Specification and Refinement of Finite Dataflow Networks — a Relational Approach[⋆]

Manfred Broy and Ketil Stølen

Institut für Informatik, TU München, Postfach 20 24 20
D-80290 München, Germany

Abstract. We specify the black box behavior of dataflow components by characterizing the relation between their input and their 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 modulo a feedback operator.

1 Introduction

Dataflow components can be specified by formulas with a free variable ranging over domains of so-called stream processing functions [7], [5]. 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 can be formulated.

Stream processing functions are required to be both monotonic and continuous with respect to the prefix ordering on domains of stream tuples. Unfortunately, there are certain weakly time dependent components, whose behaviors cannot be specified in terms of prefix monotonic stream processing functions, although explicit timing is not really needed in order to specify their 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 [7]. The behaviors of such components can of course be specified in terms of timed stream processing functions. However, this is a bit like shooting sparrows with a shot-gun.

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 streams. We distinguish between three main classes of such specifications, namely time independent specifications, weakly time dependent specifications and strongly

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

time dependent specifications — from now on shortened to ti-specifications, wtd-specifications and std-specifications, respectively. For each class of specifications refinement rules are given, which allow specifications to be decomposed modulo a feedback operator. Rules, which allow a specification of one class to be translated into a specification of another class, are also given.

Section 2 describes the underlying formalism. In Sect. 3 we introduce the three main classes of specifications. The refinement of such specifications is the topic of Sect. 4. Then, so-called general specifications are introduced in Sect. 5, and the refinement of general specifications is discussed in Sect. 6. Finally, Sect. 7 contains a brief summary and draws some conclusions.

2 Underlying Formalism

N denotes the set of natural numbers. 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 $d \in D$, $r, s \in D^\omega$, $A \subseteq D$ and j be a natural number, then:

- ϵ denotes the empty stream;
- $\langle d_1, \ldots, d_j \rangle$ denotes a stream of length j, 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, j is a natural number 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; $r|_j$, $r \frown s$ and $r \sqsubseteq s$ are generalized in the same pointwise way.

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 called admissible iff whenever P 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. P is prefix-closed iff whenever it yields true for a stream tuple, then it also yields true for any prefix of this stream tuple. P 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 $\tau \in (D^\omega)^n \to (D^\omega)^m$ is called a stream processing function iff it is prefix monotonic and continuous:

$$\text{for all stream tuples } i \text{ and } i' \text{ in } (D^\omega)^n : \; i \sqsubseteq i' \Rightarrow \tau(i) \sqsubseteq \tau(i'),$$

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

That a function is prefix monotonic 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. That a function is prefix continuous implies that the function's behavior for infinite inputs is completely determined by its behavior for finite inputs.

A stream processing function $\tau \in (D^\omega)^n \to (D^\omega)^m$ 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 [4].

The arrows $\to$, $\overset{c}{\to}$ and $\overset{cp}{\to}$ are used to tag domains of ordinary functions, domains of monotonic, continuous functions, and domains of monotonic, 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.

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 one additional function in our examples: if A is a set of n-tuples of messages, d is an n-tuple of messages, and r is an n-tuple of streams, then AⒸ is a stream processing function such that the following axioms hold:

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

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

3 Three Classes of Specifications

In this section we introduce three classes of specifications, namely time independent specifications, weakly time dependent specifications and strongly time dependent specifications — shortened to ti-, wtd- and std-specifications, respectively.

3.1 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 ti-specification S is the set of all timed, pulse-driven, stream processing functions which fulfill R when time signals are abstracted away:

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

In fact, in the case of ti-specifications we could also have used a set of untimed stream processing functions. Thus timed functions are not really required in order to model ti-specifications. However, the chosen denotation makes it easier to relate and compare the different classes of specifications. For any specification S, R_S represents its input/output relation.

Example 1. We specify a filter with two input channels y and r and one output channel s. The data elements to be filtered are input from the channel y. When the n-th message input from r is a fail, it means that the n'th data element input from y is filtered out; 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 gets through. More formally, given that $K = \{\mathsf{ok}, \mathsf{fail}\}$, the filter is specified by:

$$\text{FILTER } (y \in D^{\omega}, r \in K^{\omega} : s \in D^{\omega}) \equiv$$

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

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:

$$\text{RFM } (i \in D^{\omega}, r \in D^{\omega} : o \in D^{\omega}) \equiv$$

$$\forall d \in D : \#\{d\}\copyright i + \#\{d\}\copyright r = \#\{d\}\copyright o.$$

This specification is inconsistent due to the monotonicity constraint. To see this, assume that there is a timed, pulse-driven, stream processing function τ which fulfills the specification. This means that for $a, b \in D$:

$$\diamond\tau(a^{\infty}, \epsilon) = a^{\infty} \wedge b\copyright\tau(a^{\infty}, b^{\infty}) = b^{\infty}.$$

Clearly,

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

which means that τ is not monotonic. 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

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

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 [9]. In [3] 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.

3.2 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 wtd-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:

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

Thus in contrast to a ti-specification, a wtd-specification constrains the behavior only for complete inputs (infinite inputs at the semantic level[2]). As before, for any wtd-specification S, R_S denotes its input/output relation.

As shown in the next three examples, weakly time dependent components can be specified in a very elegant way.

Example 2. The wtd-specification

$$\mathrm{RFM}\ \langle i \in D^{\omega}, r \in D^{\omega} : o \in D^{\omega} \rangle \equiv$$

$$\forall d \in D : \#\{d\} \copyright i + \#\{d\} \copyright r = \#\{d\} \copyright o,$$

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 3. 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:

$$\mathrm{FM}\ \langle i \in D^{\omega}, r \in D^{\omega} : o \in D^{\omega} \rangle \equiv$$

$$\exists p \in \{1, 2\}^{\omega} : split_1(o, p) = i \wedge split_2(o, p) = r,$$

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

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:

$$\begin{aligned} j = b &\Rightarrow split_j(a \,\&\, o, b \,\&\, p) = a \,\&\, split_j(o,p), \\ j \neq b &\Rightarrow split_j(a \,\&\, o, b \,\&\, p) = split_j(o,p). \end{aligned}$$

Example 4. An arbiter is a component that reproduces its input data and in addition adds an infinite number of tokens, here represented by $\bullet$, to its output stream. More formally:

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

It is assumed that $\bullet$ is not an element of D.

3.3 Strongly Time Dependent Specifications

For the specification of strongly time dependent components std-specifications are needed. 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 std-specification S is the set of all timed, pulse-driven, stream processing functions which fulfill R when only complete (infinite) inputs are considered:

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

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:

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

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 practic 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 5. We specify a simple timer handling requests 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, it responds by sending the timeout signal Γ after n time-units, provided it is not reset by a reset message rst. 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:

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

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

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. For the same reason there are ti-specifications that cannot be expressed as wtd-specifications.

4 Refinement

This section introduces a refinement concept corresponding to what is normally referred to as behavioral refinement. With respect to this concept of refinement we give rules which allow specifications to be decomposed modulo a feedback operator.

We first define our feedback operator μ. Given a specification S with n input and $m \leq n$ output identifiers, then $\mu\, S$ represents the network pictured in Fig. 1 (i, x and o represent tuples of $(n-m), m$ and m streams, respectively).

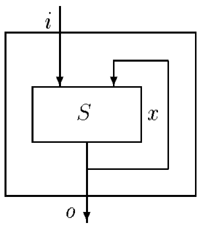

Fig. 1. The Network $\mu\, S$.

More formally, $[\![\, \mu\, S \,]\!] \stackrel{\mathsf{def}}{=} \{\mu\, \tau \mid \tau \in [\![\, S \,]\!]\}$, where for any timed input tuple i, $\mu\, \tau(i) = o$ iff o is the least fixpoint solution of τ with respect to i. This is logically expressed by the following formula:

$$\tau(i, o) = o \wedge (\forall o' : \tau(i, o') = o' \Rightarrow o \sqsubseteq o').$$

Although μ is a rather restricted operator, the refinement rules for more general feedback operators are straightforward generalizations of the rules for μ. See [6].

A specification S_2 refines another specification S_1, written $S_1 \rightsquigarrow S_2$, iff the behaviors specified by S_2 form a subset of the behaviors specified by S_1, formally: $[\![\, S_2 \,]\!] \subseteq [\![\, S_1 \,]\!]$.

The refinement relation $\rightsquigarrow$ is reflexive, transitive and a congruence with respect to feedback operators. Hence, $\rightsquigarrow$ allows compositional system development: once a specification is decomposed into a network of subspecifications, each of these subspecifications can be further refined in isolation.

We now formulate refinement rules for the feedback operator. The first one can be used to decompose a ti-specification:

$$\begin{array}{l}
\mathsf{Rule\ 1:} \\
\mathsf{adm}(\lambda x : I) \\
I[^{x}_{\epsilon}] \\
I \wedge R_{S_2} \Rightarrow I[^{x}_{o}] \\
I[^{x}_{o}] \wedge R_{S_2}[^{x}_{o}] \Rightarrow R_{S_1} \\
\hline
S_1\ (i : o) \rightsquigarrow \mu\, S_2\ (i\, , x : o)
\end{array}$$

The stream tuples are named in accordance with Fig. 1. 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 [8]. This is what fixpoint induction is based on, and this is also the idea behind Rule 1. 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

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

is of course also sound. We refer to this rule as the degenerated version of Rule 1. With the degenerated version we can only prove properties that hold for all fixpoints. Properties which hold only for the least fixpoints cannot be shown. In some sense the invariant of Rule 1 is used to characterize the least fixpoint solutions. We now look at a simple example where the inductive nature of Rule 1 is really needed.

Example 6. 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:

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

However, $R_{S_2}[{}^{x}_{o}] \Rightarrow o = \epsilon$ does not hold. This demonstrates that the degenerated version of Rule 1 is too weak. On the other hand, with $I \overset{\mathsf{def}}{=} x = \epsilon$, as invariant, it is straightforward to deduce $(*)$ using Rule 1.

With respect to wtd-specifications the formulation of refinement rules is more difficult. 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 1 (with $\langle\rangle$ substituted for $()$ in the conclusion) is unsound for wtd-specifications.

Example 7. 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

$$\mathsf{adm}(\lambda x : I), \qquad I[^x_\epsilon], \qquad I \wedge R_{S_2} \Rightarrow I[^x_o].$$

$I[^x_o] \wedge R_{S_2}[^x_o]$ implies $o = 1 \,\&\, 2^\infty$. Thus we may use Rule 1 to prove that

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

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

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

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 1 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 identifier. 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:

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

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

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

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 [8] 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 that

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

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 and admissible). Assume that $i = \diamond r$, then if for all j

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

it follows from $(*)$ and the fact that $\lambda x : I$ is prefix-closed that for all j

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

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 with respect to r, when all time ticks are removed. Consequently, to make sure that $(***)$ holds, it is enough to show that $(**)$ holds. Since

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

implies

$$I[^{i}_{\diamond t_1}], \qquad\qquad I[^{i}_{\diamond t_j}] \Rightarrow I[^{i}_{\diamond t_{j+1}}],$$

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

$$\begin{array}{l}
\textsf{Rule 2}: \\
\textsf{safe}(\lambda x : I) \\
I[^{x}_{\epsilon}] \\
I \wedge R_{S_2} \Rightarrow I[^{x}_{o}] \\
I[^{x}_{o}] \wedge R_{S_2}[^{x}_{o}] \Rightarrow R_{S_1} \\
\hline
S_1 \ \langle i : o \rangle \rightsquigarrow \mu\, S_2 \ \langle i\, , x : o \rangle
\end{array}$$

An interesting question at this point is of course: how strong is Rule 2? We start by showing that the invariant is really needed — needed in the sense that its degenerated version is strictly weaker.

Example 8. Given the wtd-specification

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

From $R_{S_2}[{}^x_o]$ we can deduce only that $\mathsf{ft}(o) = 1$. Let $I \overset{\mathsf{def}}{=} x \in \{1\}^\omega$, using Rule 2 we may deduce that

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

where $R_{S_1} \overset{\mathsf{def}}{=} \#o \geq 1 \wedge o \in \{1\}^\omega$.

Although Rule 2 is stronger than its degenerated version, it is not as strong as we would have liked. To see that consider the following example:

Example 9. Given $K = \{1, 2, \surd\}$. Let $\tau \in K^\omega \xrightarrow{cp} K^\omega$ be a function such that:

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

Let

$$R_{S_2} \overset{\mathsf{def}}{=} o = 1 \,\&\, 2^\infty \vee (\exists x' : x = 1 \,\&\, 1 \,\&\, x' \wedge o = x) \vee (x \neq \epsilon \wedge o = 1 \,\&\, x),$$

then $S_2\ \langle x : o \rangle \equiv R_2$ is the strongest wtd-specification such that $\tau \in [\![\ S_2\ ]\!]$. 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

$$\mathsf{safe}(\lambda x : I), \qquad I[{}^x_\epsilon], \qquad I \wedge R_{S_2} \Rightarrow I[{}^x_o].$$

Moreover, $I[{}^x_o] \wedge R_{S_2}[{}^x_o]$ 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 $[\![\ S_2\ \langle x : o \rangle\ ]\!]$. For example, there is no $\tau' \in [\![\ S_2\ \langle x : o \rangle\ ]\!]$ such that

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

To see that, let r' be a finite 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_{S_2}. Thus, Rule 2 is too weak in the sense that it does not allow us to remove all "solutions" for which there are no corresponding functions in $[\![\ S_2\ \langle x:o\rangle\ ]\!]$.

We now explain how Rule 2 can be strengthened. We first make two observations. For any timed, pulse-driven, stream processing function $\tau \in (T^{\omega})^n \xrightarrow{cp} (T^{\omega})^m$, where $m < n$:

- if r is τ's Kleene-chain with respect to the input s then

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

  where w is an m-tuple whose components are equal to $\surd^{\infty}$.
- if

  $$\tau'(i) \stackrel{\mathsf{def}}{=} \tau(i)|_{\#i+1},$$

  then τ and τ' have exactly the same fixpoints with respect to complete inputs, and τ' satisfies a wtd-specification S iff τ satisfies S. Moreover, τ' has a Kleene-chain consisting of tuples of only finite streams.

In Rule 2, the task of I is to characterize the elements of the Kleene-chains with their corresponding least upper bounds. Thus we may use the two observations above to strengthen the first and the third premise, in which case we get the following rule:

Rule 3 :

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

c varies over chains, and it is assumed that x represents a list of m stream identifiers. As before, I is a formula with the elements of i and x as its only free variables. See [6] for a soundness proof.
Rule 3 solves the problem of Ex. 9, if we choose $x \sqsubseteq 1 \frown 2^{\infty} \vee x \in \{1\}^{\omega}$ as the invariant.

In the case of std-specifications, the rule for the feedback operator has only one premise.

Rule 4 :

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

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 stream tuple s, the equation $\tau(s, r) = r$ has a unique, infinite solution r, an invariant is not needed. Thus there are no additional fixpoints to be eliminated.

It is straightforward to formulate rules which allow one type of specification to be refined by another type of specification. For example Rule 5 characterizes under what conditions a wtd-specification can be refined by a ti-specification, and Rule 6 allows an std-specification to be refined into a wtd-specification.

Rule 5 :

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

Rule 6 :

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

Together with the more general feedback rules given in [6] these conversion rules allow the development of networks consisting of components described by ti-, wtd- as well as std-specifications.

To discuss the completeness of the feedback rules we introduce three classes of components. Given that

$$\sigma \subseteq (D^\omega)^n \overset{c}{\rightarrow} (D^\omega)^m, \qquad\qquad \delta \subseteq (T^\omega)^n \overset{cp}{\rightarrow} (T^\omega)^m,$$

then $ti(\sigma)$, $wtd(\delta)$ and $std(\delta)$ are components whose denotations are characterized by

$$\{\tau \in (T^\omega)^n \overset{cp}{\rightarrow} (T^\omega)^m \mid \exists f \in \sigma : \forall r \in (T^\omega)^n : \diamond\tau(r) = f(\diamond r)\},$$

$$\{\tau \in (T^\omega)^n \overset{cp}{\rightarrow} (T^\omega)^m \mid \\ \exists f \in \delta : \forall r \in (T^\infty)^n : \exists r' \in (T^\infty)^n : \diamond r = \diamond r' \wedge \diamond\tau(r) = \diamond f(r')\},$$

$$\{\tau \in (T^\omega)^n \overset{cp}{\rightarrow} (T^\omega)^m \mid \exists f \in \delta : \forall r \in (T^\infty)^n : \tau(r) = f(r)\},$$

respectively. These three classes of components are of course not disjoint. Nevertheless, we refer to them as the classes of ti-, wtd- and std-components, respectively.

Since components are assigned the same type of semantics as specifications, $\rightsquigarrow$ and μ can be generalized in an obvious way. We then claim that:

1. If $S_1 \ (i : o) \rightsquigarrow \mu \, ti(\{f\})$ then we may formulate a specification $S_2 \ (i \, , x : o)$ and an invariant I such that the premises of Rule 1 are valid and $S_2 \ (i \, , x : o) \rightsquigarrow ti(\{f\})$.

2. If $S_1 \; \langle i : o \rangle \rightsquigarrow \mu \, wtd(\{\tau\})$ then we may formulate a specification $S_2 \; \langle i \, , x : o \rangle$ and an invariant I such that the premises of Rule 3 are valid and $S_2 \; \langle i \, , x : o \rangle \rightsquigarrow wtd(\{\tau\})$.
3. If $S_1 \; \{i : o\} \rightsquigarrow \mu \, std(\delta)$ then we may formulate a specification $S_2 \; \{i \, , x : o\}$ such that the premise of Rule 4 is valid and $S_2 \; \{i \, , x : o\} \rightsquigarrow std(\delta)$.

With respect to (1), let

$$R_{S_2} \overset{\mathsf{def}}{=} f(i, x) = o,$$

$$I \overset{\mathsf{def}}{=} x \sqsubseteq \mu \, f(i).$$

The proof is then straightforward since I holds for the least fixpoints only.

With respect to (2), let

$$R_{S_2} \overset{\mathsf{def}}{=} \exists i' \in (T^\infty)^{n-m} : \exists x' \in (T^\infty)^m : i = \diamond i' \wedge x = \diamond x' \wedge o = \diamond \tau(i', x'),$$

$$I \overset{\mathsf{def}}{=} \exists j : I_j \vee I_\infty,$$

where

$$I_1 \overset{\mathsf{def}}{=} x = \epsilon,$$

$$I_{j+1} \overset{\mathsf{def}}{=} \exists x' : I_j[^x_{x'}] \wedge x' \in (D^*)^m \wedge \exists o : x' \sqsubset x \sqsubseteq o \wedge R_{S_2}[^x_{x'}],$$

$$I_\infty \overset{\mathsf{def}}{=} \exists c : \forall j : I_j[^x_{c_j}] \wedge c_j \in (D^*)^m \wedge \exists o : R_{S_2}[^x_{c_j}] \wedge c_{j+1} \sqsubseteq o \wedge x = \sqcup c.$$

A proof can be found in [6].

With respect to (3), let

$$R_{S_2} \overset{\mathsf{def}}{=} \exists \tau \in \delta : \tau(i, x) = o$$

The proof is then straightforward since any timed, pulse-driven, stream processing function has a unique fixpoint for any complete input.

The result for std-specifications corresponds to relative, semantic completeness with respect to std-components. In the case of ti- and wtd-specifications we can prove relative, semantic completeness only with respect to restricted sets of ti- and wtd-components, respectively — namely with respect to the sets of all components $ti(\sigma)$ and $wtd(\delta)$ where σ and δ contain only one function.

5 General Specifications

With respect to the rules for ti- and wtd-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 [2].

Let $K = \{1, \surd\}$. To investigate the issue, (inspired by [4]) we define three timed, pulse-driven, stream processing functions $\tau_1, \tau_2, \tau_3 : K^\omega \overset{cp}{\rightarrow} K^\omega$, such that

$$\begin{array}{lll} & \tau_2(i) = 1 \,\&\, g_2(i), & \\ \tau_1(i) = 1 \,\&\, g_1(i), & \text{where} & \tau_3(i) = \surd \,\&\, g_3(i), \\ \text{where} & g_2(\surd \,\&\, i) = \surd \,\&\, g_2(i), & \text{where} \\ g_1(\surd \,\&\, i) = \surd \,\&\, g_1(i), & g_2(1 \,\&\, i) = \surd \,\&\, h_2(i), & g_3(\surd \,\&\, i) = \surd \,\&\, g_3(i), \\ g_1(1 \,\&\, i) = 1 \,\&\, \surd^{\#i}, & h_2(\surd \,\&\, i) = \surd \,\&\, h_2(i), & g_3(1 \,\&\, i) = 1 \,\&\, 1 \,\&\, \surd^{\#i}. \\ & h_2(1 \,\&\, i) = 1 \,\&\, \surd^{\#i}, & \end{array}$$

Given a wtd-specification $S\ \langle i : o \rangle$. It is easy to see that

$$\{\tau_2, \tau_3\} \subseteq [\![\ S\ \langle i : o \rangle\ ]\!] \Rightarrow \tau_1 \in [\![\ S\ \langle i : o \rangle\ ]\!].$$

Thus any wtd-specification with τ_2 and τ_3 in its denotation has also τ_1 in its denotation. This is no problem as long as there is no observable behavior of $wtd(\{\tau_1\})$ that it is not also an observable behavior of $wtd(\{\tau_2, \tau_3\})$. Unfortunately, since

$$\tau \in [\![\ wtd(\{\tau_1\})\ ]\!] \Rightarrow \diamond\mu\, \tau = \langle 1, 1 \rangle,$$

$$\tau \in [\![\ wtd(\{\tau_2, \tau_3\})\ ]\!] \Rightarrow \diamond\mu\, \tau \in \{\epsilon, \langle 1 \rangle\},$$

this is not the case, because when we apply the μ operator to $wtd(\{\tau_1\})$ we get $\langle 1, 1 \rangle$ as output stream, and when we apply the μ-operator to $wtd(\{\tau_2, \tau_3\})$ we get either $\langle 1 \rangle$ or ϵ as output stream. Consequently, there is no sound and compositional proof system for wtd-specifications, which allows us to prove that $\mu\, wtd(\{\tau_2, \tau_3\})$ cannot produce $\langle 1, 1 \rangle$, because any wtd-specification fulfilled by $wtd(\{\tau_2, \tau_3\})$ is also fulfilled by $wtd(\{\tau_1\})$, and $wtd(\{\tau_1\})$ does not satisfy the property we want to prove. This explains why in the case of wtd-specifications we could not formulate a rule for the μ-operator, which satisfies the same "strong" completeness result with respect to wtd-components as we could in the case of std-specifications with respect to std-components.

It is easy to show that ti-specifications suffer from a similar expressiveness problem. 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 modeling the nondeterministic choices taken inside a component is added. We use the same tagging convention as before to distinguish ti- and wtd-specifications:

$$S\ (i\,{:}\,o\,{:}\,p) \equiv R\ \rhd\ P, \qquad\qquad S\ \langle i\,{:}\,o\,{:}\,p\rangle \equiv R\ \rhd\ P.$$

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 repetition free list of identifiers (disjoint from the elements of i and o) 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 [1], an oracle in [7], 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:

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

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

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 wtd-component $wtd(\delta)$, then

$$S\ \langle i\,{:}\,o\,{:}\,p\rangle \equiv \exists i' \in (T^{\infty})^{n} : i = \diamond i' \wedge o = \diamond p(i')\ \rhd\ p \in [\![\ wtd(\delta)\ ]\!]$$

is a general wtd-specification, whose denotation is equal to $[\![\ wtd(\delta)\ ]\!]$.

6 Refinement of General Specifications

The definitions of $\leadsto$ and μ carry over straightforwardly. The rules are also easy to generalize. We give only the general version of Rule 2:

Rule 7 :

$$\frac{\begin{array}{l} P_{S_1} \Rightarrow \mathsf{safe}(\lambda x : I) \\ P_{S_1} \Rightarrow I[^x_\epsilon] \\ P_{S_1} \wedge I \wedge R_{S_2} \Rightarrow I[^x_o] \\ P_{S_1} \wedge I[^x_o] \wedge R_{S_2}[^x_o] \Rightarrow R_{S_1} \end{array}}{S_1 \ \langle i : o : p \rangle \rightsquigarrow \mu \, S_2 \ \langle i \, , x : o : p \rangle}$$

In Rule 7 the specifications are assumed to have identical prophecy formulas. The invariants may now also refer to prophecies.

With respect to ti- and wtd-components, the rules for general ti- and wtd-specifications satisfy the same strong completeness results as the rule for std-specifications with respect to std-components — namely what is normally referred to a semantic, relative completeness.

7 Conclusions

Relational specifications have proved to be well-suited for the description of sequential programs. Prominent techniques like Hoare's assertion method, Dijkstra's wp-calculus, or Hehner's predicative specifications 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 [2], specifications where the relationship between the input and the output streams is 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 by 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 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 wtd-specifications. 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 of a ti-specification since it is enough to construct an ordinary (untimed) stream processing function and prove that it satisfies the specification. To prove consistency of a wtd-specification it is in general 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 classes of specifications can be exploited in the very same system development. In fact, using the more general feedback operators of [6] we may build networks consisting of both ti-, wtd- and std-specifications.

Our approach is related to Park's proposals in [9]. 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 [10] are basically the rules for ti-specifications given above. In fact, this paper shows how the refinement calculus and specification technique given in [10] 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

1. Abadi, M., Lamport, L.:
   The Existence of Refinement Mappings.
   Tech. Report 29, Digital, Palo Alto, (1988)
2. Brock, J. D., Ackermann, W. B.:
   Scenarios: A Model of Non-determinate Computation.
   Proc. Formalization of Programming Concepts, LNCS 107, (1981) 252-259
3. Broy, M.:
   Towards a Design Methodology for Distributed Systems.
   Proc. Constructive Methods in Computing Science, Springer, (1989) 311-364

4. Broy, M.:
Functional Specification of Time Sensitive Communicating Systems.
Proc. Programming and Mathematical Method, Springer, (1992) 325-367
5. Broy, M., Dederichs, F., Dendorfer, C., Fuchs, M., Gritzner, T. F., Weber, R.:
The Design of Distributed Systems — An Introduction to Focus.
Tech. Report SFB 342/2/92 A, TU München (1992)
6. Broy, M., Stølen, K.:
Specification and Refinement of Finite Dataflow Networks — a Relational Approach.
Tech. Report SFB 342/7/94 A, TU München (1994)
7. Keller, R. M.:
Denotational Models for Parallel Programs with Indeterminate Operators.
Proc. Formal Description of Programming Concepts, North-Holland, (1978) 337-366
8. Kleene, S. C.:
Introduction to Metamathematics. (1952)
9. Park, D.:
The "Fairness" Problem and Nondeterministic Computing Networks.
Proc. 4th Foundations of Computer Science, Mathematical Centre Tracts 159, Mathematisch Centrum Amsterdam, (1983) 133-161
10. Stølen, K., Dederichs, F., Weber, R.:
Assumption/Commitment Rules for Networks of Asynchronously Communicating Agents.
Tech. Report SFB 342/2/93 A, TU München (1993)