \surd^{\infty}) = a^{\infty}.$$

Thus because we are not able to distinguish complete, infinite input streams with only finitely many messages different from $\surd$, from finite, incomplete inputs, when time-ticks are abstracted away, our requirements become too strong.

This observation was made already in [Par83]. In [Bro89] it led to the proposal of so-called input choice specifications. In the next section we advocate a slightly different approach with a semantically simpler foundation.

4 Weakly Time Dependent Specifications

A wtd-specification of a component with n input channels and m output channels is written in the form

$$S\ \langle i : o \rangle \equiv R,$$

where S is the specification's name; i and o are disjoint, repetition free lists of identifiers representing n respectively m streams; R is a formula with the elements of i and o as its only free variables. As before R characterizes the relation between the input and output streams. Syntactically, a wtd-specification differs from a ti-specification in that the brackets $\langle\rangle$ are used instead of $()$ to embrace the lists of input/output identifiers.

The denotation of the specification S is the set of all timed, pulse-driven, stream processing functions which fulfill R when time signals are abstracted away and only complete inputs are considered:

$$[\![\ S\ \langle i : o \rangle\ ]\!] \stackrel{\mathsf{def}}{=} \{\tau \in (T^{\omega})^{n} \xrightarrow{cp} (T^{\omega})^{m} | \forall r \in (T^{\infty})^{n} : R[^{i}_{\diamond r}\ ^{o}_{\diamond\tau(r)}]\}$$

Thus in contrast to a ti-specification, a wtd-specification constrains the behavior for complete inputs (infinite inputs at the semantic level[1]). As before, for any wtd-specification S, R_S denotes its input/output relation.

A wtd-specification $S\ \langle i : o \rangle$ is consistent iff

$$\exists \tau \in (T^{\omega})^{n} \xrightarrow{cp} (T^{\omega})^{m} : \forall r \in (T^{\infty})^{n} : R_S[^{i}_{\diamond r}\ ^{o}_{\diamond\tau(r)}].$$

Since, as in the time independent case, the denotation is a set of timed, pulse-driven, stream processing functions, the composition operator $\otimes$ and the refinement relation $\rightsquigarrow$ can be defined in the same way as earlier.

As shown in the next four examples, weakly time dependent components can be specified

[1] Note that although the streams are infinite they may have only finitely many occurrences of messages different from $\surd$.

in a very elegant way.

Example 4 Fair Merge (with Reorderings):
The wtd-specification

$$\text{RFM}\ \langle i \in D^\omega, r \in D^\omega : o \in D^\omega\rangle \equiv \forall d \in D : \#\{d\}©i + \#\{d\}©r = \#\{d\}©o$$

specifies a component performing a (not necessarily order preserving) fair merge. Since the specification constrains complete inputs only (infinite streams at the semantic level), the monotonicity problem of the previous section does not apply here. □

Example 5 Fair Merge (without Reorderings):
A component, which not only outputs a fair merge of the streams of messages received on its two input channels, but also preserves the ordering of the messages with respect to the different input channels, is specified below:

$$\text{FM}\ \langle i \in D^\omega, r \in D^\omega : o \in D^\omega\rangle \equiv \exists p \in \{1,2\}^\omega : split_1(o,p) = i \wedge split_2(o,p) = r$$

where $split_j \in D^\omega \times \{1,2\}^\omega \xrightarrow{c} D^\omega$ is an auxiliary function which, based on a oracle (its second argument), can be used to extract the stream of messages received on one of the input channels:

$$j = b \Rightarrow split_j(a \,\&\, o, b \,\&\, p) = a \,\&\, split_j(o,p),$$
$$j \neq b \Rightarrow split_j(a \,\&\, o, b \,\&\, p) = split_j(o,p).$$

□

Example 6 Busy Sender:
A component, which sends requests, represented by the signal ?, along its output channel until it receives a message on its input channel and then feeds this message along its output channel, can be specified as follows:

$$\text{BS}\ \langle i \in D^\omega : o \in (D \cup \{?\})^\omega\rangle \equiv (i = \epsilon \wedge o = ?^\infty) \vee \exists n \in \mathsf{N} : o = ?^n \frown \mathsf{ft}(i)$$

If no message is eventually received, then finally infinitely many requests are generated as output. □

Example 7 Arbiter:
An arbiter is a component that reproduces its input data and in addition adds an infinite number of tokens, here represented by •, to its output stream. More formally:

$$\text{AR}\ \langle i \in D^\omega : o \in (D \cup \{\bullet\})^\omega\rangle \equiv D©o = i \wedge \# \bullet ©o = \infty$$

It is assumed that • is not an element of D.
□

The rules for ti-specifications can be generalized to deal with wtd-specifications. The consequence rule is unchanged if we adapt the brackets:

Rule 4 :

$$\frac{R_{S_2} \Rightarrow R_{S_1}}{S_1 \ \langle i : o \rangle \rightsquigarrow S_2 \ \langle i : o \rangle}$$

The modifications of the rules for the μ- and $\otimes$-operators are less trivial. The reason is that wtd-specifications constrain the behavior for complete inputs (infinite inputs at the semantic level) only, which means that it is no longer straightforward to carry out the induction over the Kleene-chain. We first show that Rule 2, with $\langle\rangle$ substituted for $()$ in the conclusion, is unsound for wtd-specifications.

Example 8 :
Consider the following wtd-specification

$$S_2 \ \langle x \in \mathsf{N}^\omega : o \in \mathsf{N}^\omega \rangle \equiv o = 1 \,\&\, 2^\infty \vee (x \neq \epsilon \wedge o = 1 \,\&\, x).$$

Let

$$I \overset{\mathsf{def}}{=} x = \epsilon \vee \exists n \in \mathsf{N} : x = 1^n \frown 2^\infty.$$

It holds that

$$\mathsf{adm}(\lambda x : I),$$

$$I[{}^x_\epsilon],$$

$$I \wedge R_{S_2} \Rightarrow I[{}^x_o].$$

$I[{}^x_o] \wedge R_{S_2}[{}^x_o]$ implies

$$o = 1 \,\&\, 2^\infty.$$

Thus we may use Rule 2 to prove that

$$S_1 \ \langle : o \rangle \rightsquigarrow S_2 \ \langle x : o \rangle,$$

where $R_{S_1} \overset{\mathsf{def}}{=} o = 1 \,\&\, 2^\infty$. To see that this deduction is unsound, note there is a $\tau \in [\![\ S_2 \ \langle x : o \rangle \ ]\!]$ such that

$$\tau(\epsilon) = \langle 1 \rangle,$$
$$\tau(\surd \,\&\, r) = 1 \,\&\, 2^\infty,$$
$$a \neq \surd \Rightarrow \tau(a \,\&\, r) = 1 \,\&\, a \,\&\, r.$$

Since τ is pulse-driven, it has a unique, infinite fixpoint, namely

$$\tau(1^\infty) = 1^\infty.$$

Unfortunately, this fixpoint does not satisfy R_{S_1}, in which case it follows that Rule 2 is unsound for wtd-specifications.

□

We now characterize a slightly modified version of fixpoint induction. Given a wtd-specification $S_2 \; \langle i\,, x : o \rangle$ with two input and one output channel. Assume that $\tau \in [\![\; S_2 \; \langle i\,, x : o \rangle \;]\!]$. Let t be the infinite sequence of infinite streams $t_1, t_2, \; \ldots$ such that:

$$t_1 = \surd^\infty,$$
$$t_{j+1} = \tau(r, t_j),$$

for some infinite, timed stream r. For the same input r, let s be τ's Kleene-chain, i.e.:

$$s_1 = \epsilon,$$
$$s_{j+1} = \tau(r, s_j).$$

Although t is not (normally) a chain, we refer to t as τ's pseudo-chain with respect to r. Since τ is pulse-driven, and r is infinite, the equation

$$\tau(r, x) = x$$

has a unique, infinite solution, and this solution is according to Kleene's theorem [Kle52] equal to the least upper bound of the Kleene-chain:

$$\tau(r, \sqcup s) = \sqcup s.$$

Since $s_1 \sqsubseteq t_1$ and τ is monotonic, it follows by induction on j that

$$s_j \sqsubseteq t_j.$$

The monotonicity of $\diamond$ implies

$$\diamond s_j \sqsubseteq \diamond t_j. \qquad (*)$$

Let I be a formula with free variables i and x such that $\lambda x : I$ is safe (which means that $\lambda x : I$ is prefixed-closed). Then if for all j

$$I[^x_{\diamond t_j}], \qquad (**)$$

it follows from $(*)$ and the fact that $\lambda x : I$ is safe

$$I[^{x}_{\diamond s_j}].$$

Since $\diamond$ is continuous and $\lambda x : I$ is admissible, we also have that

$$I[^{x}_{\diamond \sqcup s}]. \qquad (***)$$

Thus $\lambda x : I$ holds for τ's least fixpoint solution, when all time ticks are removed. Consequently, to make sure that $(***)$ holds, it is enough to show that $(**)$ holds for each element of the pseudo-chain, when time-ticks are removed. Since

$$I[^{x}_{\epsilon}],$$

$$I \wedge R_{S_2} \Rightarrow I[^{x}_{o}],$$

implies

$$I[^{i}_{\diamond t_1}],$$

$$I[^{i}_{\diamond t_j}] \Rightarrow I[^{i}_{\diamond t_{j+1}}],$$

it follows by a slight generalization of the argumentation above that the following rule is sound:

$$\begin{array}{l}\textsf{Rule 5 :}\\ \dfrac{\begin{array}{l}\textsf{safe}(\lambda x : I)\\ I[^{x}_{\epsilon}]\\ I \wedge R_{S_2} \Rightarrow I[^{x}_{o}]\\ I[^{x}_{o}] \wedge R_{S_2}[^{x}_{o}] \Rightarrow R_{S_1}\end{array}}{S_1 \ \langle i : o \rangle \rightsquigarrow \mu S_2 \ \langle i \, , x : o \rangle}\end{array}$$

Recall that $\textsf{safe}(P)$ expresses that P is safe.

An interesting question at this point is of course: how strong is Rule 5? We start by showing that the invariant is really needed — needed in the sense that the degenerated version

$$\frac{R_{S_2}[^{x}_{o}] \Rightarrow R_{S_1}}{S_1 \ \langle i : o \rangle \rightsquigarrow \mu S_2 \ \langle i \, , x : o \rangle}$$

is strictly weaker.

Example 9 Elimination of Operationally Unfeasible Fixpoints:
Given the wtd-specification

$$S\ \langle x \in \mathsf{N}^\omega : o \in \mathsf{N}^\omega \rangle \equiv o = 1 \,\&\, x \vee (\mathsf{ft}(x) = 1 \wedge o = x).$$

From $R_S[^x_o]$ we can deduce only that

$$\mathsf{ft}(o) = 1.$$

Let

$$I \overset{\mathsf{def}}{=} x \in \{1\}^\omega.$$

Using Rule 5 we may deduce that

$$S_1\ \langle : o \rangle \rightsquigarrow \mu\, S\ \langle x : o \rangle,$$

if

$$R_{S_1} \Leftrightarrow \# o \geq 1 \wedge o \in \{1\}^\omega.$$

□

The rule for the ⊗-operator can be restated in a similar way:

Rule 6 :

$$\begin{array}{l} \mathsf{safe}(\lambda x : I_1) \vee \mathsf{safe}(\lambda y : I_2) \\ I_1[^x_\epsilon] \vee I_2[^y_\epsilon] \\ I_1 \wedge R_{S_1} \Rightarrow I_2 \\ I_2 \wedge R_{S_2} \Rightarrow I_1 \\ I_1 \wedge R_{S_1} \wedge I_2 \wedge R_{S_2} \Rightarrow R_S \\ \hline S\ \langle i\,, r : o\,, s \rangle \rightsquigarrow S_1\ \langle i\,, x : o\,, y \rangle \otimes S_2\ \langle y\,, r : x\,, s \rangle \end{array}$$

I_1 and I_2 are formulas with the elements of i, r, x and i, r, y as free variables, respectively. For any $\tau_1 \otimes \tau_2 \in [\![\ S_1 \otimes S_2\ ]\!]$, the second, third and fourth premise imply that both I_1 and I_2 hold for infinitely many elements of $\tau_1 \otimes \tau_2$'s pseudo-chain. From the first premise it then follows that at least one of the invariants holds for infinitely many elements of the corresponding Kleene-chain, and therefore also for the least upper bound of the Kleene-chain. Since this least upper bound is infinite (at the semantic level) it follows from premise three and four that both invariants hold for the least upper bound of the Kleene-chain. Thus the conclusion can be deduced from premise five. See Proposition 2 in the appendix for a more detailed proof.

Rule 5 and 6 are quite useful, but they do not satisfy a completeness result similar to that for ti-specifications. We now discuss this in more detail.

For any timed, pulse-driven, stream processing function $\tau \in (T^\omega)^n \overset{cp}{\rightarrow} (T^\omega)^m$, let

$$wtd(\tau)\ \langle i : o \rangle$$

be a wtd-specification, whose input/output relation is characterized by the following equivalence:

$$R_{wtd(\tau)} \Leftrightarrow \exists i' \in (T^{\infty})^n : i = \diamond i' \wedge o = \diamond \tau(i').$$

Note, $wtd(\tau)$ is the strongest wtd-specification whose denotation contains τ. For example, with respect to the specification S of Example 9, it holds that if

$$\begin{array}{l} \tau(\epsilon) = 1 \\ a \neq 1 \Rightarrow \tau(a \,\&\, r) = 1 \,\&\, a \,\&\, r \\ \tau(1 \,\&\, r) = 1 \,\&\, \surd \,\&\, r, \end{array}$$

then $\tau \in [\![\ S\ \langle x : o \rangle \ ]\!]$. Thus:

$$R_{wtd(\tau)} \Rightarrow R_S[^x_i].$$

In fact, since per definition

$$\begin{array}{l} \tau(\surd \,\&\, r) = 1 \,\&\, \surd \,\&\, r, \\ \tau(1 \,\&\, r) = 1 \,\&\, \surd \,\&\, r, \end{array}$$

it also follows that

$$R_S[^x_i] \Rightarrow R_{wtd(\tau)}.$$

The set $[\![\ wtd(\tau)\ \langle i : o \rangle \ ]\!]$ is of course not unary, and its elements may have different fix-points. In Example 9 we used Rule 5 to deduce that o is an element of $\{1\}^{\omega}$, whose length is greater than or equal to 1. As a matter of fact, for each such output solution 1^n, there is a corresponding function $\tau_n \in [\![\ wtd(\tau)\ \langle i : o \rangle \ ]\!]$, such that

$$\tau_n(r) = r \Rightarrow \diamond r = 1^n.$$

For example, τ_n may be defined as follows:

$$\begin{array}{l} \tau_n(\epsilon) = 1, \\ k < n \Rightarrow \tau_n(1^k) = 1^{k+1}, \\ \tau_n(1^n \frown r) = 1^n \frown \surd \,\&\, r, \\ k < n \wedge a \neq 1 \Rightarrow \tau_n(1^k \frown a \,\&\, r) = 1^{k+1} \frown a \,\&\, r. \end{array}$$

To see that a completeness result similar to that for ti-specifications does not hold for wtd-specifications with respect to the rules introduced above, consider the following example:

Example 10 :
Given $K = \{1, 2, \surd\}$. Let $\tau \in K^\omega \overset{cp}{\to} K^\omega$ be a function such that:

$$\begin{aligned}
&\tau(\epsilon) = \langle 1 \rangle,\\
&\tau(\langle 1 \rangle) = \langle 1, 1 \rangle,\\
&\tau(\surd \,\&\, in) = 1 \,\&\, 2^\infty,\\
&\tau(1 \,\&\, a \,\&\, in) = 1 \,\&\, 1 \,\&\, (\mathsf{if}\ a = 1\ \mathsf{then}\ \surd\ \mathsf{else}\ a) \,\&\, in,\\
&\tau(2 \,\&\, in) = 1 \,\&\, 2 \,\&\, in.
\end{aligned}$$

$R_{wtd(\tau)}$ is equivalent to

$$o = 1 \,\&\, 2^\infty \vee (\exists i' : i = 1 \,\&\, 1 \,\&\, i' \wedge o = i) \vee (i \neq \epsilon \wedge o = 1 \,\&\, i).$$

Let

$$I \overset{\mathsf{def}}{=} \exists n \in \mathsf{N}^+ \cup \{\infty\} : x \sqsubseteq 1^n \frown 2^\infty.$$

Then I is the strongest formula such that

$$\mathsf{safe}(\lambda x : I),$$

$$I[{}^x_\epsilon],$$

$$I \wedge R_{wtd(\tau)} \Rightarrow I[{}^x_o].$$

Moreover, $I[{}^x_o] \wedge R_{wtd(\tau)}[{}^x_o]$ implies

$$o = 1 \,\&\, 2^\infty \vee \exists z \in \{1\}^\omega : \exists y \in \{2\}^\omega : o = 1 \,\&\, 1 \,\&\, z \frown y.$$

Unfortunately, this formula is too weak in the sense that there are solutions for which there are no corresponding functions in $[\![\ wtd(\tau)\ \langle i\!:\!o\rangle\ ]\!]$. For example, there is no $\tau' \in [\![\ wtd(\tau)\ \langle i\!:\!o\rangle\ ]\!]$ such that

$$\tau'(r) = r \Rightarrow \diamond r = \langle 1, 1, 2 \rangle$$

To see that, let r' be a prefix of r such that $\diamond r' = \langle 1, 1 \rangle$. Since r is the fixpoint of τ', it follows that r must be reachable from $r' \frown \surd^\infty$ in the sense that

$$\langle 1, 1, 2 \rangle \sqsubseteq \diamond \tau'(r' \frown \surd^\infty).$$

However, such a computation is not allowed by $R_{wtd(\tau)}$. Thus, Rule 5 is too weak in the sense that it does not allow us to remove all "solutions" for which there are no corresponding functions in $[\![\ wtd(\tau)\ \langle i\!:\!o\rangle\ ]\!]$.
□

In Rule 5, the task of I is to characterize the elements of the Kleene-chains with their corresponding least upper bounds. Since for any timed, pulse-driven, stream processing function τ, it holds that

$$r_j \sqsubseteq r_{j+1} \sqsubseteq \tau(s, r_j \frown \surd^\infty),$$

if r is τ's Kleene-chain with respect to the complete input s, we may strengthen Rule 5 as follows:

Rule 7 :

$$\begin{array}{l}
(\forall j : I[^{x}_{c_j}] \wedge \exists o : R_{S_2}[^{x}_{c_j}] \wedge c_{j+1} \sqsubseteq o) \Rightarrow I[^{x}_{\sqcup c}] \\
I[^{x}_{\epsilon}] \\
I \wedge x \sqsubset x' \sqsubseteq o \wedge R_{S_2} \Rightarrow I[^{x}_{x'}] \\
I[^{x}_{o}] \wedge R_{S_2}[^{x}_{o}] \Rightarrow R_{S_1} \\
\hline
S_1\ \langle i:o\rangle \rightsquigarrow \mu S_2\ \langle i\,, x:o\rangle
\end{array}$$

c varies over chains, and I is a formula with the elements of i and x as its only free variables. Rule 7 solves the problem of Example 10, if we choose

$$x \sqsubseteq 1 \frown 2^\infty \vee x \in \{1\}^\omega$$

as the invariant I.

For any timed, pulse-driven, stream processing function τ, there is a function $\tau' \in wtd(\tau)$, which is identical to τ for complete inputs (and therefore has exactly the same fixpoints as τ for complete inputs), and whose Kleene-chain consists of only finite stream tuples (each stream in each tuple is finite). For example, we may define $\tau'(i)$ as $\tau(i)|_{\#i+1}$. Due to this fact we may weaken the premises of Rule 7 even further:

Rule 8 :

$$\begin{array}{l}
(\forall j : I[^{x}_{c_j}] \wedge c_j \in (D^*)^m \wedge \exists o : R_{S_2}[^{x}_{c_j}] \wedge c_{j+1} \sqsubseteq o) \Rightarrow I[^{x}_{\sqcup c}] \\
I[^{x}_{\epsilon}] \\
I \wedge x \in (D^*)^m \wedge x \sqsubset x' \sqsubseteq o \wedge R_{S_2} \Rightarrow I[^{x}_{x'}] \\
I[^{x}_{o}] \wedge R_{S_2}[^{x}_{o}] \Rightarrow R_{S_1} \\
\hline
S_1\ \langle i:o\rangle \rightsquigarrow \mu S_2\ \langle i\,, x:o\rangle
\end{array}$$

It is here assumed that x has m elements. See Proposition 3 in the appendix for a soundness proof. Rule 6 can be strengthened accordingly.

We can now state a completeness result similar to that for ti-specifications. For any specification $S\ \langle i:o\rangle$ and timed, pulse-driven, stream processing function τ such that

$$[\![\ \mu\, wtd(\tau)\ \langle i\,, x:o\rangle\ ]\!] \subseteq [\![\ S\ \langle i:o\rangle\ ]\!], \qquad (*)$$

it follows by Rule 8 that

$$S \ \langle i : o \rangle \leadsto \mu \, wtd(\tau) \ \langle i \, , x : o \rangle,$$

under the assumption that any valid formula of the base-logic (in which the premises are formulated) can be proved, and that the base-logic allows $wtd(\tau) \ \langle i : o \rangle$ and the strongest formula I, such that the three first premises of Rule 8 hold, to be syntactically expressed (this corresponds to relative, semantic completeness with respect to a restricted set of components). See Proposition 4 of the appendix for a proof.

The following conversion rule is also useful:

Rule 9 :

$$\begin{array}{l} R_{S_2} \Rightarrow R_{S_1} \\ \hline S_1 \ \langle i : o \rangle \leadsto S_2 \ (i : o) \end{array}$$

Due to this rule, ti-specifications and wtd-specifications can be used side by side in system developments. Observe that the inverse of Rule 9, where a ti-specification is replaced by a wtd-specification, is invalid. For example, as explained in the previous section, the specification RFM of Example 4 becomes inconsistent when it is assigned the denotation of a ti-specification. In fact, not even if S is deterministic, does it generally hold that

$$S \ (i : o) \leadsto S \ \langle i : o \rangle.$$

For example, if $R_S \overset{\mathsf{def}}{=} i = o$, then the function τ, where

$$\begin{array}{l} \#i < 10 \Rightarrow \tau(i) = \surd^{\#i+1}, \\ \#i \geq 10 \Rightarrow \tau(i) = \surd^{10} \frown i \frown \surd^{\#i}, \end{array}$$

is an element of $\llbracket \ S \ \langle i : o \rangle \ \rrbracket$, but not of $\llbracket \ S \ (i : o) \ \rrbracket$. The reason is that a wtd-specification does not constrain the behavior for partial (finite at the semantic level) inputs. However, the following rule is sound:

Rule 10 :

$$\begin{array}{l} \#i \neq \infty \Rightarrow R_{S_1} \\ R_{S_2} \Rightarrow R_{S_1} \\ \hline S_1 \ (i : o) \leadsto S_2 \ \langle i : o \rangle \end{array}$$

The first premise makes sure that S_1 does not constrain the behavior for finite inputs.

Example 11 Alternating Bit Transmission:
As in Example 3 we specify an unreliable receiver UR and a corresponding sender SND and show that they behave correctly when composed as in Figure 5. The communication between the two components is based on alternating bit transmission. This means that for each data element SND sends along z, a bit is sent along y. Clearly SND is required to send the same data element accompanied by the very same bit until this data element eventually is properly received by UR, in which case the actual bit is sent back along

x. The receiver UR looses input pairs (from z and y) in accordance with r. Thus the messages ok and fail determine whether a data element is lost or not in the same way as in Example 1. Properly received input pairs, whose bit component is equal to the bit component of the last properly received input pair, are ignored in the sense that no output is produced. The reason is of course that this pair has been sent by SND before the acknowledgement bit for the earlier pair was received. All other properly received input pairs are sent on along x (the bit) and s (the data element).

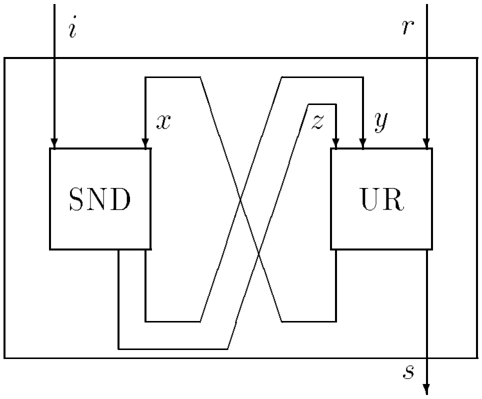

Figure 5: Alternating bit network.

The sender SND is formally specified by:

$$\mathrm{SND}\ \langle i \in D^{\omega}, x \in M^{\omega} : z \in D^{\omega}, y \in M^{\omega} \rangle \equiv$$

$$\begin{array}{ll}
\textsf{let} & \\
& (z', y') = \propto(z, y) \\
\textsf{in} & \\
& \#x \le \#i \\
& \quad \Rightarrow \\
& \#y = \#z \wedge y' = \propto(y) \wedge z' \sqsubseteq i \wedge \\
& (\#x \ne \#i \Rightarrow \#z = \infty \wedge \#z' = \#x + 1) \wedge \\
& (\#x = \#i \Rightarrow \#z' = \#i)
\end{array}$$

M denotes $\{0, 1\}$. The antecedent states the environment assumption, namely that the length of x is less than the length of i. The first conjunct of the consequence requires y and z to be of the same length; the second makes sure that two different consecutive data elements sent along z are assigned different bits; the third requires that when repetitions are ignored then the stream of data elements sent along z is a prefix of the stream of data elements received on i; the fourth requires that if the length of x is not equal to the length of i, then the length of z is infinite and the number of data elements sent along z, when repetitions are ignored, is equal to the length of x plus 1 — which basically means that the same data element is sent infinitely many times if no acknowledgement is received from the receiver; the fifth requires that if the length of x is equal to the length of i, then the number of data elements sent along z, when repetitions are ignored, is equal to the length of i.

The behavior of UR is characterized by:

$$\mathrm{UR}\ \langle z \in D^\omega\,, y \in M^\omega\,, r \in K^\omega : x \in M^\omega\,, s \in D^\omega\rangle \equiv$$

$$\begin{array}{ll} \mathsf{let} & \\ & (z', y', r') = \{(d, m, \mathsf{ok}) | d \in D, m \in M\} \copyright (z, y, r) \\ \mathsf{in} & \\ & (s, x) = \propto(z', y') \end{array}$$

K denotes $\{\mathsf{ok}, \mathsf{fail}\}$. The streams of messages sent along x and s are required to be equal to the streams of properly received input pairs when repetitions are ignored.

Given

$$\mathrm{RR}\ \langle i \in D^\omega\,, r \in K^\omega : s \in D^\omega\rangle \equiv \#\mathsf{ok} \copyright r = \infty \Rightarrow s = i,$$

we want to prove

$$\mathrm{RR}\ \langle i\,, r : s\rangle \rightsquigarrow \mathrm{SND}\ \langle i\,, x : z\,, y\rangle \otimes \mathrm{UR}\ \langle z\,, y\,, r : x\,, s\rangle. \qquad (*)$$

Let

$$I_1 \overset{\mathsf{def}}{=} \#x \leq \#i,$$

$$I_2 \overset{\mathsf{def}}{=} \#\propto(z, y) \leq \#i,$$

then $(*)$ follows by Rule 6 since it is straightforward to show that

$$\mathsf{safe}(\lambda x : I_1) \vee \mathsf{safe}(\lambda z : \lambda y : I_2),$$

$$I_1[{}^x_\epsilon] \vee I_2[{}^z_\epsilon\ {}^y_\epsilon],$$

$$I_1 \wedge R_{\mathrm{SND}} \Rightarrow I_2,$$

$$I_2 \wedge R_{\mathrm{UR}} \Rightarrow I_1,$$

$$I_1 \wedge R_{\mathrm{SND}} \wedge I_2 \wedge R_{\mathrm{UR}} \Rightarrow R_S.$$

Observe that only SND characterizes a weakly time dependent component. UR can also be stated as a ti-specification. In fact, after having shown that the network SND $\langle i\,, x : z\,, y\rangle \otimes$ UR $\langle z\,, y\,, r : x\,, s\rangle$ behaves as desired, we may use Rule 9 to translate UR into a ti-specification and thereafter complete the development of UR using the rules for ti-specifications only.

□

5 Strongly Time Dependent Specifications

An std-specification of a component with n input channels and m output channels is written in the form

$$S\ \{i:o\} \equiv R,$$

where S is the specification's name; i and o are disjoint, repetition free lists of identifiers representing n respectively m streams; R is a formula with the elements of i and o as its only free variables. Yet another pair of brackets $\{\}$ is employed to distinguish std-specifications from ti- and wtd-specifications. The denotation of the specification S is the set of all timed, pulse-driven, stream processing functions which fulfill R when only complete (infinite) inputs are considered:

$$[\![\ S\ \{i:o\}\ ]\!] \overset{\mathsf{def}}{=} \{\tau \in (T^\omega)^n \overset{cp}{\to} (T^\omega)^m | \forall i \in (T^\infty)^n : R[{}^{o}_{\tau(i)}]\}. \qquad (\dagger)$$

Observe that in this case the time signals are not abstracted away. Thus, time signals may occur explicitly in R.

As for wtd-specifications, only the behavior for complete, infinite inputs is constrained. Nevertheless, the expressiveness of an std-specification would not have been reduced if we had used the following denotation:

$$[\![\ S\ \{i:o\}\ ]\!] \overset{\mathsf{def}}{=} \{\tau \in (T^\omega)^n \overset{cp}{\to} (T^\omega)^m | \forall i \in (T^\omega)^n : R[{}^{o}_{\tau(i)}]\}. \qquad (\ddagger)$$

The reason is that in the case of std-specifications there is no time abstraction, which means that, at the syntactic level, incomplete (finite) inputs can always be distinguished from complete (infinite) inputs. However, from a practical point of view, it is not clear that the latter denotation (‡) offers any advantages. We therefore stick with the former (†) although we also refer to (‡) later on.

Example 12 Timer for Time-outs:
We specify a simple timer for time-outs. It has one input and one output channel. Whenever it receives a set timer message $\mathsf{set}(n)$, where n is a natural number, and it is not reset by a reset message rst, it responds by sending the timeout signal Γ after n time-units. Set timer messages received before the Γ for the previous set timer message has been sent are simply ignored.

Given $K = \{\mathsf{set}(n) | n \in \mathsf{N}^+\} \cup \{\mathsf{rst}, \surd\}$ and $M = \{\Gamma, \surd\}$, we may specify the timer as follows:

$\mathrm{TT}\ \{i \in K^\omega : o \in M^\omega\} \equiv$

```
$\exists \tau \in \mathsf{N} \to K^\omega \overset{c}{\to} M^\omega : o = \surd \,\&\, \tau(0)(i)$
where $\forall n, m \in \mathsf{N} : \forall i' \in K^\omega$ :
    $\tau(0)(\epsilon) = \epsilon \wedge$
    $\tau(n)(\surd \,\&\, i') =$
        if $n = 0$ then $\surd \,\&\, \tau(0)(i')$
        else if  $n = 1$ then $\Gamma \,\&\, \tau(0)(i')$
        else $\surd \,\&\, \tau(n-1)(i') \wedge$
    $\tau(n)(\mathsf{rst} \,\&\, i') = \surd \,\&\, \tau(0)(i') \wedge$
    $\tau(n)(\mathsf{set}(m) \,\&\, i') =$ if $n = 0$ then $\tau(m)(\surd \,\&\, i')$ else $\tau(n)(\surd \,\&\, i')$
```

The existentially quantified function τ, which for each natural number n returns a timed stream processing function $\tau(n)$, characterizes the relation between the input- and the output-stream. It has a "state parameter" n, that is either equal to 0, in which case the timer is in its idle state, or ≥ 1, in which case n represents the number of time-units the next time-signal Γ is to be delayed.

□

Any wtd-specification can also be expressed as an std-specification. Given the wtd-specification

$$S\ \langle i : o \rangle \equiv R,$$

then

$$S\ \{r : s\} \equiv R[^{i}_{\diamond r}\ ^{o}_{\diamond s}]$$

is an equivalent std-specification. In general, the same does not hold for ti-specifications. The reason is the way ti-specifications constrain the behavior for partial input. Let τ be a timed, pulse-driven, stream processing function, and assume that i and o are two complete (infinite), timed stream tuples such that

$$\tau(i) = o. \qquad (*)$$

Unfortunately, for any finite prefix $i' \sqsubset i$, from $(*)$ alone we can only deduce that $o|_{\#i'+1} \sqsubseteq \tau(i')$. Thus although we operate with timed stream tuples, the behavior for finite inputs (partial inputs) is only partly fixed from the behavior for infinite inputs. This can be avoided by strengthening the pulse-driveness constraint. However, this is hardly any improvement from a pragmatic point of view — on the contrary, such additional constraints may in some cases lead to more complicated specifications.

With respect to the alternative denotation (‡), we have that

$$S\ (i:o) \equiv R,$$

$$S\ \langle i:o \rangle \equiv R$$

are equivalent to

$$S\ \{r:s\} \equiv R[{}^{i}_{\diamond r}\ {}^{o}_{\diamond s}],$$

$$S\ \{r:s\} \equiv \#r = \infty \Rightarrow R[{}^{i}_{\diamond r}\ {}^{o}_{\diamond s}],$$

respectively. An std-specification $S\ \{i:o\}$ is consistent iff

$$\exists \tau \in (T^{\omega})^{n} \overset{cp}{\to} (T^{\omega})^{m} : \forall i \in (T^{\infty})^{n} : R_S[{}^{o}_{\tau(i)}].$$

Since the denotation of an std-specification is a set of timed, pulse-driven, stream processing functions, $\otimes$ and $\leadsto$ can be defined in exactly the same way as above.

In the case of std-specifications, the rules are quite simple. The consequence-rule looks as usual:

Rule 11 :

$$\frac{R_{S_2} \Rightarrow R_{S_1}}{S_1\ \{i:o\} \leadsto S_2\ \{i:o\}}$$

One premise is sufficient also in the μ-rule:

Rule 12 :

$$\frac{R_{S_2}[{}^{x}_{o}] \Rightarrow R_{S_1}}{S_1\ \{i:o\} \leadsto \mu S_2\ \{i\,, x:o\}}$$

Since there is no time abstraction, and since any $\tau \in [\![\ S_2\ (i\,, x:o)\ ]\!]$ is pulse-driven, which means that, for any infinite input stream s, the equation

$$\tau(s, r) = r \qquad (*)$$

has a unique, infinite solution r, an invariant is not needed. Thus there are no additional fixpoints to be eliminated.

For any set σ of timed, pulse-driven, stream processing functions of type $(T^{\omega})^{n} \overset{cp}{\to} (T^{\omega})^{m}$, let

$$std(\sigma)\ \{i:o\}$$

be an std-specification, whose input/output relation is characterized by the following equivalence:

$$R_{std(\sigma)} \Leftrightarrow \exists \tau \in \sigma : \tau(i) = o.$$

Then, for any std-specification $S\ \{i : o\}$, if

$$[\![\ \mu\, std(\sigma)\ \{i\,, x : o\}\ ]\!] \subseteq [\![\ S\ \{i : o\}\ ]\!],$$

it follows by Rule 12 that

$$S\ \{i : o\} \leadsto \mu\, std(\sigma)\ \{i\,, x : o\}$$

under the assumptions that any valid formula of the base-logic (in which the premises are formulated) can be proved, and that the base-logic allows $R_{std(\sigma)}$ to be syntactically expressed (this corresponds to relative, semantic completeness with respect to arbitrary components).

The rule for the $\otimes$-operator is formulated accordingly and satisfies a similar completeness result:

Rule 13 :

$$\frac{R_{S_1} \wedge R_{S_2} \Rightarrow R_S}{S\ \{i\,, r : o\,, s\} \leadsto S_1\ \{i\,, x : o\,, y\} \otimes S_2\ \{y\,, r : x\,, s\}}$$

Had we used the alternative semantics (‡), rules with invariants would have been needed, because the equation ($*$) may have more than one solution if s is finite. In fact, Rule 2 and 3, with $\{\}$ substituted for $()$, would have allowed us to claim a completeness similar to that for ti-specifications.

The following conversion rules are also useful:

Rule 14 :

$$\frac{\begin{array}{l} i \neq \infty \Rightarrow R_{S_1} \\ R_{S_2} \Rightarrow R_{S_1}[{}^{i}_{\diamond r}\ {}^{o}_{\diamond s}] \end{array}}{S_1\ (i : o) \leadsto S_2\ \{r : s\}}$$

Rule 15 :

$$\frac{R_{S_2}[{}^{i}_{\diamond r}\ {}^{o}_{\diamond s}] \Rightarrow R_{S_1}}{S_1\ \{r : s\} \leadsto S_2\ (i : o)}$$

Rule 16 :

$$\frac{R_{S_2} \Rightarrow R_{S_1}[{}^{i}_{\diamond r}\ {}^{o}_{\diamond s}]}{S_1\ \langle i : o\rangle \leadsto S_2\ \{r : s\}}$$

Rule 17 :

$$\frac{R_{S_2}[{}^{i}_{\diamond r}\ {}^{o}_{\diamond s}] \Rightarrow R_{S_1}}{S_1\ \{r : s\} \leadsto S_2\ \langle i : o\rangle}$$

6 Simple and General Specifications

Above we have advocated a relational approach for the specification of dataflow networks — relational in the sense that the relation between the input and the output streams has been specified. We have distinguished between ti-, wtd- and std-specifications. For all three classes of specifications, we have formulated rules, which allow specifications to be decomposed into networks of specifications. With respect to the rules for ti- and std-specifications, we have been able to claim only rather restricted completeness results. We now discuss this problem in more detail. As will be shown, the underlying cause is the so-called Brock/Ackermann anomaly [BA81].

Let $K = \{1, \surd\}$. To investigate the issue, (inspired by [Bro92c]) we define three timed, pulse-driven, stream processing functions

$$\tau_1, \tau_2, \tau_3 : K^\omega \overset{cp}{\rightarrow} K^\omega,$$

such that

$$\begin{array}{l}
\tau_1(in) = 1 \mathbin{\&} g_1(in),\\
\text{where}\\
g_1(\surd \mathbin{\&} in) = \surd \mathbin{\&} g_1(in),\\
g_1(1 \mathbin{\&} in) = 1 \mathbin{\&} \surd^{\#in},
\end{array}$$

$$\begin{array}{l}
\tau_2(in) = 1 \mathbin{\&} g_2(in),\\
\text{where}\\
g_2(\surd \mathbin{\&} in) = \surd \mathbin{\&} g_2(in),\\
g_2(1 \mathbin{\&} in) = \surd \mathbin{\&} h_2(in),\\
h_2(\surd \mathbin{\&} in) = \surd \mathbin{\&} h_2(in),\\
h_2(1 \mathbin{\&} in) = 1 \mathbin{\&} \surd^{\#in},
\end{array}$$

$$\begin{array}{l}
\tau_3(in) = \surd \mathbin{\&} g_3(in),\\
\text{where}\\
g_3(\surd \mathbin{\&} in) = \surd \mathbin{\&} g_3(in),\\
g_3(1 \mathbin{\&} in) = 1 \mathbin{\&} 1 \mathbin{\&} \surd^{\#in}.
\end{array}$$

The three formulas $R_{ti(\tau_1)}$, $R_{ti(\tau_2)}$ and $R_{ti(\tau_3)}$ characterize three different relations. It also holds that

$$R_{ti(\tau_1)} \Rightarrow R_{ti(\tau_2)} \vee R_{ti(\tau_3)}.$$

Thus any ti-specification with τ_2 and τ_3 in its denotation has also τ_1 in its denotation. This is no problem as long as for any pair of ti-components C_1 and C_2, characterized by

$$\begin{array}{l}
[\![\ ti(\tau_1)\ (i:o)\ ]\!],\\
[\![\ ti(\tau_2)\ (i:o)\ ]\!] \cup [\![\ ti(\tau_3)\ (i:o)\ ]\!],
\end{array}$$

respectively, there is no observable behavior of C_1 that is not also an observable behavior of C_2. Unfortunately, since

$$\begin{aligned}&\tau \in [\![\ ti(\tau_1)\ (i:o)\ ]\!] \wedge \tau(s) = s \Rightarrow \diamond s = \langle 1,1\rangle,\\ &\tau \in [\![\ ti(\tau_2)\ (i:o)\ ]\!] \wedge \tau(s) = s \Rightarrow \diamond s = \langle 1\rangle,\\ &\tau \in [\![\ ti(\tau_3)\ (i:o)\ ]\!] \wedge \tau(s) = s \Rightarrow \diamond s = \epsilon,\end{aligned}$$

this is not the case, because when we apply the μ-operator to C_1, we get $\langle 1,1\rangle$ as output stream, and when we apply the μ-operator to C_2, we either get $\langle 1\rangle$ or ϵ as output stream. Consequently, there is no sound and compositional proof system with respect to ti-specifications, which allow us to prove that μC_2 cannot produce $\langle 1,1\rangle$, because any ti-specification fulfilled by C_2 is also fulfilled by C_1, and C_1 does not satisfy the property we want to prove. This explains why in the case of ti-specifications we could not formulate rules for the μ- and $\otimes$-operators, which satisfy the same "strong" completeness result as the corresponding rules for std-specifications.

We will now prove that there is a similar problem in the case of wtd-specifications. First observe that the formulas $R_{wtd(\tau_1)}$, $R_{wtd(\tau_2)}$ and $R_{wtd(\tau_3)}$ characterize three different relations. It also holds that

$$\begin{aligned}&R_{wtd(\tau_1)} \Rightarrow R_{wtd(\tau_2)} \vee R_{wtd(\tau_3)},\\ &\tau \in [\![\ wtd(\tau_1)\ \langle i:o\rangle\ ]\!] \wedge \tau(s) = s \Rightarrow \diamond s = \langle 1,1\rangle,\\ &\tau \in [\![\ wtd(\tau_2)\ \langle i:o\rangle\ ]\!] \wedge \tau(s) = s \Rightarrow \diamond s = \langle 1\rangle,\\ &\tau \in [\![\ wtd(\tau_3)\ \langle i:o\rangle\ ]\!] \wedge \tau(s) = s \Rightarrow \diamond s = \epsilon.\end{aligned}$$

Thus we are in exactly the same situation as for ti-specifications.

Because we consider timed, pulse-driven, stream processing functions only, and we are only interested in the behavior for complete (infinite) inputs — which means that the corresponding fixpoints are always infinite and unique — there is no Brock/Ackermann anomaly in the case of std-specifications. This is also the reason why the rules for this class of specifications satisfy a stronger completeness result. On the other hand, had we used the alternative denotation (‡), we would have run into trouble with the Brock/Ackermann anomaly even in the case of std-specifications.

To get around the Brock/Ackermann anomaly, ti- and wtd-specifications are augmented with so-called prophecies. More precisely, an additional parameter (actually a list) modeling the nondeterministic choices taken inside a component is added. We use the same tagging convention as before to distinguish ti- and wtd-specifications:

$$\begin{aligned}&S\ (i:o:p) \equiv R\ \rhd\ P,\\ &S\ \langle i:o:p\rangle \equiv R\ \rhd\ P.\end{aligned}$$

S is the specification's name; i and o are disjoint, repetition free lists of identifiers representing the input and the output streams; p is a list of identifiers representing prophecies; R is a formula with the elements of i, o and p as its only free variables; P is a formula with the elements of p as its only free variables. For each prophecy alternative p, R characterizes the relation between the input- and the output-streams with respect to the nondeterministic choice characterized by p. P is a so-called prophecy formula characterizing the set of possible prophecies. There is a close correspondence between what is called a prophecy variable in [AL88], an oracle in [Kel78], and what we refer to as prophecies.

These two new formats will be referred to as general ti- and wtd-specifications, respectively. In contrast, the formats used in the earlier sections are now called simple ti- and wtd-specifications. A general specifications can be thought of as a set of simple specifications — one simple specification for each prophecy. Their denotations are characterized as follows:

$$[\![\ S\ (i\!:\!o\!:\!p)\ ]\!] \stackrel{\mathsf{def}}{=} \{\tau \in (T^\omega)^n \xrightarrow{cp} (T^\omega)^m | \exists p : P \wedge \forall r \in (T^\omega)^n : R[^{i}_{\diamond r}\ ^{o}_{\diamond \tau(r)}]\},$$

$$[\![\ S\ \langle i\!:\!o\!:\!p\rangle\ ]\!] \stackrel{\mathsf{def}}{=} \{\tau \in (T^\omega)^n \xrightarrow{cp} (T^\omega)^m | \exists p : P \wedge \forall r \in (T^\infty)^n : R[^{i}_{\diamond r}\ ^{o}_{\diamond \tau(r)}]\}.$$

General ti- and wtd-specifications are feasible iff

$$\exists \tau \in (T^\omega)^n \xrightarrow{cp} (T^\omega)^m : \exists p : P \wedge \forall r \in (T^\omega)^n : R[^{i}_{\diamond r}\ ^{o}_{\diamond \tau(r)}],$$

$$\exists \tau \in (T^\omega)^n \xrightarrow{cp} (T^\omega)^m : \exists p : P \wedge \forall r \in (T^\infty)^n : R[^{i}_{\diamond r}\ ^{o}_{\diamond \tau(r)}],$$

respectively. For any general specification S, we use respectively R_S and P_S to characterize its input/output relation and prophecy formula.

Using general specifications, the Brock/Ackermann anomaly is no longer a problem. For example, for any ti-component C, let L be a set of labels such that there is a bijection b from L to C, then

$$S\ (i\!:\!o\!:\!p) \equiv R_{ti(b(l))}\ \rhd\ l \in L$$

is a general ti-specification, whose denotation is equal to C. Of course, we then assume that our assertion language allows $R_{ti(b(l))}$ to be syntactically expressed.

Again the definitions of $\otimes$ and $\rightsquigarrow$ carry over straightforwardly. The rules are also easy to generalize. We give the general versions of Rule 5 and 6:

$$\textsf{Rule 18 :}\quad \frac{\begin{array}{l} P_{S_1} \Rightarrow \textsf{safe}(\lambda x : I) \\ P_{S_1} \Rightarrow I[^{x}_{\epsilon}] \\ P_{S_1} \wedge I \wedge R_{S_2} \Rightarrow I[^{x}_{o}] \\ P_{S_1} \wedge I[^{x}_{o}] \wedge R_{S_2}[^{x}_{o}] \Rightarrow R_{S_1} \end{array}}{S_1\ \langle i\!:\!o\!:\!p\rangle \rightsquigarrow \mu\, S_2\ \langle i\,, x\!:\!o\!:\!p\rangle}$$

$$\begin{array}{l}
\textsf{Rule 19 :}\\
P_S \Rightarrow \mathsf{safe}(\lambda x : I_1) \vee \mathsf{safe}(\lambda y : I_2)\\
P_S \Rightarrow I_1[{}^{x}_{\epsilon}] \vee I_2[{}^{y}_{\epsilon}]\\
P_S \wedge I_1 \wedge R_{S_1} \Rightarrow I_2\\
P_S \wedge I_2 \wedge R_{S_2} \Rightarrow I_1\\
P_S \wedge I_1 \wedge R_{S_1} \wedge I_2 \wedge R_{S_2} \Rightarrow R_S\\
\hline
S\ \langle i\,, r\!:\!o\,, s\!:\!p\rangle \rightsquigarrow S_1\ \langle i\,, x\!:\!o\,, y\!:\!p\rangle \otimes S_2\ \langle y\,, r\!:\!x\,, s\!:\!p\rangle
\end{array}$$

In these rules the specifications are assumed to have identical prophecy formulas. The invariants may now also refer to prophecies. The other rules for simple specifications can be translated into rules for general specifications in a similar way.

We may also formulate rules which relate simple and general specifications.

$$\begin{array}{l}
\textsf{Rule 20 :}\\
P_{S_2} \wedge R_{S_2} \Rightarrow R_{S_1}\\
\hline
S_1\ \langle i\!:\!o\rangle \rightsquigarrow S_2\ \langle i\!:\!o\!:\!p\rangle
\end{array}
\qquad
\begin{array}{l}
\textsf{Rule 21 :}\\
\exists p : P_{S_1}\\
P_{S_1} \wedge R_{S_2} \Rightarrow R_{S_1}\\
\hline
S_1\ \langle i\!:\!o\!:\!p\rangle \rightsquigarrow S_2\ (i\!:\!o)
\end{array}$$

As discussed in [SDW93], [Bro92b] there are a number of specifications, which can be expressed as simple specifications, but which perhaps become clearer when formulated as general specifications. To show that, alternating bit transmission is investigated once more.

Example 13 Alternating Bit Transmission, revisited:
We redo Example 11 under the new requirement that the lossiness of the unreliable receiver UR is nondeterministic. As indicated in Figure 6, there is no input channel r, which determines whether an input pair is properly received or not — this decision is now taken inside UR.

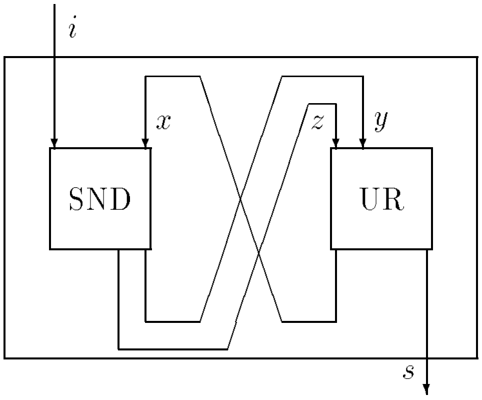

Figure 6: Alternating bit network.

The specification of SND remains as in Example 11. We give two specifications of UR. The first one is a simple wtd-specification:

$\mathrm{UR}\ \langle z \in D^\omega, y \in M^\omega : x \in M^\omega, s \in D^\omega \rangle \equiv$

$$\exists p \in K^\omega : \#\mathsf{ok}Ⓒp = \infty \wedge$$

```
let
    $(z', y', p') = \{(d, m, \mathsf{ok}) | d \in D, m \in M\}Ⓒ(z, y, p)$
in
    $(x, s) = \propto(z', y').$
```

The relationship to the identically named specification in Example 11 is striking. The existentially quantified p simulates the input channel r. This simple wtd-specification can be transformed into a general wtd-specification as follows:

$\mathrm{UR}\ \langle z \in D^\omega, y \in M^\omega : x \in M^\omega, s \in D^\omega : p \in K^\omega \rangle \equiv$

```
let
    $(z', y', p') = \{(d, m, \mathsf{ok}) | d \in D, m \in M\}Ⓒ(z, y, p)$
in
    $(x, s) = \propto(z', y')$
```

$\rhd$

$$\#\mathsf{ok}Ⓒp = \infty.$$

Thus the only difference is that the nondeterminism is characterized by a separate formula — the prophecy formula. This supports an additional structuring of specifications, which in many cases may lead to increased readability. Moreover, since the invariants in the rules for general specifications may refer to prophecies, specifications written in this general format may make it easier to construct proofs.

□

The rules for general ti- and wtd-specifications satisfy completeness results similar to those for std-specifications — namely what is normally referred to a semantic, relative completeness. These results follow easily from the restricted completeness results for simple specifications. See completeness proof for general specifications given in [SDW93].

7 Conclusions

Relational specifications have proved to be well-suited for the description of sequential programs. Prominent techniques like Hoare's assertion method [Hoa69], Dijkstra's wp-calculus [Dij76], or Hehner's predicative specifications [Heh84] are based on formulas characterizing the relation between the input and the output states.

In the case of interactive systems, the relational approach has run into difficulties. As demonstrated in [BA81], specifications where the relationship between the input and the output streams are characterized by simple relations are not sufficiently expressive to allow the behavior of a dataflow network to be deduced from the specifications of its components in a compositional style. Simple relations are not sufficiently expressive to represent the semantic information needed to determine the behavior of a component with respect to a feedback operator. Technically speaking, with respect to feedback loops, we define the behavior as the least fixpoints of the operationally feasible computations. As shown above, for simple relations it is not possible to distinguish the least fixpoints of the operationally feasible computations from other fixpoints. One way to deal with this problem is to replace relations by sets of functions that are monotonic with respect to the prefix ordering on streams. However, for certain components like fair merge a straightforward specification leads to conflicts with the monotonicity constraint.

Our paper shows how one can get around these problems taking a more pragmatic point of view. We have distinguished between three classes of specifications, namely ti-, wtd- and std-specifications. The two first classes have been split into two subclasses, namely into simple and general specifications. For each class of specifications a number of refinement rules have been formulated and their completeness have been discussed.

Components that can be specified by wtd-specifications constitute an important subclass of dataflow components. Of course such components can easily be specified by std-specifications. However, it seems more adequate to specify these components without mentioning time explicitly. In some sense a wtd-specification can be said to be more abstract than the corresponding std-specification.

Similarly, many components are time independent in the sense that they can be specified by a ti-specification. In practice such components may just as well be specified by a wtd-specification. However, as we have seen, the refinement rules for ti-specifications are simpler than those for wtd-specifications, moreover it is easier to prove consistency since it is enough to construct an ordinary (untimed) stream processing which satisfies the specification. To prove consistency of a wtd-specification it is necessary to show that it is satisfied by a timed, pulse-driven, stream processing function.

Finally, since many components can only be specified by an std-specification, we may conclude that all three classes of specifications have their respective merits. Moreover, as we have emphasized, since they are all assigned the same type of semantics, the different types of specifications may be exploited in the very same system development. In fact, the $\otimes$-operator can be used to build networks consisting of both ti-, wtd- and std-specifications. The rules, which allow one type of specification to be translated into another type of specification, can be used for the development of such networks.

Our approach is related to Park's proposals in [Par83]. In some sense he distinguishes between the same three classes of specifications as we. Our approach differs from his in the insistence upon time abstraction and also in the use of prophecies to handle the Brock/Ackermann anomaly. Another difference is our refinement calculus.

The approach presented in this paper can easily be combined with a specification style based on the assumption/commitment paradigm. The rules for assumption/commitment specifications presented in [SDW93] are basically the rules for ti-specifications given above. In fact, this paper shows how the refinement calculus and specification technique given in [SDW93] can be generalized to deal with wtd- and std-specifications.

8 Acknowledgements

We would like to thank P. Collette, F. Dederichs, T. Gritzner and R. Weber who have read earlier drafts of this report and provided valuable feedback. Helpful comments have also been received from O. J. Dahl and O. Owe.

References

[AL88] M. Abadi and L. Lamport. The existence of refinement mappings. Technical Report 29, Digital, Palo Alto, 1988.

[BA81] J. D. Brock and W. B. Ackermann. Scenarios: A model of non-determinate computation. In J. Diaz and I. Ramos, editors, *Proc. Formalization of Programming Concepts, Lecture Notes in Computer Science 107*, pages 252–259, 1981.

[BDD$^+$92] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. F. Gritzner, and R. Weber. The design of distributed systems — an introduction to Focus. Technical Report SFB 342/2/92 A, Technische Universität München, 1992.

[Bro89] M. Broy. Towards a design methodology for distributed systems. In M. Broy, editor, *Proc. Constructive Methods in Computing Science*, pages 311–364. Springer, 1989.

[Bro92a] M. Broy. Compositional refinement of interactive systems. Technical Report 89, Digital, Palo Alto, 1992.

[Bro92b] M. Broy. A functional rephrasing of the assumption/commitment specification style. Manuscript, November 1992.

[Bro92c] M. Broy. Functional specification of time sensitive communicating systems. In M. Broy, editor, *Proc. Programming and Mathematical Method*, pages 325–367. Springer, 1992.

[Bro93] M. Broy. (Inter-) Action refinement: The easy way. In M. Broy, editor, *Proc. Program Design Calculi*, page ?? Springer, 1993.

[Dij76] E. W. Dijkstra. *A Disipline of Programming.* Prentice-Hall, 1976.

[Heh84] E. C. R. Hehner. *The Logic of Programming.* Prentice-Hall, 1984.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–583, 1969.

[Jon90] C. B. Jones. *Systematic Software Development Using VDM, Second Edition.* Prentice-Hall, 1990.

[Kel78] R. M. Keller. Denotational models for parallel programs with indeterminate operators. In E. J. Neuhold, editor, *Proc. Formal Description of Programming Concepts*, pages 337–366. North-Holland, 1978.

[Kle52] S. C. Kleene. *Introduction to Metamathematics.* Van Nostrand, 1952.

[Mor90] C. Morgan. *Programming from Specifications.* Prentice-Hall, 1990.

[Par83] D. Park. The “fairness” problem and nondeterministic computing networks. In J. W. de Bakker and J van Leeuwen, editors, *Proc. 4th Foundations of Computer Science, Mathematical Centre Tracts 159*, pages 133–161. Mathematisch Centrum Amsterdam, 1983.

[SDW93] K. Stølen, F. Dederichs, and R. Weber. Assumption/commitment rules for networks of asynchronously communicating agents. Technical Report SFB 342/2/93 A, Technische Universität München, 1993.

A Proofs

Proposition 1 *Given disjoint lists of variables i, o, r, s, x and y, and formulas I_1 and I_2 with respectively the elements of i, r, x and i, r, y as free variables. If*

$$\mathsf{adm}(\lambda x : I_1) \vee \mathsf{adm}(\lambda y : I_2), \tag{1}$$

$$I_1[{}^{x}_{\epsilon}] \vee I_2[{}^{y}_{\epsilon}], \tag{2}$$

$$I_1 \wedge R_{S_1} \Rightarrow I_2, \tag{3}$$

$$I_2 \wedge R_{S_2} \Rightarrow I_1, \tag{4}$$

$$I_1 \wedge R_{S_1} \wedge I_2 \wedge R_{S_2} \Rightarrow R_S, \tag{5}$$

then

$$S\ (i\,, r\!:\!o\,, s) \leadsto S_1\ (i\,, x\!:\!o\,, y) \otimes S_2\ (y\,, r\!:\!x\,, s). \tag{6}$$

Proof: Assume that 1 - 5 hold, and that

$$\tau_1 \in [\![\ S_1\ (i\,, x\!:\!o\,, y)\ ]\!], \tag{7}$$

$$\tau_2 \in [\![\ S_2\ (y\,, r\!:\!x\,, s)\ ]\!]. \tag{8}$$

Given two timed stream tuples i' and r', and assume that

$$\tau_1(i', x') = (o', y') \wedge \tau_2(y', r') = (x', s') \tag{9}$$

is a least fixpoint solution. Let

$$(i, r, o, y, x, s) \overset{\mathsf{def}}{=} (\diamond i', \diamond r', \diamond o', \diamond y', \diamond x', \diamond s'). \tag{10}$$

The monotonicity of τ_1 and τ_2 implies that there are chains $\hat{o}, \hat{y}, \hat{x}$ and $\hat{s}$ such that

$$(\hat{o}_1, \hat{y}_1) \overset{\mathsf{def}}{=} (\epsilon, \epsilon), \tag{11}$$

$$(\hat{x}_1, \hat{s}_1) \overset{\mathsf{def}}{=} (\epsilon, \epsilon), \tag{12}$$

$$(\hat{o}_j, \hat{y}_j) \overset{\mathsf{def}}{=} \tau_1(i', \hat{x}_{j-1}) \qquad \text{if } j > 1, \tag{13}$$

$$(\hat{x}_j, \hat{s}_j) \overset{\mathsf{def}}{=} \tau_2(\hat{y}_{j-1}, r') \qquad \text{if } j > 1. \tag{14}$$

Since the least upper bound of the Kleene chain is equal to the least fixpoint solution [Kle52], 9, 11, 12, 13 and 14 imply

$$\sqcup(\hat{o}, \hat{y}, \hat{x}, \hat{s}) = (o', y', x', s'). \tag{15}$$

Assume for an arbitrary $j \geq 1$

$$I_1[{}^{x}_{\diamond\hat{x}_j}]. \tag{16}$$

7, 10 and 13 imply

$$R_{S_1}[{}^{x}_{\diamond\hat{x}_j}\ {}^{o}_{\diamond\hat{o}_{j+1}}\ {}^{y}_{\diamond\hat{y}_{j+1}}]. \tag{17}$$

3, 16 and 17 imply

$$I_2[{}^{y}_{\diamond\hat{y}_{j+1}}].$$

Thus

$$\forall j : I_1[{}^{x}_{\diamond\hat{x}_j}] \Rightarrow I_2[{}^{y}_{\diamond\hat{y}_{j+1}}]. \tag{18}$$

By a similar argument

$$\forall j : I_2[{}^{y}_{\diamond\hat{y}_j}] \Rightarrow I_1[{}^{x}_{\diamond\hat{x}_{j+1}}]. \tag{19}$$

2, 11, 12, 18 and 19 imply

$$\forall j : (\exists k : k > j \wedge I_1[{}^{x}_{\diamond\hat{x}_k}]) \wedge (\exists k : k > j \wedge I_2[{}^{y}_{\diamond\hat{y}_k}]). \tag{20}$$

1, 20 and the continuity of $\diamond$ imply

$$I_1[{}^{x}_{\diamond\sqcup\hat{x}}] \vee I_2[{}^{y}_{\diamond\sqcup\hat{y}}]. \tag{21}$$

10, 15 and 21 imply

$$I_1 \vee I_2.$$

Without loss of generality, assume

$$I_1. \tag{22}$$

7, 9 and 10 imply

$$R_{S_1}. \tag{23}$$

3, 22 and 23 imply

$$I_2. \tag{24}$$

8, 9 and 10 imply

$$R_{S_2}. \tag{25}$$

5, 22, 23, 24 and 25 imply

$$R_S. \tag{26}$$

This proves 6.

end of proof

Proposition 2 *Given disjoint lists of variables i, o, r, s, x and y, and formulas I_1 and I_2 with respectively the elements of i, r, x and i, r, y as free variables. If*

$$\mathsf{safe}(\lambda x : I_1) \vee \mathsf{safe}(\lambda y : I_2), \tag{27}$$

$$I_1[{}^{x}_{\epsilon}] \vee I_2[{}^{y}_{\epsilon}], \tag{28}$$

$$I_1 \wedge R_{S_1} \Rightarrow I_2, \tag{29}$$

$$I_2 \wedge R_{S_2} \Rightarrow I_1, \tag{30}$$

$$I_1 \wedge R_{S_1} \wedge I_2 \wedge R_{S_2} \Rightarrow R_S, \tag{31}$$

then

$$S\ \langle i\,, r\!:\!o\,, s\rangle \rightsquigarrow S_1\ \langle i\,, x\!:\!o\,, y\rangle \otimes S_2\ \langle y\,, r\!:\!x\,, s\rangle. \tag{32}$$

Proof: Assume that 27 - 31 hold, and that

$$\tau_1 \in [\![\ S_1\ \langle i\,, x\!:\!o\,, y\rangle\ ]\!], \tag{33}$$

$$\tau_2 \in [\![\ S_2\ \langle y\,, r\!:\!x\,, s\rangle\ ]\!]. \tag{34}$$

Given two timed, complete (infinite) stream tuples i' and r', and assume that

$$\tau_1(i', x') = (o', y') \wedge \tau_2(y', r') = (x', s'). \tag{35}$$

Let

$$(i, r, o, y, x, s) \overset{\mathsf{def}}{=} (\diamond i', \diamond r', \diamond o', \diamond y', \diamond x', \diamond s'). \tag{36}$$

The monotonicity of τ_1 and τ_2 implies that there are chains $\hat{o}, \hat{y}, \hat{x}$ and $\hat{s}$ such that

$$(\hat{o}_1, \hat{y}_1) \overset{\mathsf{def}}{=} (\epsilon, \epsilon), \tag{37}$$

$$(\hat{x}_1, \hat{s}_1) \overset{\mathsf{def}}{=} (\epsilon, \epsilon), \tag{38}$$

$$(\hat{o}_j, \hat{y}_j) \overset{\mathsf{def}}{=} \tau_1(i', \hat{x}_{j-1}) \qquad \text{if } j > 1, \tag{39}$$

$$(\hat{x}_j, \hat{s}_j) \overset{\mathsf{def}}{=} \tau_2(\hat{y}_{j-1}, r') \qquad \text{if } j > 1. \tag{40}$$

Since the least upper bound of the Kleene chain is equal to the least fixpoint solution [Kle52], 35, 37, 38, 39 and 40 imply

$$\sqcup(\hat{o}, \hat{y}, \hat{x}, \hat{s}) = (o', y', x', s'). \tag{41}$$

Let $\tilde{o}, \tilde{y}, \tilde{x}$ and $\tilde{s}$ be infinite sequences of complete (infinite) stream tuples such that

$$(\tilde{o}_1, \tilde{y}_1) \overset{\mathsf{def}}{=} (\surd^\infty, \surd^\infty), \tag{42}$$

$$(\tilde{x}_1, \tilde{s}_1) \overset{\mathsf{def}}{=} (\surd^\infty, \surd^\infty), \tag{43}$$

$$(\tilde{o}_j, \tilde{y}_j) \overset{\mathsf{def}}{=} \tau_1(i', \tilde{x}_{j-1}) \qquad \text{if } j > 1, \tag{44}$$

$$(\tilde{x}_j, \tilde{s}_j) \overset{\mathsf{def}}{=} \tau_2(\tilde{y}_{j-1}, r') \qquad \text{if } j > 1. \tag{45}$$

37, 38, 42 and 43 imply

$$(\hat{o}_1, \hat{y}_1, \hat{x}_1, \hat{s}_1) \sqsubseteq (\tilde{o}_1, \tilde{y}_1, \tilde{x}_1, \tilde{s}_1). \tag{46}$$

39, 40, 44, 45 and the monotonicity of τ_1 and τ_2 imply

$$\begin{aligned}&(\hat{o}_j, \hat{y}_j, \hat{x}_j, \hat{s}_j) \sqsubseteq (\tilde{o}_j, \tilde{y}_j, \tilde{x}_j, \tilde{s}_j) \ \Rightarrow \\ &\qquad (\hat{o}_{j+1}, \hat{y}_{j+1}, \hat{x}_{j+1}, \hat{s}_{j+1}) \sqsubseteq (\tilde{o}_{j+1}, \tilde{y}_{j+1}, \tilde{x}_{j+1}, \tilde{s}_{j+1}).\end{aligned} \tag{47}$$

46, 47 and induction on j imply

$$\forall j : (\hat{o}_j, \hat{y}_j, \hat{x}_j, \hat{s}_j) \sqsubseteq (\tilde{o}_j, \tilde{y}_j, \tilde{x}_j, \tilde{s}_j). \tag{48}$$

Assume for an arbitrary $j \geq 1$

$$I_1[{}^{x}_{\diamond\tilde{x}_j}]. \tag{49}$$

33, 36 and 44 imply

$$R_{S_1}[{}^{x}_{\diamond\tilde{x}_j}\ {}^{o}_{\diamond\tilde{o}_{j+1}}\ {}^{y}_{\diamond\tilde{y}_{j+1}}]. \tag{50}$$

29, 49 and 50 imply

$$I_2[{}^{y}_{\diamond\tilde{y}_{j+1}}].$$

Thus

$$\forall j : I_1[{}^{x}_{\diamond\tilde{x}_j}] \Rightarrow I_2[{}^{y}_{\diamond\tilde{y}_{j+1}}]. \tag{51}$$

By a similar argument

$$\forall j : I_2[{}^{y}_{\diamond\tilde{y}_j}] \Rightarrow I_1[{}^{x}_{\diamond\tilde{x}_{j+1}}]. \tag{52}$$

28, 42, 43, 51 and 52 imply

$$\forall j : (\exists k : k > j \wedge I_1[{}^{x}_{\diamond\tilde{x}_k}]) \wedge (\exists k : k > j \wedge I_2[{}^{y}_{\diamond\tilde{y}_k}]). \tag{53}$$

Without loss of generality, assume

$$\mathsf{safe}(\lambda x : I_1). \tag{54}$$

48, 53 and 54 imply

$$\forall j : (\exists k : k > j \wedge I_1[{}^{x}_{\diamond\hat{x}_k}]) \tag{55}$$

54, 55 and the continuity of $\diamond$ imply

$$I_1[{}^{x}_{\diamond\sqcup\hat{x}}]. \tag{56}$$

36, 41 and 56 imply

$$I_1. \tag{57}$$

33, 35 and 36 imply

$$R_{S_1}. \tag{58}$$

29, 57 and 58 imply

$$I_2. \tag{59}$$

34, 35 and 36 imply

$$R_{S_2}. \tag{60}$$

31, 57, 58, 59 and 60 imply

$$R_S. \tag{61}$$

This proves 32.

end of proof

Proposition 3 *Given disjoint lists of variables i, x and o, and a formula I with the elements of i and x as free variables. If*

$$\begin{align}
&(\forall j : I[\substack{x\\c_j}] \wedge c_j \in (D^*)^m \wedge \exists o : R_{S_2}[\substack{x\\c_j}] \wedge c_{j+1} \sqsubseteq o) \Rightarrow I[\substack{x\\\sqcup c}], \tag{62}\\
&I[\substack{x\\\epsilon}], \tag{63}\\
&I \wedge x \in (D^*)^m \wedge x \sqsubset x' \sqsubseteq o \wedge R_{S_2} \Rightarrow I[\substack{x\\x'}], \tag{64}\\
&I[\substack{x\\o}] \wedge R_{S_2}[\substack{x\\o}] \Rightarrow R_{S_1}, \tag{65}
\end{align}$$

then

$$S_1 \ \langle i : o \rangle \rightsquigarrow \mu\, S_2 \ \langle i\,, x : o \rangle. \tag{66}$$

Proof: Assume that 62-65 hold, and that

$$\tau \in [\![\ S_2 \ \langle i\,, x : o \rangle \ ]\!]. \tag{67}$$

Given a timed, complete (infinite) stream tuple i' and assume that

$$\tau(i', o') = o'. \tag{68}$$

Let

$$(i, o) = (\diamond i', \diamond o'). \tag{69}$$

The monotonicity of τ implies there is a chain $\hat{o}$ such that

$$\begin{align}
\hat{o}_1 &= \epsilon, \tag{70}\\
\hat{o}_j &= \tau(i', \hat{o}_{j-1}) \qquad \text{if } j > 1. \tag{71}
\end{align}$$

Since the least upper bound of the Kleene-chain is equal to the least fixpoint solution, [Kle52], 68, 70 and 71 imply

$$\sqcup \hat{o} = o'. \tag{72}$$

Because of the observation made on Page 22, we may assume that

$$\forall j : \hat{o}_j \in (D^*)^m. \tag{73}$$

63 and 70 imply

$$I[\substack{x\\\diamond \hat{o}_1}]. \tag{74}$$

Assume

$$I[\substack{x\\\diamond \hat{o}_j}]. \tag{75}$$

75 implies

$$I[\substack{x\\\diamond(\hat{o}_j \frown [\surd^\infty]^m)}]. \tag{76}$$

71 and the continuity of τ imply

$$\hat{o}_{j+1} \sqsubseteq \tau(i', \hat{o}_j \frown [\surd^\infty]^m). \tag{77}$$

77, the fact that $\hat{o}$ is a chain and the continuity of $\diamond$ imply

$$\diamond\hat{o}_j \sqsubseteq \diamond\hat{o}_{j+1} \sqsubseteq \diamond\tau(i', \hat{o}_j \frown [\surd^{\infty}]^m). \tag{78}$$

67 implies

$$R_{S_2}\left[\begin{smallmatrix} x & o \\ \diamond(\hat{o}_j \frown [\surd^{\infty}]^m) & \diamond\tau(i',\hat{o}_j \frown [\surd^{\infty}]^m)\end{smallmatrix}\right]. \tag{79}$$

64, 73, 76, 78 and 79 imply

$$I\genfrac{[}{]}{0pt}{}{x}{\diamond\hat{o}_{j+1}}. \tag{80}$$

78, 79 and 80 imply

$$I\genfrac{[}{]}{0pt}{}{x}{\diamond\hat{o}_{j+1}} \wedge \exists o : R_{S_2}\genfrac{[}{]}{0pt}{}{x}{\diamond\hat{o}_j} \wedge \diamond\hat{o}_{j+1} \sqsubseteq o.$$

Thus

$$\forall j : I\genfrac{[}{]}{0pt}{}{x}{\diamond\hat{o}_j} \Rightarrow I\genfrac{[}{]}{0pt}{}{x}{\diamond\hat{o}_{j+1}} \wedge \exists o : R_{S_2}\genfrac{[}{]}{0pt}{}{x}{\diamond\hat{o}_j} \wedge \diamond\hat{o}_{j+1} \sqsubseteq o. \tag{81}$$

74, 81 and induction on j imply

$$\forall j : I\genfrac{[}{]}{0pt}{}{x}{\diamond\hat{o}_j} \wedge \exists o : R_{S_2}\genfrac{[}{]}{0pt}{}{x}{\diamond\hat{o}_j} \wedge \diamond\hat{o}_{j+1} \sqsubseteq o. \tag{82}$$

62, 73, 82 and the continuity of $\diamond$ imply

$$I\genfrac{[}{]}{0pt}{}{x}{\diamond\sqcup\hat{o}}. \tag{83}$$

69, 72 and 83 imply

$$I\genfrac{[}{]}{0pt}{}{x}{o}. \tag{84}$$

67, 68 and 69 imply

$$R_{S_2}\genfrac{[}{]}{0pt}{}{x}{o}. \tag{85}$$

65, 84 and 85 imply

$$R_{S_1}. \tag{86}$$

This proves 66.

end of proof

Assume the base-logic allows any (semantic) predicate we need to be expressed. Then the following proposition holds.

Proposition 4 *Given a wtd-specification $S\ \langle i : o\rangle$ and a timed, pulse-driven, stream processing function τ such that*

$$\llbracket\ \mu\, wtd(\tau)\ \langle i\,, x : o\rangle\ \rrbracket \subseteq \llbracket\ S\ \langle i : o\rangle\ \rrbracket, \tag{87}$$

where i, x and o have respectively n, m and m elements. Then there is a formula I, with the elements of i and x as its only free variables, such that

$$(\forall j : I\genfrac{[}{]}{0pt}{}{x}{c_j} \wedge c_j \in (D^*)^m \wedge \exists o : R_{wtd(\tau)}\genfrac{[}{]}{0pt}{}{x}{c_j} \wedge c_{j+1} \sqsubseteq o) \Rightarrow I\genfrac{[}{]}{0pt}{}{x}{\sqcup c}, \tag{88}$$

$$I\genfrac{[}{]}{0pt}{}{x}{\epsilon}, \tag{89}$$

$$I \wedge x \in (D^*)^m \wedge x \sqsubset x' \sqsubseteq o \wedge R_{wtd(\tau)} \Rightarrow I\genfrac{[}{]}{0pt}{}{x}{x'}, \tag{90}$$

$$I\genfrac{[}{]}{0pt}{}{x}{o} \wedge R_{wtd(\tau)}\genfrac{[}{]}{0pt}{}{x}{o} \Rightarrow R_S. \tag{91}$$

Proof: Let

$$\begin{aligned}
&I_1 \overset{\mathsf{def}}{=} x = \epsilon,\\
&I_{j+1} \overset{\mathsf{def}}{=} \exists x' : I_j[{}^{x}_{x'}] \wedge x' \in (D^*)^m \wedge \exists o : x' \sqsubset x \sqsubseteq o \wedge R_{wtd(\tau)}[{}^{x}_{x'}],\\
&I_\infty \overset{\mathsf{def}}{=} \exists c : \forall j : I_j[{}^{x}_{c_j}] \wedge c_j \in (D^*)^m \wedge \exists o : R_{wtd(\tau)}[{}^{x}_{c_j}] \wedge c_{j+1} \sqsubseteq o \wedge x = \sqcup c,\\
&I \overset{\mathsf{def}}{=} \exists j : I_j \vee I_\infty.
\end{aligned}$$

Due to the expressiveness assumption made above, I is a formula in the base-logic.

It follows straightforwardly from the definition of I that 88-90 hold. It remains to prove 91. Given some i and o such that

$$I[{}^{x}_{o}] \wedge R_{wtd(\tau)}[{}^{x}_{o}]. \tag{92}$$

It is enough to show that

$$R_S. \tag{93}$$

We first prove the following lemma:

Lemma 1 *There is a chain r such that:*

$$\forall j : r_j \in (D^*)^m, \tag{94}$$
$$r_1 = \epsilon, \tag{95}$$
$$\forall j : \exists o : R_{wtd(\tau)}[{}^{x}_{r_j}] \wedge r_{j+1} \sqsubseteq o, \tag{96}$$
$$\sqcup r = o. \tag{97}$$

If $I_\infty[{}^{x}_{o}]$ holds, then 94-97 follow trivially. If $\neg I_\infty[{}^{x}_{o}]$, it is enough to prove by induction on j, that for any o such that $I_j[{}^{x}_{o}]$ there is a chain r such that 94-97 hold. The base-case $j = 1$ follows trivially. Assume the lemma holds for $j = k$. We prove that it holds for $j = k + 1$. Given some x such that

$$I_{k+1}, \tag{98}$$

98 and the definition of I_{k+1} imply there is an x' such that

$$I_k[{}^{x}_{x'}], \tag{99}$$
$$x' \in (D^*)^m, \tag{100}$$
$$\exists o : R_{wtd(\tau)}[{}^{x}_{x'}] \wedge x' \sqsubset x \sqsubseteq o. \tag{101}$$

The induction hypothesis implies there is a chain r such that

$$\forall j : r_j \in (D^*)^m, \tag{102}$$
$$r_1 = \epsilon, \tag{103}$$
$$\forall j : \exists o : R_{wtd(\tau)}[{}^{x}_{r_j}] \wedge r_{j+1} \sqsubseteq o, \tag{104}$$
$$\sqcup r = x'. \tag{105}$$

100 and 105 imply there is an l such that

$$\forall k : k \geq l \Rightarrow r_k = x'. \tag{106}$$

Let r' be the chain such that

$$j \le l \Rightarrow r'_j = r_j, \tag{107}$$

$$j > l \Rightarrow r'_j = x|_{j-l}. \tag{108}$$

102, 107 and 108 imply

$$\forall j : r'_j \in (D^*)^m. \tag{109}$$

103 and $l \ge 1$ imply

$$r'_1 = \epsilon. \tag{110}$$

101, 104, 107 and 108 imply

$$\forall j \le l : \exists o : R_{wtd(\tau)}[{}^{x}_{r'_j}] \wedge r'_{j+1} \sqsubseteq o. \tag{111}$$

108 implies

$$\sqcup r' = x. \tag{112}$$

It remains to prove

$$\forall j > l : \exists o : R_{wtd(\tau)}[{}^{x}_{r'_j}] \wedge r'_{j+1} \sqsubseteq o. \tag{113}$$

Let

$$t > l. \tag{114}$$

101 implies there are complete (infinite) timed stream tuples y and z such that

$$\diamond(y, z) = (i, x'), \tag{115}$$

$$x \sqsubseteq \diamond\tau(y, z). \tag{116}$$

108, 114 and 116 imply

$$r'_{t+1} \sqsubseteq \diamond\tau(y, z). \tag{117}$$

109, 117 and the continuity of τ imply there is a finite timed stream tuple z' such that

$$\diamond z' = x', \tag{118}$$

$$r'_{t+1} \sqsubseteq \diamond\tau(y, z'). \tag{119}$$

101, 106, 107, 108, 114 and 118 imply

$$\diamond(z' \frown (x' \wr r'_t) \frown [\surd^{\infty}]^m) = r'_t. \tag{120}$$

119 and the monotonicity of τ imply

$$r'_{t+1} \sqsubseteq \diamond\tau(y, z' \frown (x' \wr r'_t) \frown [\surd^{\infty}]^m). \tag{121}$$

120 and 121 imply

$$\exists o : R_{wtd(\tau)}[{}^{x}_{r'_t}] \wedge r'_{t+1} \sqsubseteq o. \tag{122}$$

This proves 113. Thus 94-97 hold for $j = k + 1$. This ends the proof of the lemma.

Let r be a chain such that 94-97 hold. Since each element of r is finite, 96 and the continuity of τ imply we may find a chain t such that

$$\forall j : t_j \in (D^*)^n, \tag{123}$$

$$\sqcup t = i, \tag{124}$$

$$\forall j : \exists o : R_{wtd(\tau)}[{}^{i}_{t_j}\,{}^{x}_{r_j}] \wedge r_{j+1} \sqsubseteq o. \tag{125}$$

Since 95 implies that the first element of r is ϵ, and it obviously holds that

$$\exists o : R_{wtd(\tau)}[{}^{i}_{\epsilon}\,{}^{x}_{\epsilon}] \wedge \epsilon \sqsubseteq o, \tag{126}$$

we may assume that

$$t_1 = r_1 = \epsilon. \tag{127}$$

Let t' and r' be strictly increasing chains of timed stream tuples such that

$$\forall j : t'_j|_{\#t'_j} = t'_j \wedge r'_j|_{\#r'_j} = r'_j \wedge \#t'_j = \#r'_j, \tag{128}$$

$$\forall j : t'_j \in (D^*)^n \wedge r'_j \in (D^*)^m, \tag{129}$$

$$\forall j : \diamond(t'_j, r'_j) = (t_j, r_j). \tag{130}$$

These chains can easily be generated from r and t by adding $\surd$'s. Note that 128 requires all streams occurring in r'_j and t'_j to be of the same length.
97, 124 and 130 imply

$$\diamond \sqcup (t', r') = (i, o). \tag{131}$$

Let t'' and r'' be sequences (not necessarily chains) of finite timed stream tuples such that

$$\forall j : \diamond t''_j = t_j \wedge \diamond r''_j = r_j, \tag{132}$$

$$\forall j : \#t''_j > \#t'_j \wedge \#r''_j > \#r'_j, \tag{133}$$

$$\forall j : r_{j+1} \sqsubseteq \diamond\tau(t''_j, r''_j). \tag{134}$$

The existence of these sequences follows from 94, 123, 125, 129 and the continuity of τ.
87 and 131 imply that 93 follows if we can construct a function

$$\tau' \in [\![\ wtd(\tau)\ \langle i\,, x\!:\!o\rangle\ ]\!], \tag{135}$$

such that

$$\tau'(\sqcup t', \sqcup r') = \sqcup r'. \tag{136}$$

We construct τ' in a step-wise fashion. First we define the behavior for any input (i, x) such that

$$\#i = \#x \wedge i|_{\#i} = i \wedge x|_{\#x} = x,$$

(i.e. which means that all the streams in i and x are of the same length):

- CASE: $(i, x) = (\sqcup t', \sqcup r')$.
  Let

$$\tau'(i, x) = x. \tag{137}$$

- CASE: $(i, x) \sqsubset (\sqcup t', \sqcup r')$.

  Let

  $$\tau'(i, x) = r'_{j+1}, \tag{138}$$

  where $j = \mathsf{max}\{k \mid (t'_k, r'_k) \sqsubseteq (i, x)\}$.

- CASE: $(i, x) \not\sqsubseteq (\sqcup t', \sqcup r')$

  Let

  $$\begin{aligned}\tau'(i, x) = {} & r'_{j+1} \frown (r_{j+1} \wr \diamond\tau(t''_j, r''_j)) \frown [\sqrt{}^{\#\tau(t''_j, r''_j)}]^m \frown \\ & (\tau(t''_j, r''_j) \wr \tau(t''_j \frown (t'_j \wr i), r''_j \frown (r'_j \wr x))),\end{aligned} \tag{139}$$

  where $j = \mathsf{max}\{k \mid (t'_k, r'_k) \sqsubseteq (i, x)\}$.

For any other input let

$$\tau'(i, x) = \tau'(i|_k, x|_k),$$

where $k = \mathsf{min}\{\#i, \#x\}$.

Since the different cases are disjoint it follows that τ' is well-defined. That τ' is monotonic and continuous follow straightforwardly from the monotonicity and continuity of τ.

With respect to the two first cases in the definition of τ' the pulse-drivenness property follows trivially. With respect to the third case (139) it follows that τ' is pulse-driven since τ is pulse-driven and 133, 139 imply

$$\begin{aligned}&\#(i, x) \leq \#(t''_j \frown (t'_j \wr i), r''_j \frown (r'_j \wr x)), \\ &\#\tau(t''_j \frown (t'_j \wr i), r''_j \frown (r'_j \wr x)) \leq \#\tau'(i, x).\end{aligned}$$

Then, finally, 135 follows from 92, 131, 137 and 139.

end of proof