Assumption/Commitment Rules for Dataflow Networks — with an Emphasis on Completeness

Ketil Stølen

Institut für Informatik, TU München, D-80290 München, Germany
email:stoelen@informatik.tu-muenchen.de

Abstract. During the last 15 years a large number of specification techniques based on the so-called assumption/commitment paradigm have been proposed. The formulation of verification rules for the composition of such specifications is known to be a difficult task. Most rules published so far impose strong constraints on the type of properties that can be expressed by the assumptions. Moreover, if completeness results are provided at all they are normally quite weak. We investigate these problems in the context of a model for dataflow networks.

1 Introduction

An *assumption/commitment specification* can be thought of as a pair of predicates (A, C), where the *assumption* A describes the environment in which the specified component is supposed to run, and the *commitment* C states requirements which any correct implementation must fulfill whenever it is executed in an environment which satisfies the assumption. The actual formulation of assumption/commitment specifications is highly dependent on the underlying communication paradigm. See [MC81], [Jon83], [Pnu85], [Sta85], [AL90], [Pan90], [PJ91], [AL93], [SDW93], [Col94] for examples of specification techniques based on the assumption/commitment paradigm.

The formulation of verification rules for the composition of assumption/commitment specifications is a non-trivial issue. The main reason is that the component specifications can be mutually dependent — a fact which easily leads to circular reasoning. Nevertheless, a large number of rules have been proposed. In the sequel we refer to such verification rules as *assumption/commitment rules*.

Most rules published so far impose strong constraints on the properties that can be expressed by the assumptions. For example, it is usual to require that the assumptions are safety properties [Jon83], [AL90], [PJ91], or admissible [SDW93]. Moreover, if the rules are published with completeness results, these results are normally quite weak. For example, it is usual to prove some variation of *relative completeness* [Stø91], [Col94] — a result which only captures some of the expectations we have to an assumption/commitment rule. We study these problems in the context of a model for dataflow networks.

The semantic model is introduced in Sect. 2. We distinguish between two formats for assumption/commitment specifications, namely the *simple* and the *general format*. The simple format is a special case of the general. The simple format can be used only when the assumption is independent of the behavior of the specified component. The simple format is the subject of Sect. 3, and Sect. 4 is devoted to the general format. For both formats we propose assumption/commitment rules with respect to a feedback operator. We prove that these rules are sound, and, moreover, that they are complete in a certain strong sense. In Sect. 5 we show how these rules can be generalized to handle parallel composition of dataflow networks. A small example is presented in Sect. 6. Finally, in Sect. 7 we give a brief summary and relate our results to results known from the literature.

2 Semantic Model

We model the communication history of a channel by a *timed stream*. A timed stream is a finite or infinite sequence of messages and time ticks. A time tick is represented by $\surd$. In any timed stream the interval between two consecutive ticks represents the same least unit of time. A tick occurs in a stream at the end of each time unit. An infinite timed stream represents a complete communication history of a channel, a finite timed stream represents a partial communication history of a channel. Since time never halts, any infinite timed stream is required to contain infinitely many ticks. Moreover, since we do not want a stream to end in the middle of a time unit, we require that any timed stream is either empty, infinite or ends with a tick.

By N, N_+, N_∞ and B we denote respectively the natural numbers, $\mathsf{N} \setminus \{0\}$, $\mathsf{N} \cup \{\infty\}$ and the Booleans. Given a set D of messages, D^ω denotes the set of all finite and infinite timed streams over D; D^∞ denotes the subset consisting of only infinite timed streams. Given a timed stream s and $j \in \mathsf{N}_\infty$, $s\downarrow_j$ denotes the shortest prefix of s containing j ticks if j is less than the number of ticks in s, and s otherwise. Note that $s\downarrow_\infty = s$. This operator is overloaded to tuples of timed streams in a point-wise style, i.e., for any tuple of timed streams t, $t\downarrow_j$ denotes the tuple we get by applying $\downarrow_j$ to each component of t. By $\sqsubseteq$ we denote the usual prefix ordering on streams. Thus, $s \sqsubseteq r$ iff the stream s is a prefix of (or equal to) the stream r. Also this operator is overloaded to tuples of timed streams in a point-wise way, i.e., given two n-tuples of streams t and v, $t \sqsubseteq v$ iff each component of t is a prefix of the corresponding component of v.

Given two tuples a and c consisting of n respectively m streams, by $a \cdot c$ we denote the tuple consisting of $n+m$ streams having a as a prefix and c as a suffix.

A function $\tau \in (D^\infty)^n \to (D^\infty)^m$ is *guarded* iff

$$i\downarrow_j = s\downarrow_j \Rightarrow \tau(i)\downarrow_{(j+1)} = \tau(s)\downarrow_{(j+1)}.$$

That a function is guarded means that the input until time j completely determines the output until time $j+1$. The arrow $\xrightarrow{g}$ is used to distinguish guarded functions from functions that are not guarded.

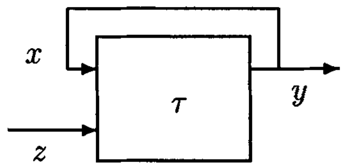

Fig. 1. $\mu\,\tau$

Given a guarded function $\tau \in (D^\infty)^n \xrightarrow{g} (D^\infty)^m$, where $n \geq m$, let $\mu\,\tau$ be the function we obtain by fixing the m last input streams to be equal to the m output streams, i.e., with respect to Fig. 1, by connecting the m output channels y to the m last input channels x. We refer to μ as the *feedback operator*. Formally:

$$\mu\,\tau(z) = y \qquad \Leftrightarrow \qquad \tau(z \cdot y) = y.$$

Due to the guardedness it is easy to prove that for any z there is a unique[1] y such that $\tau(z \cdot y) = y$. This means that $\mu\,\tau$ is well-defined. Moreover, it is also straightforward to verify that $\mu\,\tau$ is guarded. For any set of functions $F \subseteq (D^\infty)^n \xrightarrow{g} (D^\infty)^m$, $\mu\,F$ denotes $\{\mu\,\tau \mid \tau \in F\}$.

Throughout this paper, unless anything else is explicitly stated, any free occurrence of i, o, z or y in a formula should be understood to be universally quantified over tuples of infinite timed streams. Moreover, any free occurrence of j should be understood to be universally quantified over N_∞.

3 Simple Assumption/Commitment Specifications

A *simple assumption/commitment specification* of a component with n input channels and m output channels is a pair (A, C), where A and C are predicates on tuples of timed streams:

$$A \in (D^\infty)^n \to \mathsf{B}, \qquad C \in (D^\infty)^n \times (D^\infty)^m \to \mathsf{B}.$$

A and C characterize the *assumption* and the *commitment*, respectively.

The *denotation* $[\![\,(A, C)\,]\!]$ of a simple assumption/commitment specification (A, C) is the set of all type-correct, guarded functions that behave in accordance with the commitment for any input history satisfying the assumption. Mathematically expressed:

$$\{\tau \in (D^\infty)^n \xrightarrow{g} (D^\infty)^m \mid \forall i : A(i) \Rightarrow C(i, \tau(i))\}.$$

[1] As a consequence of Banach's fixpoint theorem [AdBKR89], since guarded functions can be understood as contracting functions in a complete metric space.

For any specification S, we use A_S and C_S to denote its assumption and commitment, respectively.

The feedback operator μ is lifted from guarded functions to specifications in the obvious way: $[\![\mu(A,C)]\!] \stackrel{\text{def}}{=} \mu [\![(A,C)]\!]$. A specification S_2 *refines* a specification S_1 iff $[\![S_2]\!] \subseteq [\![S_1]\!]$. We then write $S_1 \leadsto S_2$. Since any behavior of S_2 is required to be a behavior of S_1, this concept of refinement is normally referred to as *behavioral refinement.*

We now formulate an assumption/commitment rule with respect to the feedback operator. To simplify the rule, for any predicate $P \in (D^\infty)^n \to \mathsf{B}$, let $\langle P \rangle$ denote the element of $(D^\omega)^n \to \mathsf{B}$ such that:

$$\forall r \in (D^\omega)^n : \langle P \rangle(r) \Leftrightarrow \exists s \in (D^\infty)^n : r \sqsubseteq s \wedge P(s).$$

Note that $\forall s \in (D^\infty)^n : \langle P \rangle(s) \Leftrightarrow P(s)$. The following rule is obviously sound:[2]

Rule 1 :

$$\frac{A_1(z) \wedge (A_2(z \cdot y) \Rightarrow C_2(z \cdot y, y)) \Rightarrow C_1(z,y)}{(A_1, C_1) \leadsto \mu(A_2, C_2)}$$

However, this rule is not very helpful from a practical point of view. Firstly, it only translates the conclusion into the underlying logic without giving much hint about how a proof should be constructed. Secondly, the rule is too weak because the overall commitment C_1 is also required to hold for any z, y such that $A_1(z) \wedge \neg A_2(z \cdot y)$.

By introducing an *invariant* $I \in (D^\infty)^q \times (D^\omega)^m \to \mathsf{B}$ a more useful rule can be formulated:[3]

Rule 2 :

$$\frac{\begin{array}{l} A_1(z) \Rightarrow I(z, y{\downarrow_0}) \\ I(z, y{\downarrow_j}) \Rightarrow \langle A_2 \rangle(z \cdot y{\downarrow_j}) \\ I(z, y{\downarrow_j}) \wedge \langle C_2 \rangle(z \cdot y{\downarrow_j}, y{\downarrow_{(j+1)}}) \Rightarrow I(z, y{\downarrow_{(j+1)}}) \\ \forall k \in \mathsf{N} : I(z, y{\downarrow_k}) \Rightarrow I(z,y) \\ I(z,y) \wedge C_2(z \cdot y, y) \Rightarrow C_1(z,y) \end{array}}{(A_1, C_1) \leadsto \mu(A_2, C_2)}$$

It follows from the first premise that the invariant holds initially. By induction on j, it then follows from the second and third premise that the invariant holds

[2] With respect to Fig. 1, z represents the q external input channels, and y represents the m output channels which are also fed back to x. Throughout this paper we refer to A_2 and A_1 as the component assumption and the overall assumption, respectively (and accordingly for the commitments).

[3] It is here assumed that z and y vary over q- respectively m-tuples of infinite timed streams, and that each free occurrence of j varies over N_∞. It is also assumed that $\downarrow$ binds stronger than $\cdot$.

at any finite time, in which case the fourth premise implies that the invariant holds at infinite time. The conclusion then follows by the fifth premise.

Note that the third premise is a tautology for $j = \infty$. Note also that we have not imposed any constraints on the type of properties that can be expressed by the assumptions. Rule 2 allows *all* environment restrictions to be listed in the assumptions independent of whether these restrictions are safety properties or not. Moreover, although the prefix closures of A_2 and C_2 must be constructed, the rule does not depend on that the assumptions are split into safety and liveness properties.

It can be proved that Rule 2 is *relative (semantic) complete* with respect to components modeled by non-empty sets of guarded functions.

Proposition 1. *Let $F \subseteq (D^\infty)^{(q+m)} \xrightarrow{g} (D^\infty)^m$ be nonempty and assume that $\mu F \subseteq [\![S_1]\!]$. Then there is a specification S_2 and a predicate $I \in (D^\infty)^q \times (D^\omega)^m \to \mathsf{B}$ such that the five premises of Rule 2 are valid and $F \subseteq [\![S_2]\!]$.*

Proof. Let $A_{S_2}(z \cdot x) \stackrel{\text{def}}{=} \mathsf{true}, C_{S_2}(z \cdot x, y) \stackrel{\text{def}}{=} \exists \tau \in F : \tau(z \cdot x) = y, I(z, y) \stackrel{\text{def}}{=} A_1(z)$. The validness of the first four premises follows trivially. That the fifth premise is valid follows from the fact that each guarded function has a unique fix-point with respect to μ.

The completeness result characterized by Prop. 1 just says that whenever we have a dataflow network μF, which satisfies some overall specification S_1, then we can construct a specification S_2, which is satisfied by F, and use Rule 2 to prove that $S_1 \rightsquigarrow \mu S_2$. Since we are free to construct S_2 as we like, this is a weak completeness result. As shown by the proof, true can be used as component assumption, and the overall assumption suffices as invariant. Since the first four premises are trivially equivalent to true this property does not test the special features of Rule 2. In fact, also Rule 1 is complete in this sense. Thus, it is clear that Prop. 1 only captures some of the expectations we have to an assumption/commitment rule.

Before we can prove a more interesting result, we have to figure out exactly what these expectations are. First of all, we do not expect opposition when we claim that, from a practical point of view, an assumption/commitment rule is only expected to work when all specifications concerned are implementable. For example (true, false) is not a very interesting specification because any component behavior is disallowed[4]. This specification is obviously inconsistent in the sense that its denotation is empty, and it is clearly not implementable (modulo our concept of refinement $\rightsquigarrow$ if components are modeled by non-empty sets of guarded functions). In fact, any specification which disallows any component behavior for at least one input history satisfying the assumption is trivially not implementable.

[4] Remember that the complete communication history of a channel along which no message is sent is an infinite sequence of ticks. Thus, this specification also disallows the empty behavior — the behavior of a component that does nothing.

This is not, however, the only way in which a simple assumption/commitment specification can be unimplementable — it can also be unimplementable because it disallows guardedness. We say that a simple assumption/commitment specification S is *consistent* if $[\![S]\!] \neq \emptyset$.

A consistent, simple assumption/commitment specification may have a commitment that is *not fully realizable* with respect to complete input histories satisfying the assumption or partial input histories that have not yet falsified the assumption.

Example 1. Consider the specification S, where $A_S(i) \overset{\text{def}}{=} \mathsf{true}$ and $C_S(i, o) \overset{\text{def}}{=} o = \langle \surd \rangle^\infty \vee i = o = \langle 1, \surd \rangle^\infty$. It is here assumed that $\langle a_1, a_2, .., a_n \rangle$ denotes the stream consisting of n elements whose first element is a_1, whose second element is a_2, and so on. Moreover, for any stream s, s^∞ denotes the stream consisting of infinitely many copies of s. Since $\lambda i.\langle \surd \rangle^\infty \in [\![S]\!]$, it follows that S is consistent.

To see that the commitment is not fully realizable with respect to input histories satisfying the assumption, let $\tau \in [\![S]\!]$. Since $\langle \surd \rangle^\infty \downarrow_0 = \langle 1, \surd \rangle^\infty \downarrow_0$, the guardedness of τ implies $\tau(\langle \surd \rangle^\infty)\downarrow_1 = \tau(\langle 1, \surd \rangle^\infty)\downarrow_1$, in which case it follows from the formulation of S that $\tau(\langle \surd \rangle^\infty) = \langle \surd \rangle^\infty = \tau(\langle 1, \surd \rangle^\infty)$. Thus, the second disjunct of the commitment is not realizable by any guarded function (and therefore also not realizable by any implementation modulo $\rightsquigarrow$).

Such specifications can be avoided by requiring that:

$$\langle A_S \rangle(i\downarrow_j) \wedge \langle C_S \rangle(i\downarrow_j, o\downarrow_{(j+1)}) \Rightarrow \exists \tau \in [\![S]\!] : \tau(i)\downarrow_{(j+1)} = o\downarrow_{(j+1)}.$$

Thus, at any time j, if the environment assumption has not yet been falsified, then any behavior allowed by the commitment until time $j + 1$ is matched by a function in the specification's denotation. We say that a simple specification is *fully realizable* if it satisfies this constraint. Note that only unrealizable paths are eliminated by this constraint. It does not reduce the set of liveness properties that can be expressed by the assumption or the commitment.

Nevertheless, from a practical point of view, any claim that simple specifications should always be fully realizable is highly debatable. Of course, when someone comes up with a specification as the one in Ex. 1, it is most likely true that he has specified something else than he intended to specify. However, there are other situations where specifications that are not fully realizable can be simpler and more intuitive than their fully realizable counterparts.

Example 2. Consider the specification S, where $A_S(i) \overset{\text{def}}{=} \mathsf{true}$ and $C_S(i, o) \overset{\text{def}}{=} \bar{i} = \bar{o}$. For any timed stream s, by $\bar{s}$ we denote the result of removing all ticks in s. Since S allows behaviors where messages are output before they are received, or without the required delay of at least one time unit, S is not fully realizable. For example, let $i = a \frown \langle \surd \rangle^\infty$ and $o = a \frown \langle \surd \rangle^\infty$, where a is a message and the operator $\frown$ is used to extend a stream with a new first element (later it will also be used to concatenate streams). Assume there is a $\tau \in [\![S]\!]$ such that $\tau(i) = o$. We prove that this assumption leads to a contradiction. The commitment implies $\tau(\langle\surd\rangle^\infty) = \langle\surd\rangle^\infty$. Since $i{\downarrow}_0 = \langle\surd\rangle^\infty{\downarrow}_0$ it follows that τ is not guarded. This contradicts that $\tau \in [\![S]\!]$. The specification S', where

$$A_{S'}(i) \overset{\mathsf{def}}{=} \mathsf{true}, \quad C_{S'}(i,o) \overset{\mathsf{def}}{=} \bar{o} = \bar{\imath} \wedge \forall j \in \mathsf{N} : \overline{o{\downarrow}_{(j+1)}} \sqsubseteq \overline{i{\downarrow}_j},$$

is fully realizable and equivalent to S in the sense that $[\![S]\!] = [\![S']\!]$.

Of course, in this little example it does not really matter. Nevertheless, in non-trivial cases, specifications can be considerably shortened by leaving out constraints already imposed via the semantics.

To check whether a consistent specification (A, C_1) can be refined into a fully realizable specification (A, C_2) is normally easy — it is enough to check that $A \wedge C_2 \Rightarrow C_1$. To check the opposite, namely whether $(A, C_2) \rightsquigarrow (A, C_1)$, can be non-trivial. In that case, so-called adaptation rules are needed. In most practical situations the following adaptation rule is sufficient:

$$\frac{A(i) \wedge (\forall j \in \mathsf{N}_\infty : \forall s : A(i{\downarrow}_j \frown s) \Rightarrow \exists r : C_2(i{\downarrow}_j \frown s, o{\downarrow}_{(j+1)} \frown r)) \Rightarrow C_1(i,o)}{(A, C_1) \rightsquigarrow (A, C_2)}$$

Using this rule it is straightforward to prove that the specification S of Ex. 1 is a refinement of the fully realizable equivalent specification S', where $A_{S'}(i) \overset{\mathsf{def}}{=} \mathsf{true}$ and $C_{S'}(i,o) \overset{\mathsf{def}}{=} o = \langle\surd\rangle^\infty$. With respect to Ex. 2, this adaptation rule can be used to prove that the specification S is a refinement of the equivalent specification S'.

An interesting question at this point is of course: how complete is this adaptation rule — for example, is it *adaptation complete* in the sense that it can be used to refine any consistent, fully realizable specification into any semantically equivalent specification? Unfortunately, the answer is "no".

Example 3. To see that, first note that the specification S, where $A_S(i) \overset{\mathsf{def}}{=} \mathsf{true}$ and $C_S(i,o) \overset{\mathsf{def}}{=} o \neq i$ is inconsistent. To prove this, assume $\tau \in [\![S]\!]$. τ is guarded which implies that τ has a unique fix-point, i.e., there is a unique s such that $\tau(s) = s$. This contradicts that $\tau \in [\![S]\!]$. Moreover, since

$$\exists j \in \mathsf{N}, s : \forall r : o{\downarrow}_{(j+1)} \frown r = i{\downarrow}_j \frown s \Leftrightarrow \mathsf{false},$$

it follows that the adaptation rule cannot be used to adapt S.

A slightly weaker, consistent version of S is characterized by S', where $A_{S'}(i) \overset{\mathsf{def}}{=} \mathsf{true}$ and $C_{S'}(i,o) \overset{\mathsf{def}}{=} o \neq i \vee o = \langle\surd\rangle^\infty$. Since $\lambda i.\langle\surd\rangle^\infty \in [\![S']\!]$ it follows that S' is consistent. That the adaptation rule cannot be used to adapt S' can be proved in the same way as for S. Moreover, since any $\tau \in [\![S']\!]$ has $\langle\surd\rangle^\infty$ as its fix-point it follows from the guardedness of τ that for example any behavior (i, o) such that o does not start with a $\surd$ is not realizable by a function in the denotation of S'. Thus, S' is not fully realizable.

To adapt such specifications without explicitly referring to guarded functions is problematic, if at all possible. However, by referring directly to the denotation of a specification, we get the rule below, which is obviously adaptation complete.

$$\frac{A(i) \wedge \tau \in [\![(A, C_2)]\!] \Rightarrow C_1(i, \tau(i))}{(A, C_1) \rightsquigarrow (A, C_2)}$$

Of course this type of adaptation can also be built into Rule 2. However, in our opinion, assumption/commitment rules should not be expected to be adaptation complete. Firstly, as shown above, by building adaptation into an assumption/commitment rule, the rule becomes more complicated — at least if adaptation completeness is to be achieved. Secondly, for many proof systems, adaptation completeness is not achievable. Roughly speaking, adaptation completeness is only achievable if the assertion language is rich enough to allow the semantics of a specification to be expressed at the syntactic level. For example, with respect to our rules, it seems to be necessary to refer to guarded functions at the syntactic level in order to achieve adaptation completeness. Instead, we argue that assumption/commitment rules should only be expected to work when the specifications are fully realizable. Adaptation should be conducted via separate rules. If these adaptation rules are adaptation complete, then this can be proved. If not, we may still prove that the assumption/commitment rules satisfy interesting completeness properties with respect to fully realizable specifications — which basically amounts to proving these properties under the assumption that adaptation complete adaptation rules are available.

We are by no means the first to make the distinction between adaptation rules and ordinary rules. In fact, since the early days of Hoare-logic, it has been common to distinguish between syntax-directed proof-rules involving composition modulo some programming construct and pure adaptation rules. See for example the discussion on adaptation completeness in [Zwi89].

Given that the specifications are consistent and fully realizable, at a first glance one might expect the completeness property of interest to be:

- whenever the conclusion holds, then we can find an invariant I such that the five premises of Rule 2 are valid.

However, this property is too strong. Consider the single premise of Rule 1. The main contribution of Rule 2 is that whenever the first four premises of Rule 2 are valid, then the premise of Rule 1 can be simplified to:

$$I(z, y) \wedge C_2(z \cdot y, y) \Rightarrow C_1(z, y).$$

The second premise of Rule 2 makes sure that the invariant implies the component assumption A_2. Moreover, Rule 2 allows us to build the overall assumption into the invariant. Thus, this formula is basically "equivalent" to:

$$A_1(x) \wedge A(z \cdot y) \wedge C_2(z \cdot y, y) \Rightarrow C_1(z, y).$$

As a consequence, it can be argued that Rule 2 characterizes sufficient conditions under which $\Rightarrow$ in the antecedent of Rule 1's premise can be replaced by $\wedge$. In other words, the main contribution of Rule 2 with respect to Rule 1 is to make sure that for any overall input history satisfying the overall assumption, the component assumption is not falsified. In fact, this is not only a feature of Rule 2 — it seems to be a feature of assumption/commitment rules for simple specifications. For example, in the rely/guarantee method [Jon83] only simple specifications can be expressed (simple in the sense that the pre- and rely-conditions do not impose constraints on the behaviors of the specified components). Moreover, the rule for parallel composition makes sure that if the environment behaves in accordance with the overall pre- and rely-conditions, then the pre- and rely-conditions of the two component specifications are not falsified.

Thus, since for example $(\mathsf{true}, \mathsf{true}) \rightsquigarrow \mu(\mathsf{false}, \mathsf{true})$, although $[\![(\mathsf{false}, \mathsf{true})]\!]$ contains any type-correct function, the completeness property proposed above is too strong. Instead we propose the following property.

Proposition 2. *Let S_1 and S_2 be simple specifications such that $S_1 \rightsquigarrow \mu S_2$. Assume that S_2 is consistent and fully realizable, and moreover that:*

$$\tau \in [\![S_2]\!] \wedge A_{S_1}(z) \Rightarrow A_{S_2}(z \cdot \mu\tau(z)).$$

Then there is a predicate $I \in (D^{\infty})^q \times (D^{\omega})^m \to \mathsf{B}$ such that the five premises of Rule 2 are valid.

Proof. Given that $\mathrm{tm}(y)$ returns the number of ticks in y, the validness of the five premises follows straightforwardly if $I(z, y)$ is defined as follows:

$$A_1(z) \wedge \langle A_2 \rangle(z, y\downarrow_0) \wedge \forall k \in \mathsf{N}_+ : k \leq \mathrm{tm}(y) \Rightarrow \langle C_2 \rangle(z, y\downarrow_{(k-1)}, y\downarrow_k).$$

Note that Rule 1 does not satisfy this result. The proof of Prop. 2 is based on the fact that there is a *canonical invariant* — more precisely, a schema that gives an invariant that is sufficiently strong. By inserting this invariant in Rule 2 and conducting some straightforward simplifications, we obtain the following rule:[5]

Rule 2′ :

$$\frac{\begin{array}{l} A_1(z) \Rightarrow \langle A_2 \rangle(z \cdot y\downarrow_0) \\ A_1(z) \wedge \langle C_2 \rangle(z \cdot y\downarrow_j, y\downarrow_{(j+1)}) \Rightarrow \langle A_2 \rangle(z \cdot y\downarrow_{(j+1)}) \\ A_1(z) \wedge \forall k \in \mathsf{N} : \langle C_2 \rangle(z \cdot y\downarrow_k, y\downarrow_k) \Rightarrow A_2(z \cdot y) \\ A_1(z) \wedge C_2(z \cdot y, y) \Rightarrow C_1(z, y) \end{array}}{(A_1, C_1) \rightsquigarrow \mu(A_2, C_2)}$$

Rule 2 and 2′ are equivalent modulo the canonical invariant. From a practical point of view, if we are working outside the scope of automatic verification,

[5] Contrary to earlier, j varies over N.

it is debatable whether the invariant should be fixed in this way. A canonical invariant has a simplifying effect in the sense that the user himself does not have to come up with the invariant. On the other hand, it complicates the reasoning because it is then necessary to work with a large and bulky formula when in most cases a much simpler formula is sufficient.

4 General Assumption/Commitment Specifications

A *general assumption/commitment specification* is also a pair of two predicates (A, C). The difference with respect to the simple case is that not only the commitment, but also the assumption A, may refer to the output, i.e., A is now of the same type as C:

$$A \in (D^{\infty})^n \times (D^{\infty})^m \to \mathsf{B}.$$

The *denotation* $[\![\, (A, C) \,]\!]$ of a general assumption/commitment specification (A, C) is defined as follows:

$$\{\tau \in (D^{\infty})^n \xrightarrow{g} (D^{\infty})^m \mid \forall i, j : \langle A \rangle (i{\downarrow}_{(j+1)}, \tau(i){\downarrow}_j) \Rightarrow \langle C \rangle (i{\downarrow}_{(j+1)}, \tau(i){\downarrow}_{(j+1)})\}.$$

Note that, since $\langle A \rangle (i{\downarrow}_{(\infty+1)}, \tau(i){\downarrow}_{\infty}) \Rightarrow \langle C \rangle (i{\downarrow}_{(\infty+1)}, \tau(i){\downarrow}_{(\infty+1)})$ is equivalent to $A(i, \tau(i)) \Rightarrow C(i, \tau(i))$, this requirement is at least as strong as the constraint imposed on the denotation of a simple specification. In addition, any correct implementation is required to fulfill the commitment at least *one step longer than* the environment fulfills the assumption. One may ask: why not impose this second constraint also in the case of simple specifications? The reason is that the second constraint degenerates to that for simple specifications when A does not refer to the output.

Rule 2 can now be restated for general specifications as below:[6]

Rule 3 :

$$\frac{\begin{array}{l} A_1(z, y) \Rightarrow I(z, y, y{\downarrow}_0) \\ I(z, y, y{\downarrow}_j) \Rightarrow \langle A_2 \rangle (w{\downarrow}_j, y{\downarrow}_j) \\ I(z, y, y{\downarrow}_j) \wedge \langle C_2 \rangle (w{\downarrow}_j, y{\downarrow}_{(j+1)}) \Rightarrow I(z, y, y{\downarrow}_{(j+1)}) \\ \forall k \in \mathsf{N} : I(z, y, y{\downarrow}_k) \Rightarrow I(z, y, y) \\ \langle I \rangle (z, y{\downarrow}_j, y{\downarrow}_j) \wedge \langle C_2 \rangle (w{\downarrow}_j, y{\downarrow}_{(j+1)}) \Rightarrow \langle C_1 \rangle (z{\downarrow}_{(j+1)}, y{\downarrow}_{(j+1)}) \end{array}}{(A_1, C_1) \rightsquigarrow \mu\, (A_2, C_2)}$$

Contrary to earlier the overall assumption may refer to the overall output. As a consequence, it is enough to require that the invariant and the component

[6] It is here assumed that z and y vary over q- respectively m-tuples of infinite timed streams, that j varies over N_{∞}, and that $w = z \cdot y$. Moreover, $I \in (D^{\infty})^q \times (D^{\infty})^m \times (D^{\omega})^m \to \mathsf{B}$.

assumption hold at least as long as the overall assumption has not been falsified. This motivates the modifications to the first four premises. The fifth premise has been altered to accommodate that for partial input the overall commitment is required to hold at least one step longer than the overall assumption. The one-step-longer-than semantics is needed to prove the induction step.

Rule 3 is relative, semantic complete in the same sense as Rule 2. However, as for simple specifications, this is not the completeness result we want. A general assumption/commitment specification S is *consistent* iff $[\![S]\!] \neq \emptyset$ and *fully realizable* iff for any input history i, output history o and $\tau \in [\![S]\!]$ there is a $\tau' \in [\![S]\!]$ such that:

$$\langle A_S \rangle (i{\downarrow_j}, o{\downarrow_j}) \wedge o{\downarrow_j} = \tau(i){\downarrow_j} \wedge \langle C_S \rangle (i{\downarrow_j}, o{\downarrow_{(j+1)}}) \Rightarrow \tau'(i){\downarrow_{(j+1)}} = o{\downarrow_{(j+1)}}.$$

Note that this constraint degenerates to the corresponding constraint for simple specifications if S is consistent and does not refer to o in its assumption.

In Prop. 2 we made the assumption that for any input history satisfying the overall assumption, each resulting fix-point satisfies the component assumption. In the case of general assumption/commitment specifications the overall assumption may refer to the output. Thus, it makes only sense to require that the component assumption holds at least as long as the overall assumption. Proposition 2 can then be restated as below:

Proposition 3. *Let S_1 and S_2 be general specifications such that $S_1 \rightsquigarrow \mu\, S_2$. Assume that S_2 is consistent and fully realizable, and moreover that:*

$$\tau \in [\![S_2]\!] \wedge \langle A_{S_1} \rangle (z{\downarrow_j}, \mu\, \tau(z){\downarrow_j}) \Rightarrow \langle A_{S_2} \rangle ((z \cdot \mu\, \tau(z)){\downarrow_j}, \mu\, \tau(z){\downarrow_j}).$$

Then there is a predicate $I \in (D^{\infty})^q \times (D^{\infty})^m \times (D^{\omega})^m \to \mathsf{B}$ such that the five premises of Rule 3 are valid.

Proof. The validness of the five premises follows straightforwardly if $I(z, y, v)$ is defined as follows:

$$A_{S_1}(z, y) \wedge \langle A_{S_2} \rangle (w{\downarrow_0}, v{\downarrow_0}) \wedge \forall k \in \mathsf{N}_+ : k \leq \mathrm{tm}(v) \Rightarrow \langle C_2 \rangle (w{\downarrow_{(k-1)}}, v{\downarrow_k}),$$

where $w = z \cdot v$.

By inserting the canonical invariant in Rule 3 we obtain:

Rule 3′ :

$$\frac{\begin{array}{l} A_1(z, y) \Rightarrow \langle A_2 \rangle (w{\downarrow_0}, y{\downarrow_0}) \\ A_1(z, y) \wedge \langle C_2 \rangle (w{\downarrow_j}, y{\downarrow_{(j+1)}}) \Rightarrow \langle A_2 \rangle (w{\downarrow_{(j+1)}}, y{\downarrow_{(j+1)}}) \\ A_1(z, y) \wedge \forall k \in \mathsf{N} : \langle C_2 \rangle (w{\downarrow_k}, y{\downarrow_k}) \Rightarrow A_2(w, y) \\ \langle A_1 \rangle (z, y{\downarrow_j}) \wedge \langle C_2 \rangle (w{\downarrow_j}, y{\downarrow_{(j+1)}}) \Rightarrow \langle C_1 \rangle (z{\downarrow_{(j+1)}}, y{\downarrow_{(j+1)}}) \end{array}}{(A_1, C_1) \rightsquigarrow \mu\, (A_2, C_2)}$$

It is assumed that $w = z \cdot y$. Rule 3 and 3′ are equivalent modulo the canonical invariant.

5 Network Rules

So far specifications have been represented by pairs of predicates. Instead of predicates we now use *formulas* with free variables varying over timed infinite streams. Each free variable represents the communication history of the channel named by the variable. In that case, however, we need a way to distinguish the variables representing input channels from the variables representing output channels. We therefore propose a quadruple (i, o, A, C), where i is a finite, totally ordered set of input identifiers, o is a finite, totally ordered set of output identifiers, and A and C are formulas whose free variables are contained in $i \cup o$. The sets i and o are required to be disjoint. In other words, the input and output channels have different names. The advantage of this representation is that it gives us a flexible way of composing specifications into networks of specifications by connecting input and output channels whose names are identical.

Consider the s general specifications

$$(z_1 \cup x_1, y_1, A_1, C_1),\ (z_2 \cup x_2, y_2, A_2, C_2),\quad \ldots\quad , (z_s \cup x_s, y_s, A_s, C_s).$$

For each l, the sets z_l, x_l and y_l name respectively the *external input channels* (those connected to the overall environment), the *internal input channels* (those connected to the other $s-1$ specifications in the network), and the *external and internal output channels*. Let

$$z = \cup_{l=1}^{s} z_l, \quad x = \cup_{l=1}^{s} x_l, \quad y = \cup_{l=1}^{s} y_l.$$

We assume that $z \cap x = z \cap y = \emptyset$ and that $x \subseteq y$. Moreover, we also assume that $l \neq k \Rightarrow y_l \cap y_k = \emptyset$.
We can think of these s specifications as modeling a network of s components where the input and output channels are connected iff their names are identical. The constraints imposed on the sets of channel identifiers imply that two different specifications cannot send along the same channel. They may receive on the same channel, however, this read access is non-destructive in the sense that they both get a private copy of the channel's content. We represent this network by:

$$\|_{l=1}^{s}\ (z_l \cup x_l, y_l, A_l, C_l).$$

Thus, given that z and y contain n respectively m elements, the denotation of this network is the set of all functions $\tau \in (D^\infty)^n \xrightarrow{g} (D^\infty)^m$, for which there are s functions $\tau_l \in [\![\ (z_l \cup x_l, y_l, A_l, C_l)\ ]\!]$ such that for all z, $\tau(z) = y$ iff[7]

[7] In this definition each totally ordered set is interpreted as a tuple.

$$y_1 = \tau_1(z_1 \cdot x_1),\ y_2 = \tau_2(z_2 \cdot x_2),\ \dots\ ,\ y_s = \tau_s(z_s \cdot x_s).$$

Since each τ_l is guarded, it follows that τ is well-defined and guarded.

For any formula P and set of variables a, $P(a{\downarrow_j})$ denotes the result of replacing each occurrence of v in P by $v{\downarrow_j}$ for any $v \in a$. Moreover, $P(a{\downarrow_j}, b{\downarrow_k})$ is a short-hand for $P(a{\downarrow_j})(b{\downarrow_k})$. A straightforward generalization of Rule 3′ gives:[8]

Rule 4 :

$$\frac{\begin{array}{l} A \Rightarrow \wedge_{l=1}^{s}\langle A_l\rangle(w_l{\downarrow_0}, y_l{\downarrow_0}) \\ A \wedge [\wedge_{l=1}^{s}\langle C_l\rangle(w_l{\downarrow_j}, y_l{\downarrow_{(j+1)}})] \Rightarrow \wedge_{l=1}^{s}\langle A_l\rangle(w_l{\downarrow_{(j+1)}}, y_l{\downarrow_{(j+1)}}) \\ A \wedge \forall k \in \mathbb{N} : \wedge_{l=1}^{s}\langle C_l\rangle(w_l{\downarrow_k}, y_l{\downarrow_k}) \Rightarrow \wedge_{l=1}^{s} A_l \\ \langle A\rangle(y{\downarrow_j}) \wedge [\wedge_{l=1}^{s}\langle C_l\rangle(w_l{\downarrow_j}, y_l{\downarrow_{(j+1)}})] \Rightarrow \langle C\rangle(z{\downarrow_{(j+1)}}, y{\downarrow_{(j+1)}}) \end{array}}{(z, y, A, C) \rightsquigarrow \|_{l=1}^{s}(z_l \cup x_l, y_l, A_l, C_l)}$$

However, this rule ignores one aspect, namely that we are now dealing with s component specifications and not only 1. For example, if $s = 2$, it may be the case that the antecedent of the third premise implies only one of the component assumptions, say A_1, and that the second component assumption A_2 can be deduced from $A_1 \wedge C_1$. This is typically the case if A_2 contains some liveness constraint that can only be deduced from C_1. To accommodate this, we reformulate Rule 4 as below:[9]

Rule 5 :

$$\frac{\begin{array}{l} A \Rightarrow \wedge_{l=1}^{s}\langle A_l\rangle(w_l{\downarrow_0}, y_l{\downarrow_0}) \\ A \wedge [\wedge_{l=1}^{s}\langle C_l\rangle(w_l{\downarrow_j}, y_l{\downarrow_{(j+1)}})] \Rightarrow \wedge_{l=1}^{s}\langle A_l\rangle(w_l{\downarrow_{(j+1)}}, y_l{\downarrow_{(j+1)}}) \\ \langle A\rangle(y{\downarrow_j}) \wedge [\wedge_{l=1}^{s}\langle C_l\rangle(w_l{\downarrow_j}, y_l{\downarrow_{(j+1)}})] \Rightarrow \langle C\rangle(z{\downarrow_{(j+1)}}, y{\downarrow_{(j+1)}}) \\ A \wedge \forall k \in \mathbb{N} : [\wedge_{l=1}^{s}\langle C_l\rangle(w_l{\downarrow_k}, y_l{\downarrow_k})] \wedge [\wedge_{l=1}^{s} A_l \Rightarrow C_l] \Rightarrow C \end{array}}{(z, y, A, C) \rightsquigarrow \|_{l=1}^{s}(z_l \cup x_l, y_l, A_l, C_l)}$$

For Rule 5 we may prove completeness results similar to those for Rule 3′.

6 Example

We now use Rule 5 to prove that the network pictured in Fig. 2 consisting of two component specifications BQ and UQ characterizes an *unbounded FIFO queue*.

We first introduce two stream operators. For any stream s and set A, $A©s$ denotes the substream that can be obtained from s by filtering away all messages (including ticks) not in A. If $\surd \notin A$ then the resulting stream is untimed. If $A = \{m\}$ we write $m©s$ instead of $\{m\}©s$. For any untimed stream s, $\#s$ denotes the number of messages in s.

[8] The elements of z and y vary over D^∞, j varies over $\mathbb{N}_\infty$, and $w_l = z_l \cup x_l$.

[9] Contrary to earlier, j varies over $\mathbb{N}$.

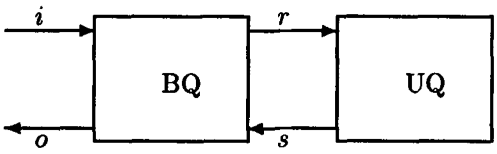

Fig. 2. The FIFO Queue

BQ, the *bounded queue*, differs from UQ, the *unbounded queue*, in that it has a bounded memory. When the internal memory of BQ is used up, it is supposed to use the unbounded queue. Let D be the set of data elements, and let $M = D \cup \{\odot\}$. $\odot$ models a request. We assume that $\odot \notin D$. Formally, the queues are specified as below:[10]

$$\mathrm{BQ} \stackrel{\mathrm{def}}{=} (\{i : M, s : D\}, \{o : D, r : M\}, P(i) \wedge Q(r,s), Q(i,o) \wedge P(r)),$$

$$\mathrm{UQ} \stackrel{\mathrm{def}}{=} (\{r : M\}, \{s : D\}, P(r), Q(r,s)),$$

where

$$P(a) \stackrel{\mathrm{def}}{=} \forall j \in \mathbb{N} : \#[\odot \copyright (a{\downarrow_j})] \leq \#[D \copyright (a{\downarrow_j})],$$

$$Q(a,b) \stackrel{\mathrm{def}}{=} \forall j \in \mathbb{N} : D \copyright (b{\downarrow_{(j+1)}}) \sqsubseteq D \copyright (a{\downarrow_j}) \wedge \#D \copyright b = \#\odot \copyright a.$$

P states that at any point in time the number of received requests is less than or equal to the number of received data elements. Q states that any data element output on b has already been input on a, and that exactly one data element is output on b for each request received on a.

Note that A_{BQ} is not a pure safety assumption. It follows straightforwardly by Rule 5 that $\mathrm{UQ}' \rightsquigarrow \mathrm{BQ} \parallel \mathrm{UQ}$, where UQ' is obtained from UQ by replacing r and s by i and o, respectively.

7 Discussion

As we see it, the contributions of this paper are as follows:

- We have introduced two specification formats for dataflow components.
- For these specification formats we have formulated assumption/commitment rules.
- We have shown that our rules are able to handle assumptions with arbitrary liveness properties.

[10] For simplicity, we do not specify the boundedness constraints imposed on BQ. By $t : T$ we specify that the channel named by t is of type T.

- We have argued that the usual concept of relative completeness captures only some of the expectations we have to such rules. We have carefully investigated exactly what those expectations are, and based on this investigation, we have proposed stronger completeness requirements and proved that our rules satisfy these requirements.
- We have shown that the rules for the feedback construct can be generalized to handle parallel composition of dataflow networks (parallel composition with mutual feedback).

We now relate our approach to approaches known from the literature.

Semantic Model: Park [Par83] employs ticks (hiatons) in the same way as us. However, his functions are defined also for finite streams, and infinite streams are not required to have infinitely many ticks. Kok [Kok87] models components by functions mapping infinite streams of finite streams to non-empty sets of infinite streams of finite streams. The finite streams can be empty which means that he can represent communication histories with only finitely many messages. His infinite streams of finite streams are isomorphic to our timed streams in the sense that we use ticks to split an infinite communication history into an infinite sequence of finite streams. Two consecutive ticks correspond to an empty stream. In the style of [Bro87], we use a set of functions to model nondeterministic behavior. This in contrast to the set valued functions of [Kok87]. Sets of functions allow unbounded nondeterminism (and thereby liveness) to be modeled in an elegant way, i.e., without using a more complicated metric. However, contrary to [Bro87], we use guarded functions and infinite timed streams. Thereby we get a simpler theory. The actual formulation of guardedness has been taken from [Bro95].

Specification Formats: The distinction between simple and general specifications can also be found in [SDW93], [Bro94]. However, in these papers, the techniques for expressing general specifications are more complicated. [SDW93] uses a specification format based on prophecies. [Bro94] employs so-called input-choice specifications. The one-step-longer-than semantics used by us is strongly inspired by [AL93].

Assumption/Commitment Rules: A large number of assumption/commitment rules have been published. Most rules proposed so far impose strong constraints on the properties that can occur in the assumptions. For example, it is common to require the assumptions to be safety properties [AL90], [PJ91] or admissible [SDW93]. An assumption/commitment rule handling general liveness properties in the assumptions can be found in [Pnu85] (related rules are proposed in [Sta85], [Pan90]). However, this rule is based on the $\Rightarrow$ semantics we used for simple specifications. Our rules for general specifications require the stronger one-step-longer-than semantics. Although the rule in [AL93] is intended for safety assumptions, it seems to be a close relationship between this rule and Rule 5. The first premise of Rule 5 is captured by the fact that [AL93] defines the empty sequence to satisfy any formula. The second and third premise are almost of the same form as the first and second premise in [AL93] if the consequent of the first premise in [AL93] is replaced by its safety closure. Finally, the fourth premise of Rule 5 "translates to" $A \wedge (\wedge_{l=1}^{s} \mathrm{Cl}(C_l)) \wedge (\wedge_{l=1}^{s} A_l \Rightarrow C_l) \Rightarrow C$.[11] The "resulting" variant of the rule in [AL93] is sound and deals with liveness assumptions. Nevertheless, it can be deduced from the rule in [AL93] by exploiting the fact that any assumption/commitment specification can be converted to an equivalent specification whose assumption is a safety property.[12]

[AL93] argues that from a pragmatic point of view specifications should always be formulated in such a way that the assumptions are safety properties. Because we have too little experience in using our formalism, we do not take any standpoint to this claim here. However, we have at least shown that our assumption/commitment rules do not depend upon this restriction, and, moreover, that, with respect to our formalism, there are cases where it seems natural to allow assumptions to also impose liveness properties.

Completeness: [Zwi89] distinguishes between three concepts of completeness, namely compositional, adaptation and modular completeness. Roughly speaking, a proof system is compositional complete if it is compositional and relative complete. A proof system is modular complete if it is compositional complete and in addition adaptation complete. Our concept of completeness can almost be understood as modular completeness under the assumption that adaptation complete adaptation rules are available.

8 Acknowledgment

Manfred Broy, Pierre Collette and Stephan Merz read earlier drafts of this paper and provided many helpful comments. This work was supported by the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen".

References

[AdBKR89] P. America, J. de Bakker, J. N. Kok, and J. Rutten. Denotational semantics of a parallel object-oriented language. *Information and Computation*, 83:152–205, 1989.

[AL90] M. Abadi and L. Lamport. Composing specifications. Technical Report 66, Digital, SRC, Palo Alto, 1990.

[AL93] M. Abadi and L. Lamport. Conjoining specifications. Technical Report 118, Digital, SRC, Palo Alto, 1993.

[Bro87] M. Broy. Semantics of finite and infinite networks of concurrent communicating agents. *Distributed Computing*, 2:13–31, 1987.

[Bro94] M. Broy. A functional rephrasing of the assumption/commitment specification style. Technical Report SFB 342/10/94 A, Technische Universität München, 1994.

[11] Cl represents the safety closure in [AL93]. We have renamed assumptions and commitments in accordance with the naming convention used above.

[12] Inspired by our rule, Pierre Collette suggested the variant of the rule in [AL93]. That this variant can be derived from the rule in [AL93] was pointed out by Leslie Lamport [Lam95].

[Bro95] M. Broy. Advanced component interface specification. In *Proc. TPPP'94, Lecture Notes in Computer Science 907*, pages 369–392, 1995.

[Col94] P. Collette. *Design of Compositional Proof Systems Based on Assumption-Commitment Specifications — Applications to UNITY.* PhD thesis, Université Catholique de Louvain, 1994.

[Jon83] C. B. Jones. Specification and design of (parallel) programs. In *Proc. Information Processing 83*, pages 321–331. North-Holland, 1983.

[Kok87] J. N. Kok. A fully abstract semantics for data flow nets. In *Proc. PARLE'87, Lecture Notes in Computer Science 259*, pages 351–368, 1987.

[Lam95] L. Lamport. E-mail to Pierre Collette. December 15, 22:01:13 EDT, 1995.

[MC81] J. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7:417–426, 1981.

[Pan90] P. K. Pandya. Some comments on the assumption-commitment framework for compositional verification of distributed programs. In *Proc. REX Workshop on Stepwise Refinement of Distributed Systems, Lecture Notes in Computer Science 430*, pages 622–640, 1990.

[Par83] D. Park. The "fairness" problem and nondeterministic computing networks. In *Proc. 4th Foundations of Computer Science, Mathematical Centre Tracts 159*, pages 133–161. Mathematisch Centrum Amsterdam, 1983.

[PJ91] P. K. Pandya and M. Joseph. P-A logic — a compositional proof system for distributed programs. *Distributed Computing*, 5:37–54, 1991.

[Pnu85] A. Pnueli. In transition from global to modular temporal reasoning about programs. In *Proc. Logics and Models of Concurrent Systems*, pages 123–144. Springer, 1985.

[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. To appear in Formal Aspects of Computing.

[Sta85] E. W. Stark. A proof technique for rely/guarantee properties. In *Proc. 5th Conference on the Foundation of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science 206*, pages 369–391, 1985.

[Stø91] K. Stølen. A method for the development of totally correct shared-state parallel programs. In *Proc. CONCUR'91, Lecture Notes in Computer Science 527*, pages 510–525, 1991.

[Zwi89] J. Zwiers. *Compositionality, Concurrency and Partial Correctness: Proof Theories for Networks of Processes and Their Relationship*, volume 321 of *Lecture Notes in Computer Science.* 1989.