TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

**Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen**

Assumption/Commitment Rules for Data-flow Networks — with an Emphasis on Completeness

Ketil Stølen

**TUM-I9516
SFB-Bericht Nr.342/09/95 A
Mai 1995**

TUM–INFO–05-95-I16-350/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
Arcisstr. 21 / Postfach 20 24 20
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Assumption/Commitment Rules for Data-flow Networks — with an Emphasis on Completeness

Ketil Stølen
Institut für Informatik, TU München, D-80290 München

October 30, 1995

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 data-flow 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. This has led to a rich flora of specification techniques based on the assumption/commitment format. See [MC81], [Jon83], [ZdBdR84], [BK84], [Pnu85], [Sta85], [Sti88], [AL90], [Pan90], [Stø91], [XH91], [PJ91], [AL93], [SDW93], [Col94a], [JT95] for examples.

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 in the sense that only some of the properties we would like such rules to have are captured. For example it is usual to prove some variation of relative completeness [Stø91], [Col94a] — a result which only captures some of the expectations to an assumption/commitment rule.

We study these problems in the context of a model for data-flow networks. We distinguish between two specification formats, namely simple and general specifications. The first format can only be used when the assumption is independent of the component's behavior. For both formats we propose verification rules. We prove that these rules are sound, and, moreover, that they are complete in a certain strong sense.

There are basically two styles in which assumption/commitment rules are formulated.

$$
\frac{\begin{array}{l} Premise_1 \\ \vdots \\ Premise_n \\ (A_1, C_1) \leadsto P_1 \\ (A_2, C_2) \leadsto P_2 \end{array}}{(A, C) \leadsto P_1 \parallel P_2}
\qquad\qquad
\frac{\begin{array}{l} Premise_1 \\ \vdots \\ Premise_n \end{array}}{(A, C) \leadsto (A_1, C_1) \parallel (A_2, C_2)}
$$

In the first style (see rule to the left), used already by [Hoa69], P_1, P_2 are components, and $P_1 \parallel P_2$ represents their parallel composition. Moreover, $\leadsto$ denotes the satisfaction relation. In the second style (see rule to the right), used by [AL90] and also employed in this paper, P_1, P_2 and P are eliminated by lifting the operators for parallel composition and satisfaction from components to specifications.
The rest of the paper is split into six main sections. Section 2 introduces our semantic model. Then in Section 3 simple assumption/commitment specifications are introduced, and we formulate an assumption/commitment rule with respect to a feedback operator. In Section 4 we do the same for general specifications. Then in Section 5 the assumption/commitment rules of the previous two sections are generalized to handle finite data-flow networks. Section 6 contains a brief summary and relates our approach to other approaches known from the literature. Finally, there is an appendix containing detailed proofs.

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, $D^{\overline{\omega}}$ denotes the set of all finite and infinite timed streams over D, and $D^{\overline{\infty}}$ 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, it holds that $t \sqsubseteq v$ 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 $\tau \in (D^{\overline{\infty}})^n \to (D^{\overline{\infty}})^m$ is pulse-driven iff

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

Pulse-drivenness means that the input until time j completely determines the output until time $j + 1$. The arrow $\stackrel{p}{\to}$ is used to distinguish pulse-driven functions from functions that are not pulse-driven.

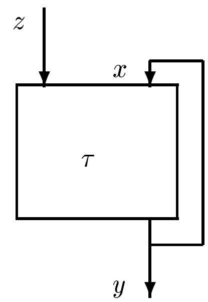

Figure 1: $\mu\,\tau$

Given a pulse-driven function $\tau \in (D^{\overline{\infty}})^n \xrightarrow{p} (D^{\overline{\infty}})^m$, where $n \geq m$, let $\mu\,\tau$ be the function we get by connecting the m output channels to the m last input channels, i.e., with respect to Figure 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 pulse-drivenness it is easy to prove that for any z there is a unique 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 pulse-driven.
For any set of functions $F \subseteq (D^{\overline{\infty}})^n \xrightarrow{p} (D^{\overline{\infty}})^m$, $\mu\,F$ denotes the set characterized by

$$\{\tau \in (D^{\overline{\infty}})^{(n-m)} \xrightarrow{p} (D^{\overline{\infty}})^m \mid \forall \theta : \exists \tau' \in F : \tau(\theta) = \mu\,\tau'(\theta)\}.$$

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

We now introduce the first of the two formats for assumption/commitment specifications, namely what we refer to as simple assumption/commitment specifications. We first define what a simple assumption/commitment specification is. We then formulate an assumption/commitment rule with respect to a simple feedback operator. We show that this rule can handle liveness properties in the assumptions. Then we discuss the completeness of this rule. We first show that relative completeness only captures some of the expectations to this rule. We then investigate more closely what these expectations are. Based on this investigation we formulate a stronger completeness property and show that our rule satisfies this property.
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

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

A and C characterize the assumption and the commitment, respectively.
The denotation of a simple assumption/commitment specification (A, C) is the set of all type-correct, pulse-driven functions that behaves in accordance with the commitment for

any input history satisfying the assumption. In other words, the set of all functions $\tau \in (D^{\overline{\infty}})^n \xrightarrow{p} (D^{\overline{\infty}})^m$ such that

$$A(i) \Rightarrow C(i, \tau(i)).$$

Throughout this paper, for any assumption/commitment specification S, we use $[\![\ S\ ]\!]$ to denote its denotation, and A_S and C_S to denote its assumption and commitment, respectively. The feedback operator μ is lifted from pulse-driven functions to specifications in the obvious way: $[\![\ \mu\,(A,C)\ ]\!] \stackrel{\mathsf{def}}{=} \mu\,[\![\ (A,C)\ ]\!]$. A specification S_2 is said to refine a specification S_1 iff $[\![\ S_2\ ]\!] \subseteq [\![\ S_1\ ]\!]$. We then write $S_1 \rightsquigarrow S_2$. 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 μ operator. To simplify the rule, for any predicate $P \in (D^{\overline{\infty}})^n \rightarrow \mathsf{B}$, let $\langle P \rangle$ denote the element of $(D^{\overline{\omega}})^n \rightarrow \mathsf{B}$ such that

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

The following rule is obviously sound[1]

Rule 1 :

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

However, this rule is not very helpful from a practical point of view. It only translates the conclusion into the underlying logic without giving much hint about how a proof should be constructed.
By introducing an invariant $I \in (D^{\overline{\infty}})^q \times (D^{\overline{\omega}})^m \rightarrow \mathsf{B}$ a more useful rule can be formulated

Rule 2 :

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

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_∞. In the sequel we often refer to A_1 as the overall assumption and to A_2 as the component assumption (and accordingly for the commitments).

Lemma 1 *Rule 2 is sound.*

Proof: 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 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.
A detailed proof can be found in Section A.3 of the appendix. □

Note 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, the rule

[1] With respect to Figure 1, z represents the q external input channels, and y represents the m output channels which are also fed back to x.

does not depend on that the assumptions are split into safety and liveness parts. Thus, Rule 2 allows assumption/commitment specifications to be reasoned about in a natural way.
The main reason why Rule 2 can deal with arbitrary liveness properties in the assumptions is that it makes a clear distinction between induction hypotheses and component assumption. Without this distinction — in other words, if we had used the component assumption as induction hypotheses, which is common in the case of assumption/commitment rules, the component assumption would be required to satisfy the same type of admissibility property which is imposed on the invariant by the fourth premise. As a consequence, we would only be able to handle restricted types of liveness properties in the component assumption, namely those having this admissibility property.
To show how Rule 2 can be used to handle liveness properties in the assumptions, we present a small example. For this purpose, we first have to introduce some operators on streams.
An untimed stream is a finite or infinite sequence of messages. It differs from a timed one in that it has no occurrences of ticks. Given an untimed stream r and a positive natural number n; $\#r$ denotes the length of r (∞ if r is infinite) and $r(n)$ denotes the n'th element of r if $n \leq \#r$. These operators are overloaded to timed streams in a straightforward way. Given that for any timed stream r, $\overline{r}$ denotes the result of removing all ticks in r, then $\#r \stackrel{\mathsf{def}}{=} \#\overline{r}$ and $r(n) \stackrel{\mathsf{def}}{=} \overline{r}(n)$.

Example 1 Liveness in the assumptions:
Consider the two specifications S_1 and S_2, where

$$\begin{array}{lcl}
A_{S_1}(z) & \stackrel{\mathsf{def}}{=} & \#z = \infty, \\
C_{S_1}(z, y) & \stackrel{\mathsf{def}}{=} & \#y = \infty \wedge \forall j \in \mathsf{N}_+ : y(j) = \sum_{k=1}^{j-1} z(k), \\
A_{S_2}(z \cdot x) & \stackrel{\mathsf{def}}{=} & \#x = \#z = \infty, \\
C_{S_2}(z \cdot x, y) & \stackrel{\mathsf{def}}{=} & y(1) = 0 \wedge y(j+1) = z(j) + x(j) \wedge \\
 & & \forall j \in \mathsf{N}_+ : \#y\downarrow_j = \mathsf{min}\{\#x\downarrow_{(j-1)}, \#z\downarrow_{(j-1)}\} + 1.
\end{array}$$

We assume all channels are of type natural number. Since A_{S_1} and A_{S_2} can be falsified only by infinite observations, they characterize pure liveness properties. S_1 first outputs a 0 and thereafter, each time S_1 receives a natural number n along its only input channel z, the sum of n and the sum of all the numbers previously received.
S_2, on the other hand, first outputs a 0, and thereafter the sum of each pair (n, m), where n is the j'th number received on z and m is the j'th number received on x. This explains the two first conjuncts of C_{S_2}. The delay along y is required to be exactly one time unit with respect to the most recently received number. This timing constraint is expressed by the third conjunct. We may use Rule 2 to prove that

$$S_1 \rightsquigarrow \mu\, S_2$$

by defining

$$\begin{array}{lcl}
I(z, y) & \stackrel{\mathsf{def}}{=} & \#z = \infty \wedge \\
 & & \forall j \in \mathsf{N}_+ : \#z\downarrow_{(j-1)} < \#z\downarrow_j \wedge y\downarrow_j \neq y\downarrow_{(j+1)} \Rightarrow \#y\downarrow_j < \#y\downarrow_{(j+1)}
\end{array}$$

In [AL90] it is shown that any assumption/commitment specification satisfying a certain realizability constraint can be translated into an equivalent specification, whose assumption is a safety property, by placing the liveness assumptions in the commitment. A similar result holds for our specifications. With respect S_1 and S_2 both assumptions would then become equivalent to true; moreover we could use true as invariant, in which case the first four premises would follow trivially. However, the verification of the fifth premise would then become more complicated. □

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

Lemma 2 *Given a non-empty set* $F \subseteq (D^{\overline{\infty}})^{(q+m)} \stackrel{p}{\to} (D^{\overline{\infty}})^m$ *and assume that* $\mu F \subseteq [\![S_1]\!]$. *Then there is a specification* S_2 *and a predicate*

$$I \in (D^{\overline{\infty}})^q \times (D^{\overline{\omega}})^m \to \mathsf{B}$$

such that the five premises of Rule 2 are valid and $F \subseteq [\![S_2]\!]$.

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

The completeness result characterized by Lemma 2 just says that whenever we have a data-flow 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 both as component assumption and invariant. Since the validness of the first four premises follows trivially this result does not test the special features of Rule 2. Thus, it is clear that Lemma 2 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[2]. This specification is obviously inconsistent in the sense that its denotation is empty, and it is clearly not implementable (modulo our concept of refinement $\rightsquigarrow$ and components modeled by non-empty sets of pulse-driven functions). In fact, any specification, which disallows any component behavior for at least one input history satisfying the assumption, is trivially not implementable.
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 pulse-drivenness.

Example 2 Disallowing pulse-drivenness:
Consider the specification S, where

$$A_S(i) \stackrel{\mathsf{def}}{=} \mathsf{true},$$

$$C_S(i, o) \stackrel{\mathsf{def}}{=} (i = \langle\surd\rangle^\infty \wedge o = \langle\surd\rangle^\infty) \vee (i \neq \langle\surd\rangle^\infty \wedge o = 1 \frown \langle\surd\rangle^\infty).$$

The operator $\frown$ is used to extend a stream with a new first element (later it will also be used to concatenate streams), and $\langle\surd\rangle^\infty$ denotes an infinite timed stream consisting of only ticks. Assume $\tau \in [\![S]\!]$. For any input history $i \neq \langle\surd\rangle^\infty$ it holds that

$$i{\downarrow_0} = \langle\surd\rangle^\infty{\downarrow_0}.$$

The pulse-drivenness of τ implies

$$\tau(i){\downarrow_1} = \tau(\langle\surd\rangle^\infty){\downarrow_1}.$$

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

But then, $\tau \in [\![S]\!]$ implies $1 = \surd$. Thus, S is inconsistent. Nevertheless, S allows an output behavior for any input behavior satisfying the assumption. Thus, S is inconsistent because it disallows pulse-drivenness. $\Box$

We say that a simple assumption/commitment specification S is consistent if $[\![S]\!] \neq \emptyset$.
A simple assumption/commitment specification as defined above may have a commitment that is not fully realizable with respect to input histories satisfying the assumption or partial input histories that have not yet falsified the assumption.

Example 3 Not fully realizable:
Consider the specification S, where

$$A_S(i) \stackrel{\mathsf{def}}{=} \mathsf{true}, \quad C_S(i,o) \stackrel{\mathsf{def}}{=} o = \langle \surd \rangle^\infty \vee i = o = \langle 1, \surd \rangle^\infty .$$

It is here assumed that $\langle 1, \surd \rangle^\infty$ denotes the timed stream we get by concatenating infinitely many copies of the finite stream consisting of a 1 followed by a $\surd$. 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 pulse-drivenness 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 pulse-driven function (and therefore also not realizable by any implementation modulo $\rightsquigarrow$). $\Box$

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 (possibly infinite) 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.

Example 4 Fully realizable specification:
For example, given that for any message m and timed stream s, $m \copyright s$ returns the result of removing any element in s different from m, then the specification S where

$$A_S(i) \stackrel{\mathsf{def}}{=} \#1 \copyright i = \infty, \qquad C_S(i,o) \stackrel{\mathsf{def}}{=} \#2 \copyright o = \infty$$

is both consistent and fully realizable. Both the assumption and the commitment are liveness properties since they can only be falsified by infinite observations. $\Box$

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 Example 3, 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 than their fully realizable counterparts.

Example 5 Implicit constraints:
Consider for example the specification S where

$$A_S(i) \stackrel{\mathsf{def}}{=} \mathsf{true}, \qquad C_S(i,o) \stackrel{\mathsf{def}}{=} \bar{\imath} = \bar{o}.$$

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

$$i = a \frown \langle \surd \rangle^\infty, \qquad o = a \frown \langle \surd \rangle^\infty.$$

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 pulse-driven. This contradicts that $\tau \in [\![S]\!]$. The specification S', where

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

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

Of course, in this small example it does not really matter. Nevertheless, in nontrivial cases specifications can be considerably shortened by leaving out constraints already imposed via the semantics. On the other hand, specifications with such implicit constraints will more often be misunderstood and lead to mistakes because the implicit constraints are overseen. The debate on implicit constraints is to some degree related to the debate on whether specifications split into safety and liveness conditions should be machine-closed or not [AS85], [DW90], [AAA$^+$91], [DW91]. We do not take any standpoint to this here.
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

$$\frac{A(i) \wedge (\forall j \in \mathsf{N}_\infty : \forall s : A(i{\downarrow_j} \frown s) \Rightarrow \exists r : C(i{\downarrow_j} \frown s, o{\downarrow_{(j+1)}} \frown r)) \Rightarrow C'(i,o)}{(A, C') \rightsquigarrow (A, C)}$$

Example 6 Adaptation:
For example, this rule can be used to prove that the specification S of Example 3 is a refinement of the fully realizable equivalent specification S' where

$$A_{S'}(i) \stackrel{\mathsf{def}}{=} \mathsf{true}, \qquad C_{S'}(i,o) \stackrel{\mathsf{def}}{=} o = \langle \surd \rangle^\infty.$$

Assume $i = o = \langle 1, \surd \rangle^\infty$. S' can be deduced from S by the adaptation rule if we can find an s and a $j \in \mathsf{N}_\infty$ such that for all r

$$o{\downarrow_{(j+1)}} \frown r \neq \langle \surd \rangle^\infty \wedge \neg(i{\downarrow_j} \frown s = o{\downarrow_{(j+1)}} \frown r = \langle 1, \surd \rangle^\infty)$$

For example, this is the case if $s = \langle \surd \rangle^\infty$ and $j = 0$. □

Example 7 Adaptation:
With respect to Example 5, the adaptation rule can also be used to prove that the specification S is a refinement of the equivalent specification S'. To see that, let i, o and j be such that $\overline{o\downarrow_{(j+1)}} \not\sqsubseteq \overline{i\downarrow_j}$. S' can be deduced from S by the adaptation rule if we can find an s such that for all r

$$\overline{(i\downarrow_j \frown s)} \neq \overline{(o\downarrow_{(j+1)} \frown r)}.$$

Clearly, this is the case if $s = \langle\surd\rangle^\infty$. $\Box$

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 under the assumption that we have a complete set of deduction rules for our assertion language? Unfortunately, the answer is "no".

Example 8 Incompleteness:
To see that, first note that the specification S where

$$A_S(i) \stackrel{\mathsf{def}}{=} \mathsf{true}, \qquad C_S(i,o) \stackrel{\mathsf{def}}{=} o \neq i,$$

is inconsistent.
To prove this, assume $\tau \in [\![\ S \ ]\!]$. τ is pulse-driven 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

$$\forall j \in \mathsf{N}, s : \exists r : o\downarrow_{(j+1)} \frown r \neq i\downarrow_j \frown s,$$

it follows that the adaptation rule cannot be used to adapt S. A slightly weaker, consistent version of S is S' where

$$A_{S'}(i) \stackrel{\mathsf{def}}{=} \mathsf{true}, \qquad C_{S'}(i,o) \stackrel{\mathsf{def}}{=} o \neq i \vee o = \langle\surd\rangle^\infty.$$

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' follows by the same argument as for S. Moreover, since any $\tau \in [\![\ S' \ ]\!]$ has $\langle\surd\rangle^\infty$ as its fix-point, it follows from the pulse-drivenness 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. $\Box$

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

$$\frac{A(i) \wedge \tau \in [\![\ (A, C) \ ]\!] \Rightarrow C'(i, \tau(o))}{(A, C') \rightsquigarrow (A, C)}$$

Of course this type of adaptation can also be built into Rule 2. It is enough to replace the antecedent of the third premise by

$$I(z, y\downarrow_j) \wedge \exists \tau \in [\![\ (A_2, C_2) \ ]\!] : \tau(z \cdot y) = y.$$

However, in our opinion, assumption/commitment rules should not be expected to be adaptation complete. Firstly, as shown above, by building adaptation into assumption/commitment rules, the rules become 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 pulse-driven 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 this 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 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, as shown in the proof of Lemma 3 below, Rule 2 allows us to build the overall assumption into the invariant. Thus, this formula is basically "equivalent" to

$$A_1(z) \wedge A_2(z \cdot y) \wedge C_2(z \cdot y, y) \Rightarrow C_1(z,y).$$

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 depend upon the specified component's behavior). Moreover, the rule for parallel composition makes sure that if the environment behaves in accordance with the overall pre- and rely-conditions, then the two components behave in accordance with their respective pre- and rely-conditions.

Thus, since for example $S_1 \rightsquigarrow \mu\, S_2$, given that

$$A_{S_1}(i) \stackrel{\mathsf{def}}{=} C_{S_1}(i,o) \stackrel{\mathsf{def}}{=} C_{S_2}(i,o) \stackrel{\mathsf{def}}{=} \mathsf{true}, \quad A_{S_2}(i) \stackrel{\mathsf{def}}{=} \mathsf{false},$$

although

$$[\![\ S_2\ ]\!] = (D^{\overline{\infty}})^{(q+m)} \xrightarrow{p} (D^{\overline{\infty}})^m,$$

the completeness property proposed above is too strong. It has to be weakened into

- whenever the conclusion holds, and $z \cdot \tau(z)$ satisfies the component assumption A_2 for any input history z satisfying the overall assumption A_1 and function $\tau \in [\![S_1]\!]$, then we can find an invariant I such that the five premises of Rule 2 are valid.

More formally

Lemma 3 *Given two simple specifications S_1 and S_2 such that $S_1 \rightsquigarrow \mu\, S_2$. Assume that S_2 is consistent and fully realizable, and moreover that*

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

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

Proof: Let

$$I(z, y) \stackrel{\mathsf{def}}{=} A_{S_1}(z) \wedge \forall k \in \mathsf{N} : \langle A_{S_2} \rangle (z \cdot y{\downarrow_k}) \wedge \langle C_{S_2} \rangle (z \cdot y{\downarrow_k}, y{\downarrow_k}).$$

See Section A.4 of the appendix for more details. □

The proof of Lemma 3 is based on the fact that there is a canonical invariant — more precisely, a schema that gives an invariant that is sufficiently strong. As a consequence, if we fix the invariant in accordance with the proof of Lemma 3, we may simplify Rule 2 by removing the fourth premise and replacing the second by $I(z, y) \Rightarrow A_2(z \cdot y)$. However, from a practical point of view, 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

We now introduce the second specification format, namely so-called general assumption/commitment specifications. We first discuss the semantics of this format. Then we reformulate the assumption/commitment rule for simple specifications. We prove that this new rule is sound and satisfies a completeness result similar to that for the previous rule.
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

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

The denotation of a general assumption/commitment specification (A, C) is the set of all functions $\tau \in (D^{\overline{\infty}})^n \xrightarrow{p} (D^{\overline{\infty}})^m$ such that

$$\langle A \rangle (i, \tau(i){\downarrow_j}) \Rightarrow \langle C \rangle (i, \tau(i){\downarrow_{(j+1)}}).$$

Note that, since

$$\langle A \rangle (i, \tau(i){\downarrow_\infty}) \Rightarrow \langle C \rangle (i, \tau(i){\downarrow_{(\infty+1)}})$$

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, we now also require that if the environment behaves in accordance with the assumption until time j, then any correct implementation must behave in accordance with the commitment until time $j + 1$.

Thus, this semantics guarantees that any correct implementation fulfills the commitment at least one step longer than the environment fulfills the assumption. As will be shown, this one-step-longer-than semantics allows Rule 2 to be restated for general specifications in a straightforward way.
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.
An alternative semantics would be the set of all functions $\tau \in (D^{\overline{\infty}})^n \xrightarrow{p} (D^{\overline{\infty}})^m$ such that

$$\langle A \rangle (i{\downarrow_j}, \tau(i){\downarrow_j}) \Rightarrow \langle C \rangle (i{\downarrow_j}, \tau(i){\downarrow_{(j+1)}}).$$

We use $[\![\ (A, C) \ ]\!]_{alt}$ to denote this set. We prefer the $[\![\quad]\!]$ semantics because $[\![\quad]\!]$ is more natural as long as the two predicates A and C characterize relations on infinite communication histories. Moreover, as will be shown below, we can always restate an assumption/commitment specification in such a way that $[\![\quad]\!]$ and $[\![\quad]\!]_{alt}$ yield the same set of functions.

Lemma 4 *For any general assumption/commitment specification S, we have that*

$$[\![\ S \ ]\!] \subseteq [\![\ S \ ]\!]_{alt}.$$

Proof: Let $\tau \in [\![\ S \ ]\!]$ and assume $\langle A_S \rangle (i{\downarrow_j}, \tau(i){\downarrow_j})$. It follows straightforwardly that there is an s such that $\langle A_S \rangle (i{\downarrow_j} \frown s, \tau(i){\downarrow_j})$, in which case we also have that $\langle C_S \rangle (i{\downarrow_j} \frown s, \tau(i){\downarrow_{(j+1)}})$. But this implies $\langle C_S \rangle (i{\downarrow_j}, \tau(i){\downarrow_{(j+1)}})$. Thus $\tau \in [\![\ S \ ]\!]_{alt}$. $\Box$

On the other hand, in the general case, it does not hold that

$$[\![\ S \ ]\!]_{alt} \subseteq [\![\ S \ ]\!].$$

Example 9 $[\![\quad]\!]$ versus $[\![\quad]\!]_{alt}$:
To see that, let S be the specification such that

$$A_S(i, o) \stackrel{\mathsf{def}}{=} i = \surd \frown o,$$

$$C_S(i, o) \stackrel{\mathsf{def}}{=} (i = \langle \surd \rangle^\infty \wedge o = \langle \surd \rangle^\infty) \vee (\exists n \in \mathsf{N} : o = \langle n, \surd \rangle^\infty).$$

Let $\tau = \lambda i. \langle \surd \rangle^\infty$. Clearly

$$\langle A_S \rangle (i{\downarrow_j}, \tau(i){\downarrow_j}) \Leftrightarrow i{\downarrow_j} = \langle \surd \rangle^j.$$

Since $\langle C_S \rangle (\langle \surd \rangle^\infty{\downarrow_j}, \langle \surd \rangle^\infty{\downarrow_{(j+1)}})$ we have that $\tau \in [\![\ S \ ]\!]_{alt}$. On the other hand, let $i = \langle \surd \rangle^{j+1} \frown \langle 1, \surd \rangle^\infty$. It clearly holds that $\langle A_S \rangle (i, \tau(i){\downarrow_j})$, but $\langle C_S \rangle (i, \tau(i){\downarrow_{(j+1)}})$ does not hold. Thus $\tau \notin [\![\ S \ ]\!]$. $\Box$

Nevertheless, it can be shown that

$$[\![\ S \ ]\!]_{alt} = [\![\ (A_S, \exists \tau \in [\![\ S \ ]\!]_{alt} : \tau(i) = o) \ ]\!]_{alt},$$

$$[\![\ (A_S, \exists \tau \in [\![\ S \ ]\!]_{alt} : \tau(i) = o) \ ]\!]_{alt} = [\![\ (A_S, \exists \tau \in [\![\ S \ ]\!]_{alt} : \tau(i) = o) \ ]\!].$$

The correctness of the first equality follows trivially. The correctness of the second follows since each function is defined for any input history. Thus, we can always find an equivalent specification such that the difference between $[\![\quad]\!]$ and $[\![\quad]\!]_{alt}$ does not matter.

Under the assumption that z and y vary over q- respectively m-tuples of infinite timed streams, and j varies over N_∞, the assumption/commitment rule for the μ construct can be restated as below

$$\begin{array}{l}
\text{Rule 3 :} \\
A_1(z,y) \Rightarrow I(z,y{\downarrow}_0) \\
I(z,y{\downarrow}_j) \Rightarrow \langle A_2\rangle(z \cdot y{\downarrow}_j, y{\downarrow}_j) \\
A_1(z,y) \wedge I(z,y{\downarrow}_j) \wedge \langle C_2\rangle(z \cdot y{\downarrow}_j, y{\downarrow}_{(j+1)}) \Rightarrow I(z,y{\downarrow}_{(j+1)}) \\
A_1(z,y) \wedge \forall k \in \mathsf{N} : I(z,y{\downarrow}_k) \Rightarrow I(z,y) \\
I(z,y{\downarrow}_j) \wedge \langle C_2\rangle(z \cdot y{\downarrow}_j, y{\downarrow}_{(j+1)}) \Rightarrow \langle C_1\rangle(z,y{\downarrow}_{(j+1)}) \\
\hline
(A_1, C_1) \rightsquigarrow \mu\,(A_2, C_2)
\end{array}$$

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 assumption hold at least as long as the overall assumption is not falsified. This explains the modifications to the third and fourth premise. 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.

Lemma 5 *Rule 3 is sound.*

Proof: An informal justification has been given above. See Section A.1 of the appendix for a detailed proof. □

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

$$\begin{array}{l}
\forall \tau \in [\![\ S\ ]\!] : \exists \tau' \in [\![\ S\ ]\!] : \\
\qquad \langle A_S\rangle(i{\downarrow}_j, o{\downarrow}_j) \wedge o{\downarrow}_j = \tau(i){\downarrow}_j \wedge \langle C_S\rangle(i{\downarrow}_j, o{\downarrow}_{(j+1)}) \Rightarrow \tau'(i){\downarrow}_{(j+1)} = o{\downarrow}_{(j+1)}.
\end{array}$$

Thus, a general specification is fully realizable iff for any complete input history i and complete output history o such that the assumption holds until time j and the commitment holds until time $j+1$: if there is a pulse-driven function τ in the denotation of S that behaves in accordance with (i,o) until time j, then there is a pulse-driven function τ' in the denotation of S that behaves in accordance with (i,o) until time $j+1$. 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 Lemma 3 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. Lemma 3 can then be restated as below

Lemma 6 *Given two general specifications S_1 and S_2 such that $S_1 \rightsquigarrow \mu\, S_2$. Assume that S_2 is consistent and fully realizable, and moreover that*

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

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

Proof: Let

$$I(z, y) \stackrel{\mathsf{def}}{=} \langle A_{S_1} \rangle(z, y) \wedge \forall k \in \mathsf{N} : \langle A_{S_2} \rangle(z \cdot y{\downarrow_k}, y{\downarrow_k}) \wedge \langle C_{S_2} \rangle(z \cdot y{\downarrow_k}, y{\downarrow_k}).$$

See Section A.2 of the appendix for details. □

5 Network Rule

We now outline how the rules introduced above can be generalized to deal with finite data-flow networks. For this purpose, we represent specifications in a slightly different way.
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 the following format

$$(i, o, A, C),$$

where i is a finite totally ordered set of input names, o is a finite totally ordered set of output names, 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. As shown below, the advantage of this format 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.
Given n general specifications

$$(z_1 \cup x_1, y_1, A_1, C_1),\ (z_2 \cup x_2, y_2, A_2, C_2), \quad \ldots \quad, (z_n \cup x_n, y_n, A_n, C_n).$$

For each k, the sets z_k, x_k and y_k name respectively the external input channels (those connected to the overall environment), the internal input channels (those connected to the other $n-1$ specifications in the network), and the external and internal output channels.
Let

$$z = \cup_{k=1}^{n} z_k, \quad x = \cup_{k=1}^{n} x_k, \quad y = \cup_{k=1}^{n} y_k.$$

It is assumed that $z \cap x = z \cap y = \emptyset$ and that $x \subseteq y$. Moreover, it is assumed that

$$l \neq k \Rightarrow y_l \cap y_k = \emptyset.$$

We can then think of these n specifications as modeling a network of n components where the input and output channels are connected iff they have identical names. The constraints imposed on the sets of channel names imply that two different specifications cannot write on the same channel. They may have read access to 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

$$\|_{k=1}^{n}\ (z_k \cup x_k, y_k, A_k, C_k).$$

Thus, given that z and y contain n respectively m elements, the denotation of this network is the set of all functions

$$\tau \in (D^{\overline{\infty}})^n \xrightarrow{p} (D^{\overline{\infty}})^m,$$

where for each z, there are functions $\tau_j \in [\![\ (z_j \cup x_j, y_j, A_j, C_j) \ ]\!]$ such that[3]

$$\tau(z) = y$$

if

$$y_1 = \tau_1(z_1 \cdot x_1), \ y_2 = \tau_2(z_2 \cdot x_2), \ \ldots \ , \ y_n = \tau_n(z_n \cdot x_n).$$

Due to the pulse-drivenness of each τ_j, it follows that for each z there is a unique y such that (z, y) is a solution of these n equations. Thus, τ is well-defined, and it is also easy to prove that τ is pulse-driven.
By defining $P[^a_b]$ to denote the result of replacing each occurrence of a in P by b, a straightforward generalization of Rule 3 gives:

$$\begin{array}{l}
\text{Rule 4 :} \\
\ A \Rightarrow I[^{y}_{y\downarrow_0}] \\
\ I[^{y}_{y\downarrow_j}] \Rightarrow (\wedge_{k=1}^{n} \langle A_k \rangle [^{y}_{y\downarrow_j}]) \\
\ A \wedge I[^{y}_{y\downarrow_j}] \wedge (\wedge_{k=1}^{n} \langle C_k \rangle [^{x_k}_{x_k\downarrow_j} \ ^{y_k}_{y_k\downarrow_{(j+1)}}]) \Rightarrow I[^{y}_{y\downarrow_{(j+1)}}] \\
\ A \wedge \forall k \in \mathsf{N} : I[^{y}_{y\downarrow_k}] \Rightarrow I \\
\ I[^{y}_{y\downarrow_j}] \wedge (\wedge_{k=1}^{n} \langle C_k \rangle [^{x_k}_{x_k\downarrow_j} \ ^{y_k}_{y_k\downarrow_{(j+1)}}]) \Rightarrow \langle C \rangle [^{y}_{y\downarrow_{(j+1)}}] \\
\hline
\ (z, y, A, C) \rightsquigarrow \|_{k=1}^{n} (z_k \cup x_k, y_k, A_k, C_k)
\end{array}$$

The elements of z and y vary over $D^{\overline{\infty}}$, and j varies over N_∞.
However this rule ignores one aspect, namely that we are now dealing with n specifications and not only 1. For example, if $n = 2$, it may be the case that the invariant I only implies 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:

$$\begin{array}{l}
\text{Rule 5 :} \\
\ A \Rightarrow I[^{y}_{y\downarrow_0}] \\
\ I[^{y}_{y\downarrow_j}] \Rightarrow (\wedge_{k=1}^{n} \langle A_k \rangle [^{y}_{y\downarrow_j}]) \\
\ A \wedge I[^{y}_{y\downarrow_j}] \wedge (\wedge_{k=1}^{n} \langle C_k \rangle [^{x_k}_{x_k\downarrow_j} \ ^{y_k}_{y_k\downarrow_{(j+1)}}]) \Rightarrow I[^{y}_{y\downarrow_{(j+1)}}] \\
\ A \wedge \forall k \in \mathsf{N} : I[^{y}_{y\downarrow_k}] \Rightarrow I \\
\ I[^{y}_{y\downarrow_j}] \wedge (\wedge_{k=1}^{n} \langle C_k \rangle [^{x_k}_{x_k\downarrow_j} \ ^{y_k}_{y_k\downarrow_{(j+1)}}]) \Rightarrow \langle C \rangle [^{y}_{y\downarrow_{(j+1)}}] \\
\ I \wedge (\wedge_{k=1}^{n} A_k \Rightarrow C_k) \Rightarrow C \\
\hline
\ (z, y, A, C) \rightsquigarrow \|_{k=1}^{n} (z_k \cup x_k, y_k, A_k, C_k)
\end{array}$$

As for Rule 4, the elements of z and y vary over $D^{\overline{\infty}}$. However, j now only varies over N.
For Rule 5 we may prove soundness and completeness results similar to those for Rule 3.

6 Conclusions

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

- We have introduced two specification formats, namely simple and general assumption/commitment specifications.

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

- For these specification formats we have formulated assumption/commitment rules and proved their soundness.
- We have shown that our rules handle assumptions with arbitrary liveness properties. We have argued that this is due to the fact that the rules make a clear distinction between induction hypotheses and environment assumptions.
- We have argued that the usual concept of relative completeness only captures 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 a stronger completeness requirement and proved that our rules satisfy this requirement.
- For general specifications we have proposed a semantics that guarantees that a correct implementation will behave in accordance with the commitment at least one step longer than the environment behaves in accordance with the assumption.
- Finally, we have outlined how the specification formats and proposed rules can be generalized to specify and prove properties of general data-flow networks.

We have had many sources of inspiration. We now relate our approach to the most important.
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. However, contrary to [Bro87], we use pulse-driven functions and infinite timed streams. Thereby we get a simpler theory. The actual formulation of pulse-drivenness has been taken from [Bro95]. We refer to [GS95] for more details on the semantic model.
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]. [Col94b] employs a slightly weaker semantics — the commitment is only required to hold at least as long the assumption has not been falsified.
Assumption/Commitment Rules: A large number of composition rules for assumption/commitment specifications have been published. In the case of sequential systems they were introduced with Hoare-logic [Hoa69]. In the concurrent case such rules were first proposed by [Jon81], [MC81].
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. The rule proposed in [AL93] handles some liveness properties in the assumptions. We have not yet understood the exact relationship to our rules.
[AL93] argues that from a pragmatic point of view specifications should always be formulated in such a way that the assumption is a safety property. 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. [SF95] contains a case-study using the proposed formalism.
Completeness: The concept of relative completeness was first introduced by [Coo78]. See for example [Apt81], [HdRLX92] for an overview of the literature on relative 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 lies in between compositional and modular completeness. It can almost be understood as modular completeness under the assumption that adaptation complete adaptation rules are available.
Expressiveness: The results presented in this paper are all of a rather semantic nature in the sense that we do not explicitly define a logical assertion language. Let L be the assertion language in which the assumptions, commitments and invariants are expressed. The proof of Lemma 6 depends on the formulation of a canonical invariant. This means that if L allows the operator $\downarrow$, the closure $\langle P \rangle$ for any formula P in L, the set $(\overline{D^{\infty}})^q$ for any set D and natural number q, and the usual logical operators to be expressed, then our completeness result carry over. Examples of languages that have this expressiveness are PVS [OSR93] and HOLCF [Reg94].

7 Acknowledgment

This work is supported by the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen". Manfred Broy, Pierre Collette and Stephan Merz have read an early draft of this paper and provided helpful comments.

References

[AAA$^+$91] M. Abadi, B. Alpern, K. R. Apt, N. Francez, S. Katz, L. Lamport, and F. Schneider. Preserving liveness: Comments on "Safety and liveness from a methodological point of view". *Information Processing Letters*, 40:141–142, 1991.

[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.

[Apt81] K. R. Apt. Ten years of Hoare's logic: A survey — part I. *ACM Transactions on Programming Languages and Systems*, 3:431–483, 1981.

[AS85] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.

[BK84] H. Barringer and R. Kuiper. Towards the hierarchical temporal logic specification of concurrent systems. In *Proc. The Analysis of Concurrent Systems, Lecture Notes in Computer Science 207*, pages 157–184, 1984.

[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.

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

[Col94a] P. Collette. *Design of Compositional Proof Systems Based on Assumption-Commitment Specifications — Applications to UNITY*. PhD thesis, Université Catholique de Louvain, 1994.

[Col94b] P. Collette. An explanatory presentation of composition rules for assumption-commitment specifications. *Information Processing Letters*, 50:31–35, 1994.

[Coo78] S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7:70–90, 1978.

[DW90] F. Dederichs and R. Weber. Safety and liveness from a methodological point of view. *Information Processing Letters*, 36:25–30, 1990.

[DW91] F. Dederichs and R. Weber. Reply to the comments by M. Abadi et al. *Information Processing Letters*, 40:143–143, 1991.

[GS95] R. Grosu and K. Stølen. A denotational model for mobile point-to-point dataflow networks. Technical Report SFB 342/14/95 A, Technische Universität München, 1995. http://www4.informatik.tu-muenchen.de/reports/TUM-I9527.

[HdRLX92] J. Hooman, W. P. de Roever, Y. Lakhneche, and Q. Xu. Verification and specification of concurrent programs: From noncompositional to compositional proof methods. Manuscript of book, February 1992.

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

[Jon81] C. B. Jones. *Development Methods for Computer Programs Including a Notion of Interference*. PhD thesis, Oxford University, 1981.

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

[JT95] B. Jonsson and Y.-K. Tsay. Assumption/guarantee specifications in linear-time temporal logic. In *Proc. TAPSOFT'95, Lecture Notes in Computer Science 915*, pages 262–276, 1995.

[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.

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

[OSR93] S. Owre, N. Shankar, and J. M. Rushby. *User Guide for the PVS Specification and Verification System (Beta Release)*. Computer Science Laboratory, SRI International, 1993.

[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.

[Reg94] F. Regensburger. *HOLCF: eine konservative Erweiterung von HOL um LCF*. PhD thesis, Technische Universität München, 1994.

[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.

[SF95] K. Stølen and M. Fuchs. A formal method for hardware/software co-design. Technical Report SFB 342/10/95 A, Technische Universität München, 1995.

[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.

[Sti88] C. Stirling. A generalization of Owicki-Gries's Hoare logic for a concurrent while language. *Theoretical Computer Science*, 58:347–359, 1988.

[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.

[XH91] Q. Xu and J. He. A theory of state-based parallel programming by refinement:part 1. In *Proc. 4th BCS-FACS Refinement Workshop*. Springer, 1991.

[ZdBdR84] J. Zwiers, A. de Bruin, and W. P. de Roever. A proof system for partial correctness of dynamic networks of processes. In *Proc. Logics of Programs, Lecture Notes in Computer Science 164*, pages 513–527, 1984.

[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.

A Detailed Proofs

We first prove Lemmas 5 and 6. These two lemmas are then used to prove Lemmas 1 and 3. In these proofs, unless anything else is stated explicitly, any free occurrence of z, y and j is universally quantified over $(D^{\overline{\infty}})^q$, $(D^{\overline{\infty}})^m$ and N_∞, respectively.

A.1 Proof of Lemma 4

Assume

$$
\begin{aligned}
&(1) : A_1(z, y) \Rightarrow I(z, y{\downarrow_0}),\\
&(2) : I(z, y{\downarrow_j}) \Rightarrow \langle A_2\rangle(z \cdot y{\downarrow_j}, y{\downarrow_j}),\\
&(3) : A_1(z, y) \wedge I(z, y{\downarrow_j}) \wedge \langle C_2\rangle(z \cdot y{\downarrow_j}, y{\downarrow_{(j+1)}}) \Rightarrow I(z, y{\downarrow_{(j+1)}}),\\
&(4) : A_1(z, y) \wedge \forall k \in \mathsf{N} : I(z, y{\downarrow_k}) \Rightarrow I(z, y),\\
&(5) : I(z, y{\downarrow_j}) \wedge \langle C_2\rangle(z \cdot y{\downarrow_j}, y{\downarrow_{(j+1)}}) \Rightarrow \langle C_1\rangle(z, y{\downarrow_{(j+1)}}).
\end{aligned}
$$

It must be shown that

$(6) : (A_1, C_1) \rightsquigarrow \mu\,(A_2, C_2).$

Given

$(7) : \tau \in [\![\ \mu\,(A_2, C_2)\ ]\!].$

(6) follows if it can be shown that

$(8) : \langle A_1 \rangle (z, \tau(z){\downarrow_j}) \Rightarrow \langle C_1 \rangle (z, \tau(z){\downarrow_{(j+1)}}).$

Given some $z \in (D^{\overline{\infty}})^q$. (7) and the definition of μ imply there is a τ' such that

$(9) : \tau' \in [\![\ (A_2, C_2)\ ]\!],$
$(10) : \tau(z) = \mu\,\tau'(z).$

Let

$(11) : y = \mu\,\tau'(z).$

(8) follows if it can be shown that

$(12) : A_1(z, y) \Rightarrow C_1(z, y),$
$(13) : \forall j \in \mathsf{N} : \langle A_1 \rangle (z, y{\downarrow_j}) \Rightarrow \langle C_1 \rangle (z, y{\downarrow_{(j+1)}}).$

To prove (12), assume

$(14) : A_1(z, y).$

(12) follows if it can be shown that

$(15) : C_1(z, y).$

We prove by induction on k that

$(16) : \forall k \in \mathsf{N} : I(z, y{\downarrow_k}).$

$(1), (14)$ imply the base-case. Assume

$(17) : I(z, y{\downarrow_l}).$

It must be shown that

$(18) : I(z, y{\downarrow_{(l+1)}}).$

$(2), (17)$ imply

$(19) : \langle A_2 \rangle (z \cdot y{\downarrow_l}, y{\downarrow_l}).$

$(9), (11), (19)$ imply

$(20) : \langle C_2 \rangle (z \cdot y{\downarrow_l}, y{\downarrow_{(l+1)}}).$

$(3), (14), (17), (20)$ imply (18). This ends the proof of (16).

$(4), (14), (16)$ imply

$$(21) : I(z, y).$$

$(2), (21)$ imply

$$(22) : A_2(z \cdot y, y).$$

$(9), (11), (22)$ imply

$$(23) : C_2(z \cdot y, y).$$

$(5), (21), (23)$ imply (15). This ends the proof of (12).

To prove (13), let $j \in \mathsf{N}$ and assume that

$$(24) : \langle A_1 \rangle (z, y{\downarrow_j}).$$

(13) follows if it can be shown that

$$(25) : \langle C_1 \rangle (z, y{\downarrow_{j+1}}).$$

In the same way as above, it follows by induction that

$$(26) : I(z, y{\downarrow_j}).$$

$(2), (26)$ imply

$$(27) : \langle A_2 \rangle (z \cdot y{\downarrow_j}, y{\downarrow_j}).$$

$(9), (11), (27)$ imply

$$(28) : \langle C_2 \rangle (z \cdot y{\downarrow_j}, y{\downarrow_{(j+1)}}).$$

$(5), (26), (28)$ imply (25). This ends the proof of (13).

A.2 Proof of Lemma 6

Assume

$$\begin{aligned}
&(1) : (A_1, C_1) \rightsquigarrow \mu\,(A_2, C_2),\\
&(2) : \tau \in [\![\ (A_1, C_1)\ ]\!] \wedge \langle A_1 \rangle (z, \tau(z){\downarrow_j}) \Rightarrow \langle A_2 \rangle (z \cdot \tau(z){\downarrow_j}, \tau(z){\downarrow_j}),\\
&(3) : (A_2, C_2) \text{ is consistent},\\
&(4) : (A_2, C_2) \text{ is fully realizable.}
\end{aligned}$$

It must be shown that there is a predicate

$$I \in (D^{\overline{\infty}})^q \times (D^{\overline{\omega}})^m \to \mathsf{B},$$

such that

$(5) : A_1(z, y) \Rightarrow I(z, y{\downarrow_0})$,
$(6) : I(z, y{\downarrow_j}) \Rightarrow \langle A_2\rangle(z \cdot y{\downarrow_j}, y{\downarrow_j})$,
$(7) : A_1(z, y) \wedge I(z, y{\downarrow_j}) \wedge \langle C_2\rangle(z \cdot y{\downarrow_j}, y{\downarrow_{(j+1)}}) \Rightarrow I(z, y{\downarrow_{(j+1)}})$,
$(8) : A_1(z, y) \wedge \forall k \in \mathsf{N} : I(z, y{\downarrow_k}) \Rightarrow I(z, y)$,
$(9) : I(z, y{\downarrow_j}) \wedge \langle C_2\rangle(z \cdot y{\downarrow_j}, y{\downarrow_{(j+1)}}) \Rightarrow \langle C_1\rangle(z, y{\downarrow_{(j+1)}})$.

Let

$$(10) : I(z, y) \stackrel{\mathsf{def}}{=} \langle A_1\rangle(z, y) \wedge \forall k \in \mathsf{N} : \langle A_2\rangle(z \cdot y{\downarrow_k}, y{\downarrow_k}) \wedge \langle C_2\rangle(z \cdot y{\downarrow_k}, y{\downarrow_k}).$$

To prove (5), assume there are $z \in (D^{\overline{\infty}})^q, y \in (D^{\overline{\infty}})^m$ such that

$$(11) : A_1(z, y).$$

(5) follows if it can be shown that

$$(12) : I(z, y{\downarrow_0}).$$

(3) implies there is a τ such that

$$(13) : \tau \in [\![\ (A_2, C_2) \ ]\!].$$

(11) implies

$$(14) : \langle A_1\rangle(z, y{\downarrow_0}).$$

$(1), (2), (13), (14)$ imply

$$(15) : \langle A_2\rangle(z \cdot y{\downarrow_0}, y{\downarrow_0}).$$

$(13), (15)$ imply

$$(16) : \langle C_2\rangle(z \cdot y{\downarrow_0}, y{\downarrow_0}).$$

$(10), (14), (15), (16)$ imply (12). This ends the proof of (5).

To prove (6), assume there are $z \in (D^{\overline{\infty}})^q, y \in (D^{\overline{\infty}})^m, j \in \mathsf{N}_\infty$ such that

$$(17) : I(z, y{\downarrow_j}).$$

(6) follows if it can be shown that

$$(18) : \langle A_2\rangle(z \cdot y{\downarrow_j}, y{\downarrow_j}).$$

$(10), (17)$ imply (18) if $j < \infty$. Assume

$$(19) : j = \infty.$$

(18) follows if it can be shown that

$(20) : A_2(z \cdot y, y)$.

$(10), (17), (19)$ imply

$(21) : A_1(z,y) \wedge \forall k \in \mathsf{N} : \langle A_2 \rangle (z \cdot y{\downarrow_k}, y{\downarrow_k}) \wedge \langle C_2 \rangle (z \cdot y{\downarrow_k}, y{\downarrow_{(k+1)}})$

We prove by induction on k that there is an infinite sequence $\tau_0, \tau_1, \tau_2, \ldots$ of functions such that for all $k \in \mathsf{N}$

$(22) : \tau_k \in [\![\ (A_2, C_2) \ ]\!]$,
$(23) : y{\downarrow_{(k+1)}} = \tau_k(z \cdot y){\downarrow_{(k+1)}}$.

$(3), (4), (21)$ imply the base-case. $(4), (21)$ imply the induction step.
Let τ be the function such that

$(24) : 0 < k < \infty \wedge (z{\downarrow_{(k-1)}}, y{\downarrow_{(k-1)}}) \sqsubseteq w \not\sqsupseteq (z{\downarrow_k}, y{\downarrow_k}) \Rightarrow \tau(w) = \tau_{(k-1)}(w)$,
$(25) : \tau(z \cdot y) = y$.

τ is clearly well-defined. We now show that τ is pulse-driven. Given $v, u \in (D^{\overline{\infty}})^{(q+m)}$, $j \in \mathsf{N}$ such that

$(26) : v{\downarrow_j} = u{\downarrow_j}$.

It is enough to show that

$(27) : \tau(v){\downarrow_{(j+1)}} = \tau(u){\downarrow_{(j+1)}}$.

There are two cases to consider:

$(28) : v{\downarrow_j} \not\sqsubseteq (z, y)$,
$(29) : v{\downarrow_j} \sqsubseteq (z, y)$.

Assume (28). Then there is a unique $0 < l \leq j$ such that

$(30) : (z{\downarrow_{(l-1)}}, y{\downarrow_{(l-1)}}) \sqsubseteq v$,
$(31) : (z{\downarrow_l}, y{\downarrow_l}) \not\sqsubseteq v$.

$(24), (26), (30), (31)$ imply

$(32) : \tau(v) = \tau_l(v)$,
$(33) : \tau(u) = \tau_l(u)$.

$(26), (32), (33)$ and the pulse-drivenness of the functions imply (27).
Assume (29). $(24), (25), (29)$ imply

$(34) : \tau(v) = y \vee \exists k \in \mathsf{N} : k \geq j \wedge \tau(v) = \tau_k(v)$,
$(35) : \tau(u) = y \vee \exists k \in \mathsf{N} : k \geq j \wedge \tau(u) = \tau_k(u)$.

$(23), (26), (29), (34), (35)$ and the pulse-drivenness of τ_k imply (27). Thus, τ is pulse-driven.
To prove (20), there are two cases to consider:

$(36) : \tau \in [\![\ (A_2, C_2) \ ]\!]$,
$(37) : \tau \notin [\![\ (A_2, C_2) \ ]\!]$.

Assume (36). $(1), (2), (21), (25), (36)$ imply (20).
Assume (37). $(22), (24), (25), (37)$ imply

$$(38) : A_2(z \cdot y, y) \wedge \neg C_2(z \cdot y, y).$$

(38) implies (20). Thus, (20) has been proved. This ends the proof of (6).

To prove (7), assume there are $z \in (D^{\overline{\infty}})^q, y \in (D^{\overline{\infty}})^m, j \in \mathsf{N}_\infty$ such that

$(39) : A_1(z, y)$,
$(40) : I(z, y{\downarrow_j})$,
$(41) : \langle C_2 \rangle(z \cdot y{\downarrow_j}, y{\downarrow_{(j+1)}})$.

(7) follows if it can shown that

$$(42) : I(z, y{\downarrow_{(j+1)}}).$$

If $j = \infty$ then (42) follows trivially. Thus, assume $j < \infty$.
$(3), (4), (10), (40)$ imply (by arguing in the same way as for $(22), (23)$) there is a τ such that

$(43) : \tau \in [\![\ (A_2, C_2) \ ]\!]$,
$(44) : y{\downarrow_{(j+1)}} = \tau(z \cdot y){\downarrow_{(j+1)}}$.

(44) and the pulse-drivenness of τ imply

$$(45) : \mu\, \tau(z){\downarrow_{(j+1)}} \sqsubseteq y.$$

$(1), (2), (39), (43), (45)$ imply

$$(46) : \langle A_2 \rangle(z \cdot y{\downarrow_{(j+1)}}, y{\downarrow_{(j+1)}}).$$

$(43), (44), (46)$ imply

$$(47) : \langle C_2 \rangle(z \cdot y{\downarrow_{(j+1)}}, y{\downarrow_{(j+1)}}).$$

$(10), (39), (40), (46), (47)$ imply (42). This ends the proof of (7).

To prove (8), assume there are $z \in (D^{\overline{\infty}})^q, y \in (D^{\overline{\infty}})^m$ such that

$(48) : A_1(z, y)$,
$(49) : \forall k \in \mathsf{N} : I(z, y{\downarrow_k})$.

(8) follows if it can be shown that

$$(50) : I(z, y).$$

$(10), (48), (49)$ imply (50). This ends the proof of (8).

To prove (9), assume there are $z \in (D^{\overline{\infty}})^q, y \in (D^{\overline{\infty}})^m$, $j \in \mathsf{N}_\infty$ such that

$$(51): I(z, y\downarrow_j),$$
$$(52): \langle C_2 \rangle (z \cdot y\downarrow_j, y\downarrow_{(j+1)}).$$

(9) follows if it can be shown that

$$(53): \langle C_1 \rangle (z, y\downarrow_{(j+1)}).$$

$(3), (4), (10), (51)$ imply (by arguing in the same way as for $(22), (23)$) there is a τ such that

$$(54): \tau \in [\![(A_2, C_2)]\!],$$
$$(55): y\downarrow_{(j+1)} = \tau(z \cdot y)\downarrow_{(j+1)}.$$

$(1), (54)$ imply

$$(56): \mu\,\tau \in [\![(A_1, C_1)]\!].$$

(55) and the pulse-drivenness of τ imply

$$(57): y\downarrow_{(j+1)} \sqsubseteq \mu\,\tau(z).$$

$(10), (51), (56), (57)$ imply (53). This ends the proof of (9).

A.3 Proof of Lemma 1

Assume

$$(1): A_1(z) \Rightarrow I(z, y\downarrow_0),$$
$$(2): I(z, y\downarrow_j) \Rightarrow \langle A_2 \rangle (z \cdot y\downarrow_j),$$
$$(3): I(z, y\downarrow_j) \wedge \langle C_2 \rangle (z \cdot y\downarrow_j, y\downarrow_{(j+1)}) \Rightarrow I(z, y\downarrow_{(j+1)}),$$
$$(4): \forall k \in \mathsf{N}: I(z, y\downarrow_k) \Rightarrow I(z, y),$$
$$(5): I(z, y) \wedge C_2(z \cdot y, y) \Rightarrow C_1(z, y).$$

It must be shown that

$$(6): (A_1, C_1) \rightsquigarrow \mu\,(A_2, C_2).$$

Given

$$(7): \tau \in [\![\mu\,(A_2, C_2)]\!].$$

(6) follows if it can be shown that

$$(8): A_1(z) \Rightarrow C_1(z, \tau(z)).$$

(1)-(4) imply $(A.1.1)$-$(A.1.4)$[4]. Moreover, (5) implies $(A.1.5)$ for the case that $j = \infty$. Since the proof of $(A.1.12)$ relies upon $(A.1.5)$ only for the case that $j = \infty$, it follows that (8) holds.

A.4 Proof of Lemma 3

Assume

[4] By $(A.1.n)$ we mean (n) of Section A.1

$(1) : (A_1, C_1) \rightsquigarrow \mu\,(A_2, C_2),$
$(2) : \tau \in [\![\ (A_1, C_1)\ ]\!] \wedge A_1(z) \Rightarrow A_2(z \cdot \tau(z)),$
$(3) : (A_2, C_2)$ is consistent,
$(4) : (A_2, C_2)$ is fully realizable.

It must be shown that there is a predicate $I \in (D^{\overline{\infty}})^q \times (D^{\overline{\omega}})^m \to \mathsf{B}$ such that

$(5) : A_1(z) \Rightarrow I(z, y{\downarrow_0}),$
$(6) : I(z, y{\downarrow_j}) \Rightarrow \langle A_2 \rangle(z \cdot y{\downarrow_j}),$
$(7) : I(z, y{\downarrow_j}) \wedge \langle C_2 \rangle(z \cdot y{\downarrow_j}, y{\downarrow_{(j+1)}}) \Rightarrow I(z, y{\downarrow_{(j+1)}}),$
$(8) : \forall k \in \mathsf{N} : I(z, y{\downarrow_k}) \Rightarrow I(z, y),$
$(9) : I(z, y) \wedge C_2(z \cdot y, y) \Rightarrow C_1(z, y).$

Let

$$(10) : I(z, y) \stackrel{\mathsf{def}}{=} A_1(z) \wedge \forall k \in \mathsf{N} : \langle A_2 \rangle(z \cdot y{\downarrow_k}) \wedge \langle C_2 \rangle(z \cdot y{\downarrow_k}, y{\downarrow_k}).$$

(1)-(4), (10) imply $(A.2.1)$-$(A.2.4)$, $(A.2.10)$, in which case (5)-(9) follow by Lemma 6 and (10).

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

342/1/90 A	Robert Gold, Walter Vogler: Quality Criteria for Partial Order Semantics of Place/Transition-Nets, Januar 1990
342/2/90 A	Reinhard Fösmeier: Die Rolle der Lastverteilung bei der numerischen Parallelprogrammierung, Februar 1990
342/3/90 A	Klaus-Jörn Lange, Peter Rossmanith: Two Results on Unambiguous Circuits, Februar 1990
342/4/90 A	Michael Griebel: Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen Transformations- Mehrgitter-Methode
342/5/90 A	Reinhold Letz, Johann Schumann, Stephan Bayerl, Wolfgang Bibel: SETHEO: A High-Performance Theorem Prover
342/6/90 A	Johann Schumann, Reinhold Letz: PARTHEO: A High Performance Parallel Theorem Prover
342/7/90 A	Johann Schumann, Norbert Trapp, Martin van der Koelen: SETHEO/PARTHEO Users Manual
342/8/90 A	Christian Suttner, Wolfgang Ertel: Using Connectionist Networks for Guiding the Search of a Theorem Prover
342/9/90 A	Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Jaroslav Kremenek, Robert Lindhof, Thomas Ludwig, Peter Luksch, Thomas Treml: TOPSYS, Tools for Parallel Systems (Artikelsammlung)
342/10/90 A	Walter Vogler: Bisimulation and Action Refinement
342/11/90 A	Jörg Desel, Javier Esparza: Reachability in Reversible Free- Choice Systems
342/12/90 A	Rob van Glabbeek, Ursula Goltz: Equivalences and Refinement
342/13/90 A	Rob van Glabbeek: The Linear Time - Branching Time Spectrum
342/14/90 A	Johannes Bauer, Thomas Bemmerl, Thomas Treml: Leistungsanalyse von verteilten Beobachtungs- und Bewertungswerkzeugen
342/15/90 A	Peter Rossmanith: The Owner Concept for PRAMs
342/16/90 A	G. Böckle, S. Trosch: A Simulator for VLIW-Architectures
342/17/90 A	P. Slavkovsky, U. Rüde: Schnellere Berechnung klassischer Matrix-Multiplikationen
342/18/90 A	Christoph Zenger: SPARSE GRIDS
342/19/90 A	Michael Griebel, Michael Schneider, Christoph Zenger: A combination technique for the solution of sparse grid problems
342/20/90 A	Michael Griebel: A Parallelizable and Vectorizable Multi- Level-Algorithm on Sparse Grids
342/21/90 A	V. Diekert, E. Ochmanski, K. Reinhardt: On confluent semi- commutations-decidability and complexity results
342/22/90 A	Manfred Broy, Claus Dendorfer: Functional Modelling of Operating System Structures by Timed Higher Order Stream Processing Functions

Reihe A

342/23/90 A Rob van Glabbeek, Ursula Goltz: A Deadlock-sensitive Congruence for Action Refinement

342/24/90 A Manfred Broy: On the Design and Verification of a Simple Distributed Spanning Tree Algorithm

342/25/90 A Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Peter Luksch, Roland Wismüller: TOPSYS - Tools for Parallel Systems (User's Overview and User's Manuals)

342/26/90 A Thomas Bemmerl, Arndt Bode, Thomas Ludwig, Stefan Tritscher: MMK - Multiprocessor Multitasking Kernel (User's Guide and User's Reference Manual)

342/27/90 A Wolfgang Ertel: Random Competition: A Simple, but Efficient Method for Parallelizing Inference Systems

342/28/90 A Rob van Glabbeek, Frits Vaandrager: Modular Specification of Process Algebras

342/29/90 A Rob van Glabbeek, Peter Weijland: Branching Time and Abstraction in Bisimulation Semantics

342/30/90 A Michael Griebel: Parallel Multigrid Methods on Sparse Grids

342/31/90 A Rolf Niedermeier, Peter Rossmanith: Unambiguous Simulations of Auxiliary Pushdown Automata and Circuits

342/32/90 A Inga Niepel, Peter Rossmanith: Uniform Circuits and Exclusive Read PRAMs

342/33/90 A Dr. Hermann Hellwagner: A Survey of Virtually Shared Memory Schemes

342/1/91 A Walter Vogler: Is Partial Order Semantics Necessary for Action Refinement?

342/2/91 A Manfred Broy, Frank Dederichs, Claus Dendorfer, Rainer Weber: Characterizing the Behaviour of Reactive Systems by Trace Sets

342/3/91 A Ulrich Furbach, Christian Suttner, Bertram Fronhöfer: Massively Parallel Inference Systems

342/4/91 A Rudolf Bayer: Non-deterministic Computing, Transactions and Recursive Atomicity

342/5/91 A Robert Gold: Dataflow semantics for Petri nets

342/6/91 A A. Heise; C. Dimitrovici: Transformation und Komposition von P/T-Netzen unter Erhaltung wesentlicher Eigenschaften

342/7/91 A Walter Vogler: Asynchronous Communication of Petri Nets and the Refinement of Transitions

342/8/91 A Walter Vogler: Generalized OM-Bisimulation

342/9/91 A Christoph Zenger, Klaus Hallatschek: Fouriertransformation auf dünnen Gittern mit hierarchischen Basen

342/10/91 A Erwin Loibl, Hans Obermaier, Markus Pawlowski: Towards Parallelism in a Relational Database System

342/11/91 A Michael Werner: Implementierung von Algorithmen zur Kompaktifizierung von Programmen für VLIW-Architekturen

342/12/91 A Reiner Müller: Implementierung von Algorithmen zur Optimierung von Schleifen mit Hilfe von Software-Pipelining Techniken

342/13/91 A Sally Baker, Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Udo Graf, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Rainer Knödlseder, Jaroslav Kremenek, Siegfried Langenbuch, Robert Lindhof, Thomas Ludwig, Peter Luksch, Roy Milner, Bernhard Ries, Thomas Treml: TOPSYS - Tools for Parallel Systems (Artikelsammlung); 2., erweiterte Auflage

342/14/91 A Michael Griebel: The combination technique for the sparse grid solution of PDE's on multiprocessor machines

Reihe A

342/15/91 A   Thomas F. Gritzner, Manfred Broy: A Link Between Process Algebras and Abstract Relation Algebras?

342/16/91 A   Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Thomas Treml, Roland Wismüller: The Design and Implementation of TOPSYS

342/17/91 A   Ulrich Furbach: Answers for disjunctive logic programs

342/18/91 A   Ulrich Furbach: Splitting as a source of parallelism in disjunctive logic programs

342/19/91 A   Gerhard W. Zumbusch: Adaptive parallele Multilevel-Methoden zur Lösung elliptischer Randwertprobleme

342/20/91 A   M. Jobmann, J. Schumann: Modelling and Performance Analysis of a Parallel Theorem Prover

342/21/91 A   Hans-Joachim Bungartz: An Adaptive Poisson Solver Using Hierarchical Bases and Sparse Grids

342/22/91 A   Wolfgang Ertel, Theodor Gemenis, Johann M. Ph. Schumann, Christian B. Suttner, Rainer Weber, Zongyan Qiu: Formalisms and Languages for Specifying Parallel Inference Systems

342/23/91 A   Astrid Kiehn: Local and Global Causes

342/24/91 A   Johann M.Ph. Schumann: Parallelization of Inference Systems by using an Abstract Machine

342/25/91 A   Eike Jessen: Speedup Analysis by Hierarchical Load Decomposition

342/26/91 A   Thomas F. Gritzner: A Simple Toy Example of a Distributed System: On the Design of a Connecting Switch

342/27/91 A   Thomas Schnekenburger, Andreas Weininger, Michael Friedrich: Introduction to the Parallel and Distributed Programming Language ParMod-C

342/28/91 A   Claus Dendorfer: Funktionale Modellierung eines Postsystems

342/29/91 A   Michael Griebel: Multilevel algorithms considered as iterative methods on indefinite systems

342/30/91 A   W. Reisig: Parallel Composition of Liveness

342/31/91 A   Thomas Bemmerl, Christian Kasperbauer, Martin Mairandres, Bernhard Ries: Programming Tools for Distributed Multiprocessor Computing Environments

342/32/91 A   Frank Leške: On constructive specifications of abstract data types using temporal logic

342/1/92 A   L. Kanal, C.B. Suttner (Editors): Informal Proceedings of the Workshop on Parallel Processing for AI

342/2/92 A   Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS

342/2-2/92 A   Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS - Revised Version (erschienen im Januar 1993)

342/3/92 A   Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems

342/4/92 A   Claus Dendorfer, Rainer Weber: Development and Implementation of a Communication Protocol - An Exercise in FOCUS

342/5/92 A   Michael Friedrich: Sprachmittel und Werkzeuge zur Unterstüt- zung paralleler und verteilter Programmierung

342/6/92 A   Thomas F. Gritzner: The Action Graph Model as a Link between Abstract Relation Algebras and Process-Algebraic Specifications

Reihe A

342/7/92 A Sergei Gorlatch: Parallel Program Development for a Recursive Numerical Algorithm: a Case Study

342/8/92 A Henning Spruth, Georg Sigl, Frank Johannes: Parallel Algorithms for Slicing Based Final Placement

342/9/92 A Herbert Bauer, Christian Sporrer, Thomas Krodel: On Distributed Logic Simulation Using Time Warp

342/10/92 A H. Bungartz, M. Griebel, U. Rüde: Extrapolation, Combination and Sparse Grid Techniques for Elliptic Boundary Value Problems

342/11/92 A M. Griebel, W. Huber, U. Rüde, T. Störtkuhl: The Combination Technique for Parallel Sparse-Grid-Preconditioning and -Solution of PDEs on Multiprocessor Machines and Workstation Networks

342/12/92 A Rolf Niedermeier, Peter Rossmanith: Optimal Parallel Algorithms for Computing Recursively Defined Functions

342/13/92 A Rainer Weber: Eine Methodik für die formale Anforderungsspezifkation verteilter Systeme

342/14/92 A Michael Griebel: Grid– and point–oriented multilevel algorithms

342/15/92 A M. Griebel, C. Zenger, S. Zimmer: Improved multilevel algorithms for full and sparse grid problems

342/16/92 A J. Desel, D. Gomm, E. Kindler, B. Paech, R. Walter: Bausteine eines kompositionalen Beweiskalküls für netzmodellierte Systeme

342/17/92 A Frank Dederichs: Transformation verteilter Systeme: Von applikativen zu prozeduralen Darstellungen

342/18/92 A Andreas Listl, Markus Pawlowski: Parallel Cache Management of a RDBMS

342/19/92 A Erwin Loibl, Markus Pawlowski, Christian Roth: PART: A Parallel Relational Toolbox as Basis for the Optimization and Interpretation of Parallel Queries

342/20/92 A Jörg Desel, Wolfgang Reisig: The Synthesis Problem of Petri Nets

342/21/92 A Robert Balder, Christoph Zenger: The d-dimensional Helmholtz equation on sparse Grids

342/22/92 A Ilko Michler: Neuronale Netzwerk-Paradigmen zum Erlernen von Heuristiken

342/23/92 A Wolfgang Reisig: Elements of a Temporal Logic. Coping with Concurrency

342/24/92 A T. Störtkuhl, Chr. Zenger, S. Zimmer: An asymptotic solution for the singularity at the angular point of the lid driven cavity

342/25/92 A Ekkart Kindler: Invariants, Compositionality and Substitution

342/26/92 A Thomas Bonk, Ulrich Rüde: Performance Analysis and Optimization of Numerically Intensive Programs

342/1/93 A M. Griebel, V. Thurner: The Efficient Solution of Fluid Dynamics Problems by the Combination Technique

342/2/93 A Ketil Stølen, Frank Dederichs, Rainer Weber: Assumption / Commitment Rules for Networks of Asynchronously Communicating Agents

342/3/93 A Thomas Schnekenburger: A Definition of Efficiency of Parallel Programs in Multi-Tasking Environments

342/4/93 A Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: A Proof of Convergence for the Combination Technique for the Laplace Equation Using Tools of Symbolic Computation

342/5/93 A Manfred Kunde, Rolf Niedermeier, Peter Rossmanith: Faster Sorting and Routing on Grids with Diagonals

342/6/93 A Michael Griebel, Peter Oswald: Remarks on the Abstract Theory of Additive and Multiplicative Schwarz Algorithms

Reihe A

342/7/93 A Christian Sporrer, Herbert Bauer: Corolla Partitioning for Distributed Logic Simulation of VLSI Circuits

342/8/93 A Herbert Bauer, Christian Sporrer: Reducing Rollback Overhead in Time-Warp Based Distributed Simulation with Optimized Incremental State Saving

342/9/93 A Peter Slavkovsky: The Visibility Problem for Single-Valued Surface (z = f(x,y)): The Analysis and the Parallelization of Algorithms

342/10/93 A Ulrich Rüde: Multilevel, Extrapolation, and Sparse Grid Methods

342/11/93 A Hans Regler, Ulrich Rüde: Layout Optimization with Algebraic Multigrid Methods

342/12/93 A Dieter Barnard, Angelika Mader: Model Checking for the Modal Mu-Calculus using Gauß Elimination

342/13/93 A Christoph Pflaum, Ulrich Rüde: Gauß' Adaptive Relaxation for the Multilevel Solution of Partial Differential Equations on Sparse Grids

342/14/93 A Christoph Pflaum: Convergence of the Combination Technique for the Finite Element Solution of Poisson's Equation

342/15/93 A Michael Luby, Wolfgang Ertel: Optimal Parallelization of Las Vegas Algorithms

342/16/93 A Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: Pointwise Convergence of the Combination Technique for Laplace's Equation

342/17/93 A Georg Stellner, Matthias Schumann, Stefan Lamberts, Thomas Ludwig, Arndt Bode, Martin Kiehl und Rainer Mehlhorn: Developing Multicomputer Applications on Networks of Workstations Using NXLib

342/18/93 A Max Fuchs, Ketil Stølen: Development of a Distributed Min/Max Component

342/19/93 A Johann K. Obermaier: Recovery and Transaction Management in Write-optimized Database Systems

342/20/93 A Sergej Gorlatch: Deriving Efficient Parallel Programs by Systemating Coarsing Specification Parallelism

342/01/94 A Reiner Hüttl, Michael Schneider: Parallel Adaptive Numerical Simulation

342/02/94 A Henning Spruth, Frank Johannes: Parallel Routing of VLSI Circuits Based on Net Independency

342/03/94 A Henning Spruth, Frank Johannes, Kurt Antreich: PHIroute: A Parallel Hierarchical Sea-of-Gates Router

342/04/94 A Martin Kiehl, Rainer Mehlhorn, Matthias Schumann: Parallel Multiple Shooting for Optimal Control Problems Under NX/2

342/05/94 A Christian Suttner, Christoph Goller, Peter Krauss, Klaus-Jörn Lange, Ludwig Thomas, Thomas Schnekenburger: Heuristic Optimization of Parallel Computations

342/06/94 A Andreas Listl: Using Subpages for Cache Coherency Control in Parallel Database Systems

342/07/94 A Manfred Broy, Ketil Stølen: Specification and Refinement of Finite Dataflow Networks - a Relational Approach

342/08/94 A Katharina Spies: Funktionale Spezifikation eines Kommunikationsprotokolls

342/09/94 A Peter A. Krauss: Applying a New Search Space Partitioning Method to Parallel Test Generation for Sequential Circuits

342/10/94 A Manfred Broy: A Functional Rephrasing of the Assumption/Commitment Specification Style

342/11/94 A Eckhardt Holz, Ketil Stølen: An Attempt to Embed a Restricted Version of SDL as a Target Language in Focus

342/12/94 A Christoph Pflaum: A Multi-Level-Algorithm for the Finite-Element-Solution of General Second Order Elliptic Differential Equations on Adaptive Sparse Grids

Reihe A

342/13/94 A	Manfred Broy, Max Fuchs, Thomas F. Gritzner, Bernhard Schätz, Katharina Spies, Ketil Stølen: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems
342/14/94 A	Maximilian Fuchs: Technologieabhängigkeit von Spezifikationen digitaler Hardware
342/15/94 A	M. Griebel, P. Oswald: Tensor Product Type Subspace Splittings And Multilevel Iterative Methods For Anisotropic Problems
342/16/94 A	Gheorghe Ştefănescu: Algebra of Flownomials
342/17/94 A	Ketil Stølen: A Refinement Relation Supporting the Transition from Unbounded to Bounded Communication Buffers
342/18/94 A	Michael Griebel, Tilman Neuhoeffer: A Domain-Oriented Multilevel Algorithm-Implementation and Parallelization
342/19/94 A	Michael Griebel, Walter Huber: Turbulence Simulation on Sparse Grids Using the Combination Method
342/20/94 A	Johann Schumann: Using the Theorem Prover SETHEO for verifying the development of a Communication Protocol in FOCUS - A Case Study -
342/01/95 A	Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids
342/02/95 A	Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law
342/03/95 A	Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space
342/04/95 A	Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes
342/05/95 A	Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems
342/06/95 A	Maximilian Fuchs: Formal Design of a Model-N Counter
342/07/95 A	Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology
342/08/95 A	Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies
342/09/95 A	Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B	Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B	Jörg Desel: On Abstraction of Nets
342/3/90 B	Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B	Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B	Barbara Paech1: Concurrency as a Modality
342/2/91 B	Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox - Anwenderbeschreibung
342/3/91 B	Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B	Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B	Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B	Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B	W. Reisig: Concurrent Temporal Logic
342/1/92 B	Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B	Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B	Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B	Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B	Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS