TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

A Formal Method for Hardware/Software Co-Design

Ketil Stølen, Max Fuchs

TUM-I9517
SFB-Bericht Nr.342/10/95 A
Mai 1995

TUM–INFO–05-95-I17-350/1.–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

A Formal Method for Hardware/Software Co-Design

Ketil Stølen, Max Fuchs
Institut für Informatik, TU München, D-80290 München
email:stoelen@informatik.tu-muenchen.de

November 6, 1995

Abstract

This paper presents a formal method supporting hardware/software co-design with respect to specification and verification. We introduce three different specification formats. Two of these are intended for the specification of asynchronous software; the third is more suited for digital hardware applications. All three formats are based on the assumption/commitment paradigm. We introduce a refinement relation and formulate verification rules for the parallel composition of specifications. We apply the proposed method to specify and decompose a timed FIFO queue which is partly to be implemented in hardware and partly to be implemented in software.

1 Introduction

The label "hardware/software co-design" is used to denote the simultaneous design of both hardware and software to implement a desired function or specification. Hardware/software co-design is still a very new research area and is currently receiving considerable interest. See for example [Buc94].
The objective of this paper is to present a formal method supporting hardware/software co-design. We first introduce a semantic framework based on streams and pulse-driven functions. A simple composition operator is used to compose pulse-driven functions into networks of pulse-driven functions — networks which, when observed from the outside, themselves can be understood as pulse-driven functions. Thus, our model is fully compositional.
Specifications explicitly characterize the relationship between the complete communication histories of the input channels and the complete communication histories of the output channels. Specifications can be written at several levels of abstraction. We introduce three specification formats. In each format a specification is modeled by a set of pulse-driven functions modulo a particular abstraction. We refer to this set as the specification's denotation. The two first formats, which we call respectively time dependent and time independent, are mainly intended for the specification of software components or, more explicitly, interfaces communicating asynchronously. If real-time constraints are to be imposed then the time dependent format must be

employed; otherwise the time independent format should be chosen. The third format, which we call synchronous, is mainly intended for the specification of hardware. All three formats are based on the so-called assumption/commitment paradigm.
Since any specification is modeled by a set of pulse-driven functions the composition operator can be lifted from functions to specifications in a straightforward way. We may then express system specifications containing component specifications of all three formats.
A specification is said to refine another specification if its denotation is contained in that of the latter. We give verification rules for the composition of specifications modulo this refinement principle. We also explain how conversion rules can be used to translate a specification of one format into a specification of another format.
Thus, in our approach there is a uniform, common semantic basis for specification formats, composition and refinement. Our method is compositional and well-suited for top-down design. We can handle both synchronous and asynchronous communication. Real-time constraints can be specified, and we have specially designed rules for the verification of refinement steps. This allows hardware/software co-design to be conducted in a very elegant way.
We employ our approach to specify and decompose a timed FIFO queue which is partly to be implemented in hardware and partly to be implemented in software.
The paper is organized as follows. Section 2 introduces the basic concepts and notations. In Section 3 the three specification formats are introduced. Refinement and refinement rules are the subjects of Section 4. Section 5 is devoted to the case-study. Finally, Section 6 contains a brief summary and relates our approach to other approaches known from the literature.

2 Semantic Model

We represent the communication histories of channels by timed streams. A timed stream is a finite or infinite sequence of messages and time ticks. A time tick is represented by "$\surd$". The interval between two consecutive ticks represents the least unit of time. A tick occurs in a stream at the end of each time unit.
An infinite timed stream represents a complete communication history, a finite timed stream represents a partial communication history. Since time never halts, any infinite timed stream is required to have infinitely many ticks. We do not want timed streams to end in the middle of a time unit. Thus, we insist that a timed stream is either empty, infinite or ends with a tick.
Given a set of messages M, then $M^{\overline{\infty}}$, $M^{\overline{*}}$ and $M^{\overline{\omega}}$ denote respectively the set of all infinite timed streams over M, the set of all finite timed streams over M, and the set of all finite and infinite timed streams over M. We use N to denote the set of natural numbers, and N_∞ to denote $\mathsf{N} \cup \{\infty\}$. Given $s \in M^{\overline{\omega}}$ and $j \in \mathsf{N}_\infty$, $s{\downarrow_j}$ denotes the prefix of s characterizing the behavior until time j, i.e., $s{\downarrow_j}$ denotes s if j is greater than the number of ticks in s, and the shortest prefix of s containing j ticks, otherwise. Note that $s{\downarrow_\infty} = s$. The operator is overloaded to tuples of timed streams t in a point-wise style, i.e., $t{\downarrow_j}$ denotes the tuple we get by applying $\downarrow_j$ to each component of t.
A named stream tuple is a mapping $\alpha \in a \rightarrow M^{\overline{\omega}}$ from a set of channel identifiers

to timed streams. Intuitively, α assigns a (possibly partial) communication history to each channel named by the channel identifiers in a. The operator $\downarrow$ is overloaded to named stream tuples in the same point-wise style as for tuples of timed streams. Given two named stream tuples

$$\alpha \in a \to M^{\overline{\omega}}, \qquad \beta \in b \to M^{\overline{\omega}},$$

if $a \cap b = \emptyset$ then $\alpha \uplus \beta$ denotes the element of $a \cup b \to M^{\overline{\omega}}$ such that

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

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

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

A function

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

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

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

Pulse-drivenness means that the input until time j completely determines the output until time $j+1$. In other words, a pulse-driven function imposes a delay of at least one time unit between input and output and is in addition "lazy" in the sense that it can be (partially) computed based on partial input. We use the arrow $\stackrel{p}{\to}$ to distinguish pulse-driven functions from functions that are not pulse-driven.
We model specifications by sets of pulse-driven functions. Each function or subset of functions contained in such a set represents one possible implementation. For example, a specification of a component, whose input and output channels are named by i and o, respectively, is modeled by a set of pulse-driven functions F such that

$$F \subseteq (i \to M^{\overline{\infty}}) \stackrel{p}{\to} (o \to M^{\overline{\infty}}).$$

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

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

where $i_1 \cap i_2 = o_1 \cap o_2 = i_1 \cap o_1 = i_2 \cap o_2 = \emptyset$, is characterized by $\tau_1 \otimes \tau_2$. Informally speaking, any output channel of τ_1 and input channel of τ_2, and any output channel of τ_2 and input channel of τ_1, whose names are identical, are connected and hidden in the sense that they cannot be observed from the outside.

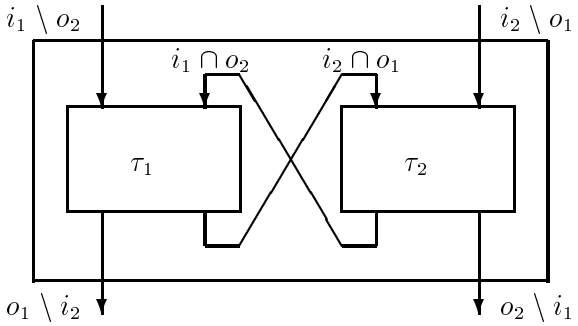

Figure 1: Network Characterized by $\tau_1 \otimes \tau_2$

Given that

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

then for all $\alpha \in i \to M^{\overline{\infty}}$, we define

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

Note that pulse-drivenness implies[1] that for any α there are unique ψ, θ such that

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

Thus, $\tau_1 \otimes \tau_2$ is well-defined. It is also easy to prove that $\tau_1 \otimes \tau_2$ is pulse-driven. This implies

$$\tau_1 \otimes \tau_2 \in (i \to M^{\overline{\infty}}) \xrightarrow{p} (o \to M^{\overline{\infty}}).$$

Given $n > 1$ pulse-driven functions

$$\tau_j \in (i_j \to M^{\overline{\infty}}) \xrightarrow{p} (o_j \to M^{\overline{\infty}}),$$

such that $i_j \cap o_j = \emptyset$, and $j \neq k \Rightarrow i_j \cap i_k = o_j \cap o_k = \emptyset$, then $\otimes_{j=1}^{n} \tau_j$ is a short-hand for $\tau_1 \otimes \ldots \otimes \tau_n$. Note that the restrictions imposed on the identifier sets imply that $\otimes$ is associative — this explains why there are no brackets.
As will be shown below, the composition operator $\otimes$ can be lifted from functions to specifications in a straightforward way.

3 Specification

Based on the semantic model introduced in the previous section, we may write specifications at several levels of abstraction. Below we give three different specification

[1] As a consequence of Banach's fix-point theorem [AdBKR89], since pulse-driven functions can be understood as contracting functions in a complete metric space.

formats. However, first we introduce some useful operators on streams.

3.1 Operators on Streams

We also use streams without ticks. We refer to such streams as untimed. Given a set of messages M, then M^∞, M^* and M^ω denote respectively the set of all infinite untimed streams over M, the set of all finite untimed streams over M, and the set of all finite and infinite untimed streams over M.
Given $A \subseteq M \cup \{\surd\}$, (timed or untimed) streams r and s over M, $n \in \mathsf{N}_\infty$ and integer j:

- $\#r$ denotes the length of r, i.e. ∞ if r is infinite, and the number of elements in r otherwise. Note that time ticks are counted. For example the length of a stream consisting of infinitely many ticks is ∞.
- $r.j$ denotes the j'th element of r if $1 \leq j \leq \#r$.
- $\langle a_1, a_2, .., a_n \rangle$ denotes the stream of length n whose first element is a_1, whose second element is a_2, and so on. $\langle\rangle$ denotes the empty stream.
- $A©r$ denotes the result of removing all messages (ticks included) not in A, i.e., the projection of r on A. If $A = \{d\}$ we write $d©r$ instead of $\{d\}©r$. For example

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

- $r|_j$ denotes $\langle\rangle$ if $j \leq 0$, the prefix of r of length j if $0 < j < \#r$, and r otherwise. This means that $r|_\infty = r$. This operator is overloaded to stream tuples in a point-wise way. Note the difference with respect to $\downarrow$.
- $r \frown s$ denotes the result of concatenating r to s. Thus, $\langle c, c\rangle \frown \langle a, b\rangle = \langle c, c, a, b\rangle$. If r is infinite then $r \frown s = r$.
- $r \sqsubseteq s$ holds iff r is a prefix of s. Also this operator is overloaded to tuples of streams in the obvious way.
- $\mathsf{sy}(r)$ holds iff r is a timed stream, and exactly one message occurs between two consecutive ticks, i.e.: $\forall j \in \mathsf{N} : 0 \leq j \leq \#\surd©r \Rightarrow \#M©(r{\downarrow_j}) = j$. Any stream r such that $\mathsf{sy}(r)$ is said to be synchronous.
- $\overline{r}$ denotes the result of removing all ticks in r. Thus, $\overline{\langle a, \surd, b, \surd\rangle} = \langle a, b\rangle$.

3.2 Time Dependent Specifications

To specify real-time components some explicit notion of time is required. We refer to such specifications as time dependent. A time dependent specification is written in the form

$$S \stackrel{\mathsf{td}}{\equiv} (i \rhd o) :: \mathsf{ass} : A \; \mathsf{com} : C$$

S is the specification's name; i is a finite list of input identifiers; o is a finite list of output identifiers — i and o have no identifier in common and are both without repetitions in the sense that the same identifier occurs only once; td is a label distinguishing time dependent specifications from other specification formats; A and C are formulas whose free variables are contained in $i \cup o$[2]. The identifiers in i name the input channels, and the identifiers in o name the output channels. We refer to (i, o) as the specification's interface. In A and C each input and output identifier represents an infinite timed stream. In the tradition of [Hoa69], [Jon81], [MC81] the formula A characterizes the assumptions about the environment in which the specified component is supposed to run. C, on the other hand, characterizes what the component is committed to do whenever it is executed in an environment which behaves in accordance with the assumption. Such specifications are normally called assumption/commitment specifications. This is the motivation for the keywords ass and com.

For simplicity, in this section the input/output identifiers are untyped. However, the introduction of types is a straightforward extension, and this is exploited in the examples.

For any formula P and mapping $m \in I \to D$ such that the free variables in P are contained in I and vary over D, $m \models P$ holds iff P evaluates to true whenever each free variable i in P is interpreted as $m(i)$.

For any formula P, identifier x and expression y, $P[^x_y]$ denotes the result of replacing each occurrence of x in P by y. This substitution operator is generalized in the obvious way for the case that x is a repetition free list of identifiers and y is a list of expressions of the same length as x.

For any formula P, whose free variables vary over $M^{\overline{\infty}}$, we use $\langle P \rangle$ to denote its prefix closure. $\langle P \rangle$ has the same free variables as P. However, in $\langle P \rangle$ they vary over $M^{\overline{\omega}}$. Formally, if s is the list of free variables in P, then $\langle P \rangle$ denotes $s \in M^{\overline{\omega}} \wedge \exists q \in M^{\overline{\infty}} : P[^s_q] \wedge s \sqsubseteq q$ for some list of variables q disjoint from s. Note that $\langle P \rangle$ is not necessarily a safety property. For example, for infinite timed streams P and $\langle P \rangle$ are equivalent.

The denotation of a time dependent specification S is the set of all pulse-driven functions

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

such that

$$\forall \alpha \in i \to M^{\overline{\infty}} : j \in \mathsf{N}_\infty : (\alpha \uplus \tau(\alpha)\downarrow_j) \models \langle A \rangle \Rightarrow (\alpha \uplus \tau(\alpha)\downarrow_{(j+1)}) \models \langle C \rangle.$$

Thus

- $(j = \infty)$: if the environment always behaves in accordance with the assumption, then τ always behaves in accordance with the commitment,
- $(j < \infty)$: if the environment behaves in accordance with the assumption until time j, then τ behaves in accordance with the commitment until time $j + 1$.

[2]Any list without repetitions can be represented as a totally ordered set, and the other way around. We therefore often apply set operators to such lists without first conducting a conversion.

This one-step-longer-than semantics is closely related to the semantics of the $\stackrel{+}{\rightarrow}$ operator in [AL93]. Note the way $\models$ allows us to represent variables over domains of streams (at the syntactic level) by named stream tuples (at the semantic level). Throughout this paper, for any specification S, we use $[\![\ S\ ]\!]$ to represent its denotation. Moreover, for any assumption/commitment specification S, by A_S and C_S we denote its assumption and commitment, respectively.

$$\mathrm{INT} \stackrel{\mathsf{td}}{\equiv} (x \in M^{\overline{\infty}}, s \in {D_+}^{\overline{\infty}} \rhd y \in D^{\overline{\infty}}, r \in {M_+}^{\overline{\infty}}) ::$$

ass :

$$\mathsf{sy}(x) \wedge \forall m \in M^{\overline{*}} : m \sqsubseteq x \Rightarrow \#(\odot \copyright m) \leq \#(D_+ \copyright m)$$

com :

$$\mathsf{sy}(y) \wedge D_+ \copyright y = D \copyright s \wedge M \copyright r = M_+ \copyright x$$

Figure 2: Specification of an Interface Component

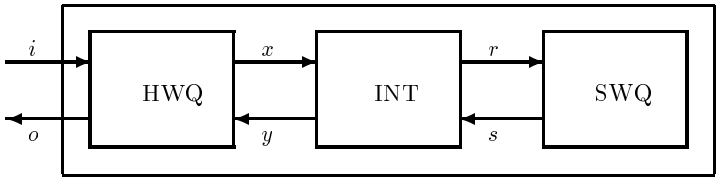

Figure 3: FIFO Queue.

Example 1 Specification of an Interface Component:
To show the use of the time dependent format, we specify an interface component INT (see Figure 2). It is later used to connect a software queue SWQ to a hardware queue HWQ (see Figure 3). The resulting network is supposed to behave as a FIFO queue. HWQ is to be realized in hardware and has therefore only a bounded amount of memory. The hardware queue is supposed to take advantage of the other two components of the network when its memory gets filled up. On the other hand, SWQ is to be implemented in software and is required to have an unbounded memory[3]. Since the hardware queue communicates in a synchronous manner and the software queue communicates in an asynchronous manner, we need the interface component as a converter. INT forwards data elements and requests received on x (sent by the hardware queue) along r (to the software queue), and forwards replies received on s (from the software queue) along y (to the hardware queue).
Let D be some set such that $0 \in D$ and $\odot \notin D$. We use M to denote $D \cup \{\odot\}$, D_+ to denote $D \setminus \{0\}$, and M_+ to denote $M \setminus \{0\}$. We refer to D_+ as the set

[3]Of course, in practice also the software queue will only have a bounded amount of memory. However, for simplicity we ignore this additional complication in this paper.

of data elements, and to $\odot$ as a request. The hardware queue communicates in a synchronous manner. When no real message can be sent within a certain time unit, it outputs a default message denoted by 0. Similarly, INT forwards a 0 along y when it has nothing else to send within a certain time unit.
We now go through the specification of the interface in Figure 2. The first conjunct of the assumption requires the input on x to be synchronous. This is a sensible assumption since the hardware queue communicates in a synchronous manner. Note that this conjunct expresses a real-time requirement. The second conjunct requires that the number of requests received on x until some time j is always less than or equal to the number of data elements received until time j.
The first conjunct of the commitment insists that the output along y is synchronous. The second conjunct makes sure that any data element received on s eventually is forwarded along y (and that no other data element is output along y). If at a certain point in time no data element can be sent, the specified component must forward a 0. Finally, the third conjunct requires that any data element or request received on x eventually is forwarded along r (and that no other message is output along r). $\Box$

The time dependent format does not allow the specifier to fix the least unit of time, i.e., for instance to express that the least unit of time (the interval between two consecutive ticks) is equal to 2 seconds. However, this is a straightforward generalization. It has been left out here because it is of little importance in the examples below.

3.3 Time Independent Specifications

For many applications explicit timing is not really necessary for the simple reason that there are no real-time constraints to be imposed. In those cases we want to abstract from time ticks. We refer to such specifications as time independent. A time independent specification is written in the form

$$S \overset{\mathsf{ti}}{\equiv} (i \rhd o) :: \mathsf{ass} : A \ \mathsf{com} : C$$

As in the previous case: S is the specification's name; i is a finite list of input identifiers (without repetitions); o is a finite list of output identifiers (without repetitions and disjoint from i); ti is a label; A and C are formulas whose free variables are contained in $i \cup o$. However, contrary to the previous case, in A and C each such identifier now represents an untimed stream. Moreover, this untimed stream may be finite (since an infinite timed stream with only finitely many messages degenerates to a finite stream when the ticks are removed).
For any $\alpha \in a \to M^{\overline{\omega}}$ we use $\overline{\alpha}$ to denote the element of $a \to M^{\omega}$ such that $\forall c \in a : \overline{\alpha}(c) = \overline{\alpha(c)}$. The prefix operator is overloaded to formulas whose free variables vary over M^{ω} in the obvious way. By $\langle P \rangle_o$ we denote the prefix closure of P with respect to the identifiers in o.
The denotation of a time independent specification S can then be defined as the set of all pulse-driven functions

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

such that for all $\alpha \in i \to M^{\overline{\infty}}$ we have that

$$\overline{(\alpha \uplus \tau(\alpha))} \models A \Rightarrow \overline{(\alpha \uplus \tau(\alpha))} \models C,$$

$$\forall j \in \mathsf{N} : \overline{(\alpha \uplus \tau(\alpha)\downarrow_j)} \models \langle A \rangle_o \Rightarrow \overline{(\alpha \uplus \tau(\alpha)\downarrow_{(j+1)})} \models \langle C \rangle_o.$$

Because of the time abstraction, the constraints for finite and infinite time cannot easily be merged into one constraint as in the time dependent case. There is a close relationship between time independent and time dependent specifications. For any time independent specification S, we define $[S]$ to denote the specification that can be obtained from S by substituting the label td for ti and by, for any input/output identifier v, replacing any free occurrence of v in A_S and C_S by $\overline{v}$. We then have that $[\![S]\!] = [\![[S]]\!]$. Thus, any time independent specification can be translated into an equivalent time dependent specification in this way. A translation in the other direction is of course normally not possible since the time dependent format is more expressive.

$$\mathrm{SWQ} \overset{\mathsf{ti}}{\equiv} (r \in {M_+}^{\omega} \rhd s \in {D_+}^{\omega}) ::$$

ass :

$$\forall m \in M^* : m \sqsubseteq r \Rightarrow \#(\odot \copyright m) \leq \#(D \copyright m)$$

com :

$$s = (D \copyright r)|_{\#(\odot \copyright r)}$$

Figure 4: Specification of a FIFO Queue

Example 2 Software Queue:
To show the use of the time independent format, we specify (see Figure 4) the software queue mentioned in Example 1. It is supposed to work in a FIFO style. D_+ and M_+ are defined as in Example 1. $\odot$ models a request, and D_+ models the set of data elements.
The assumption requires that the queue never receives a request when it is empty. The commitment, on the other hand, insists that any request receives a reply, and that data elements are output in the correct FIFO manner. □

3.4 Synchronous Specifications

The two formats introduced above are intended for the specification of components communicating asynchronously. To specify components communicating in a synchronous (pulsed) style, like for example digital hardware components, we introduce

a third format. We refer to the specifications written in this format as synchronous. A synchronous specification is of the form

$$S \overset{\mathsf{sy}}{\equiv} (i \rhd o) :: \mathsf{ass} : A \ \mathsf{com} : C$$

As before A and C are formulas whose free variables are contained in $i \cup o$. However, contrary to the previous cases, in A and C each such free identifier now represents an infinite untimed stream.
If $\alpha \in a \rightarrow M^{\overline{\infty}}$, then $\mathsf{sy}(\alpha)$ holds iff $\forall c \in a : \mathsf{sy}(\alpha(c))$. The prefix operator is overloaded to formulas whose free variables vary over M^{∞} in the obvious way. The denotation of a synchronous specification can then be defined as the set of all pulse-driven functions

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

such that

$$\forall \alpha \in i \rightarrow M^{\overline{\infty}} : j \in \mathsf{N}_{\infty} : \\ \mathsf{sy}(\alpha) \wedge \overline{(\alpha \uplus \tau(\alpha)\downarrow_j)} \models \langle A \rangle \Rightarrow \overline{(\alpha \uplus \tau(\alpha)\downarrow_{(j+1)})} \models \langle C \rangle \wedge \mathsf{sy}(\tau(\alpha)\downarrow_{(j+1)}).$$

As in the time dependent case, the constraints for finite and infinite time are merged. This is easy, since for synchronous streams, we can still distinguish between partial and total input after time abstraction. As in the time independent case, there is a close relationship between synchronous and time dependent specifications. For any synchronous specification S, we define $[S]$ to denote the specification that can be obtained from S by substituting the label td for sy, by adding a conjunct $\mathsf{sy}(i)$ to the assumption for each input identifier i, by adding a conjunct $\mathsf{sy}(o)$ to the commitment for each output identifier o, and by for any input/output identifier v, replacing any free occurrence of v in A_S and C_S by $\overline{v}$. We then have that $[\![S]\!] = [\![[S]]\!]$. Thus, any synchronous specification can be translated into an equivalent time dependent specification in this way. A translation in the other direction is of course normally not possible since the time dependent format is more expressive.

Example 3 Hardware Queue:
We once more specify (see Figure 5) a FIFO queue, namely the hardware queue mentioned in Example 1.
More precisely, we want to specify a component HWQ which communicates with INT of Example 1 and SWQ of Example 2 in accordance with Figure 3.
The first conjunct of the assumption is similar to the previous cases. Clearly, the hardware queue can only be required to behave correctly as long as the rest of the network behaves as a FIFO queue. This is stated by the second and third conjunct of the assumption. The second states that what the component receives on y is what it has already sent along x. The third states a liveness property, namely that it receives a data element on y for each request it sends along x.
The first conjunct of the commitment requires that the hardware queue will always reply to a request and output data-elements in the right FIFO manner. In addition we must make sure that the hardware queue uses the software queue correctly, i.e., it

$$\mathrm{HWQ} \overset{\mathsf{sy}}{\equiv} (i \in M^\infty, y \in D^\infty \rhd o \in D^\infty, x \in M^\infty) ::$$

ass :

$$\forall m \in M^* : m \sqsubseteq i \Rightarrow \#(\odot \copyright m) \leq \#(D_+ \copyright m) \wedge$$

$$\forall j \in \mathsf{N} : D_+ \copyright (y|_{(j+1)}) \sqsubseteq (D_+ \copyright (x|_j))|_{\#(\odot \copyright (x|_j))} \wedge$$

$$\#(D_+ \copyright y) = \#(\odot \copyright x)$$

com :

$$D_+ \copyright o = (D_+ \copyright i)|_{\#(\odot \copyright i)} \wedge$$

$$\forall m \in M^* : m \sqsubseteq x \Rightarrow \#(\odot \copyright m) \leq \#(D_+ \copyright m)$$

Figure 5: Specification of a Hardware Queue.

should only send requests when the software queue is not empty. This requirement is stated by the second conjunct of the commitment. □

Synchronous specifications can be used to specify components communicating synchronously in the sense that along each channel exactly one message is sent per unit of time. Of course, as in the time dependent case, more general formats can be formulated in which the specifier himself can fix the pulse-frequency with respect to the least unit of time.

3.5 Composing Specifications

Since the denotation of a specification is a set of pulse-driven functions, the operator $\otimes$ can be lifted from pulse-driven functions to specifications in a straightforward way. Let $i_1 \cap i_2 = o_1 \cap o_2 = \emptyset$, $i = (i_1 \setminus o_2) \cup (i_2 \setminus o_1)$ and $o = (o_1 \setminus i_2) \cup (o_2 \setminus i_1)$. If S_1 and S_2 are specifications whose interfaces are characterized by (i_1, o_1) and (i_2, o_2), respectively, then $[\![S_1 \otimes S_2]\!]$ denotes the set of all $\tau \in (i \to M^{\overline{\infty}}) \xrightarrow{p} (o \to M^{\overline{\infty}})$ such that

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

Moreover, the overall specification has (i, o) as its external interface. Thus, the channels used to connect S_1 and S_2 are hidden. The operator $\otimes_{j=1}^n$ can of course be lifted accordingly.
For example the network pictured in Figure 3 is characterized by

$$\mathrm{HWQ} \otimes \mathrm{INT} \otimes \mathrm{SWQ}.$$

Thus, we may write specifications containing sub-specifications of all three formats. This allows for example hardware/software co-design to be conducted in an elegant way.

4 Refinement

A specification formalism as introduced in the previous section is of little value on its own. To take advantage of a specification formalism we need to know exactly what it means for a system to implement or satisfy a specification. To move from an abstract requirement specification to a concrete realization in one step is very hard — if at all possible. Instead we advocate a step-wise development in the sense that the requirement specification is gradually refined into its realization via a number of intermediate specifications. In fact one may think of the final implementation as just another specification. What is needed is therefore a formalization of what it means for a specification to refine another specification.
Clearly, this refinement relation should guarantee that any behavior of the more concrete specification is also a behavior of the more abstract specification. Given two specifications S_1 and S_2, we write $S_1 \rightsquigarrow S_2$ if S_2 is a refinement of S_1. Formally

$$S_1 \rightsquigarrow S_2 \Leftrightarrow [\![S_2]\!] \subseteq [\![S_1]\!].$$

Thus, a specification S_2 is said to refine a specification S_1 iff they have the same syntactic interface, and any behavior of S_2 is also a behavior of S_1. This refinement relation is reflexive, transitive and a congruence modulo specification composition. Hence, $\rightsquigarrow$ admits compositional system development in the sense that

- Design decisions can be verified at the point in a development where they are taken, i.e., independent of how the component specifications are implemented later on.
- Once a specification has been decomposed into a network of sub-specifications, and the correctness of this decomposition has been verified, any further refinement of these sub-specifications can be carried out in isolation, i.e., independent of the other component specifications and their refinements.

Because $\rightsquigarrow$ allows the semantic behavior of a specification to be refined, this refinement relation is normally referred to as behavioral refinement. To simplify system developments it is often helpful to also refine the syntactic interfaces of specifications. For that purpose a more general refinement relation is needed, namely what is usually referred to as interface refinement. In the tradition of [Hoa72], interface refinement is basically behavioral refinement modulo a representation function relating the concrete to the abstract interface. A detailed discussion of interface refinement is beyond the scope of this paper. Here we just want to point out that our approach also supports this more general notion of refinement.

The formulation of verification rules for assumption/commitment specifications with respect to a parallel operator like $\otimes$ is known to be a difficult task. The main

reason is that the component specifications can be mutually dependent — a fact which easily leads to circular reasoning. We now want to formulate such a rule with respect to time dependent specifications. For this purpose we introduce the following convention: $P(a\downarrow_j)$ and $P(a\downarrow_j, b\downarrow_j)$ are short-hands for $P[^a_{a\downarrow_j}]$ and $P[^{a\ b}_{a\downarrow_j\ b\downarrow_j}]$, respectively. Moreover, we use $(P_i)_{i\in[1..n]}$ as a short-hand for $P_1 \wedge \ldots \wedge P_n$.
Given n time dependent specifications $S_1, S_2, \ldots, S_n$. Let z_i, x_i and y_i be repetition free lists consisting of respectively the input channels of S_i that are connected to the overall environment, the input channels of S_i that are connected to the other $n-1$ component specifications, and the output channels of S_i that are either connected to the other $n-1$ specifications or to the overall environment. We require that z_i, x_i and y_i are mutually disjoint. We also require that[4]

$$i \neq k \Rightarrow (z_i \cup x_i) \cap (z_k \cup x_k) = \emptyset = y_i \cap y_k.$$

This means we have point-to-point communication — the same channel is accessed by only two components, namely the "receiver" and the "sender".
Let z and y be repetition free lists such that $z = \cup_{i=1}^n z_i$ and $y = \cup_{i=1}^n y_i$. It seems sensible to base the rule on some sort of computational induction. This requires the formulation of an induction hypotheses. This hypotheses must be constrained to hold initially and to be maintained by each computation step as long as the overall assumption remains valid. In the field of program verification such an induction hypotheses is normally referred to as an invariant. Let I be this invariant. We assume that I is a formula whose free variables are contained in $z \cup y$. In each proof-obligation below the elements of z and y are universally quantified over infinite timed streams. Moreover, any free occurrence of j is universally quantified over N.
As already mentioned, it must be shown that the invariant holds initially, i.e., at time 0. This is secured through the following proof obligation

$$A_S \Rightarrow I(y\downarrow_0).$$

Secondly, it must be shown that whenever the invariant holds at time j, then it also holds at time $j+1$. A correct implementation of a time dependent assumption/commitment specification is required to satisfy the commitment one step longer than the environment satisfies the assumption. Thus, we first have to make sure that whenever the invariant holds at time j, then the component assumptions have not been falsified. This is ensured through the following proof obligation

$$I(y\downarrow_j) \Rightarrow \langle A_{S_i}\rangle(y\downarrow_j)_{i\in[1..n]}.$$

Given this proof-obligation, the invariant is maintained by each computation step, if it can be shown that

$$I(y\downarrow_j) \wedge \langle C_{S_i}\rangle(x_i\downarrow_j, y_i\downarrow_{j+1})_{i\in[1..n]} \Rightarrow I(y\downarrow_{j+1}).$$

We now have three proof obligations. Together they guarantee that the invariant

[4] We use set notation since there is a one-to-one mapping between repetition free lists and totally ordered sets.

holds after any finite number of computation steps. To make sure that the invariant holds at infinite time, the following proof obligation is sufficient

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

To guarantee the validity of the conclusion two things must be shown. Firstly, it must be shown that if the overall assumption A_S has not been falsified at some finite time j then the overall commitment C_S holds at least one step longer. This is ensured by reformulating the third proof obligation as below

$$I(y{\downarrow_j}) \wedge \langle C_{S_i}\rangle(x_i{\downarrow_j}, y_i{\downarrow_{j+1}})_{i\in[1..n]} \Rightarrow I(y{\downarrow_{j+1}}) \wedge \langle C_S\rangle(y{\downarrow_{j+1}}).$$

Secondly, it must be shown that if the overall assumption A_S holds at infinite time then the overall commitment C_S holds at infinite time. To guarantee this it is enough to widen the scope of j in the second proof obligation from N to N_∞ and to add the following proof obligation

$$I \wedge (C_{S_i})_{i\in[1..n]} \Rightarrow C_S.$$

However, we rather leave the second proof obligation as it is and replace the last proof obligation by

$$I \wedge (A_{S_i} \Rightarrow C_{S_i})_{i\in[1..n]} \Rightarrow C_S.$$

We then get a stronger rule. The reason is that since the assumptions may contain liveness properties, it will often be the case that the invariant only implies some of the component assumptions, say A_{S_1} and A_{S_n}, and that the remaining $n-2$ can be deduced from $I \wedge C_{S_1} \wedge C_{S_n}$. Thus, we end up with the rule below

$$\frac{\begin{array}{l} A_S \Rightarrow I(y{\downarrow_0}) \\ I(y{\downarrow_j}) \Rightarrow \langle A_{S_i}\rangle(y{\downarrow_j})_{i\in[1..n]} \\ I(y{\downarrow_j}) \wedge \langle C_{S_i}\rangle(x_i{\downarrow_j}, y_i{\downarrow_{j+1}})_{i\in[1..n]} \Rightarrow I(y{\downarrow_{j+1}}) \wedge \langle C_S\rangle(y{\downarrow_{j+1}}) \\ \forall k \in \mathsf{N} : I(y{\downarrow_k}) \Rightarrow I \\ I \wedge (A_{S_i} \Rightarrow C_{S_i})_{i\in[1..n]} \Rightarrow C_S \end{array}}{S \rightsquigarrow \otimes_{i=1}^{n} S_i}$$

Remember that in each premise the elements of z and y are universally quantified over infinite timed streams and that j is universally quantified over N. Moreover, S, $S_1, \ldots, S_n$ are time dependent specifications. For soundness and completeness results we refer to [Stø95].
Additionally, we also need an adaptation rule. Let S and S' be time dependent specifications of the same syntactic interface (i, o). If these specifications have identical assumptions, then the following rule is valid

$$\frac{A_S \wedge (\forall j \in \mathsf{N}_\infty : \forall r : \langle A_S\rangle(i{\downarrow_j} \frown r, o{\downarrow_j}) \Rightarrow \langle C_{S'}\rangle(i{\downarrow_j} \frown r, o{\downarrow_{j+1}})) \Rightarrow C_S}{S \rightsquigarrow S'}$$

Similar rules can be formulated for time independent and synchronous specifications. For example, in the synchronous case, it is enough to replace any occurrence of $\downarrow$ by $|$.
Given the two rules introduced above we may for example prove that

$$\mathrm{NET} \rightsquigarrow \mathrm{HWQ} \otimes \mathrm{INT} \otimes \mathrm{SWQ},$$

where NET is given in Figure 6.

$$\mathrm{NET} \overset{\mathsf{sy}}{\equiv} (i \in M^{\infty} \rhd o \in D^{\infty}) ::$$

ass :
$$\forall m \in M^{*} : m \sqsubseteq i \Rightarrow \#(\odot \copyright m) \leq \#(D_{+} \copyright m)$$

com :
$$D_{+} \copyright o = (D_{+} \copyright i)|_{\#(\odot \copyright i)}$$

Figure 6: Specification of NET

Since

$$\mathrm{NET} \rightsquigarrow [\mathrm{NET}], \qquad [\mathrm{HWQ}] \otimes [\mathrm{SWQ}] \rightsquigarrow \mathrm{HWQ} \otimes \mathrm{SWQ},$$

and $\rightsquigarrow$ is transitive and a congruence with respect to $\otimes$, it is enough to show that

$$[\mathrm{NET}] \rightsquigarrow [\mathrm{HWQ}] \otimes \mathrm{INT} \otimes [\mathrm{SWQ}].$$

By the adaptation rule we have that

$$\mathrm{INT}' \otimes [\mathrm{SWQ}]' \rightsquigarrow \mathrm{INT} \otimes [\mathrm{SWQ}],$$

where INT' denotes the result of strengthening the commitment of INT with the following conjuncts

$$\forall j \in \mathsf{N} : M \copyright (r{\downarrow_{(j+1)}}) \sqsubseteq M_{+} \copyright (x{\downarrow_{j}}) \wedge D_{+} \copyright (y{\downarrow_{(j+1)}}) \sqsubseteq D \copyright (s{\downarrow_{j}}),$$

and $[\mathrm{SWQ}]'$ denotes the result of strengthening the commitment of [SWQ] with the following conjunct

$$\forall j \in \mathsf{N} : s{\downarrow_{(j+1)}} \sqsubseteq (D \copyright r)|_{\#(\odot \copyright r{\downarrow_{j}})}.$$

Thus, the transitivity and congruence property enjoyed by $\rightsquigarrow$ imply it is enough to prove that

$$[\text{NET}] \rightsquigarrow [\text{HWQ}] \otimes \text{INT}' \otimes [\text{SWQ}]'.$$

The latter follows by the composition rule if I is chosen as below

$$\langle A_{[\text{HWQ}]} \rangle \wedge A_{\text{INT}} \wedge A_{[\text{SWQ}]}.$$

5 A Timed FIFO-Queue

In Section 3 we have already specified a FIFO queue consisting of three components, namely a hardware queue HWQ, a software queue SWQ, and an interface INT, where the latter was responsible for the communication between the two queues. The first of these was intended to be implemented in hardware. The other two were intended to be implemented in software.
Contrary to earlier, we now impose constraints on the response time of the FIFO queue. We first give an overall specification characterizing the requirements imposed on the whole network. Since the queue's external interface works in a synchronous manner, this specification is written in the synchronous format. We assume that

- The FIFO queue has a required delay of l time units in the sense that the reply is required to come exactly l time units after the request has been received.

We then decompose this overall specification in accordance with Figure 3. Clearly, the hardware queue should be specified in the synchronous format, and since we impose real-time constraints, both the software queue and the interface must be specified in the time dependent format. We assume that

- The hardware queue has a required delay of l time units in the same sense as for the overall queue.
- The interface has a delay of not more than one time unit in the direction from the hardware queue to the software queue. In the other direction it also forwards the messages as soon as it can. However, since the software queue may produce more than one message within a time unit, and the hardware queue expects synchronous input, the delay can be greater than one time unit.
- The interface and the software queue together have a maximal delay of $k + 2$ time units in the sense that any reply will be provided within this time range.

It is beyond the scope of this paper to characterize the exact relationship between the constants l and k. We just assume that k has been chosen in such a way that the specification HWQ is realizable in hardware.

5.1 Overall Specification

The overall requirements are specified in Figure 7. As in earlier specifications the assumption makes sure that no request is sent when the queue is empty. Remember that 0 is sent whenever no ordinary message can be sent within a time unit. With

$\mathrm{T\_NET} \overset{\mathsf{sv}}{\cong} (i \in M^{\infty} \rhd o \in D^{\infty}) ::$

ass :

$\forall m \in M^{*} : m \sqsubseteq i \Rightarrow \#(\odot \copyright m) \leq \#(D_{+} \copyright m)$

com :

$\forall j \in \mathsf{N}_{+} : \mathsf{let}\ n = \#(\odot \copyright (i|_{j}))\ \mathsf{in}\ i.j = \odot \Rightarrow o.(j+l) = (D_{+} \copyright i).n$

Figure 7: Overall Requirements

respect to the commitment, for a given j, n is the number of requests received until time j. This means that if $i.j$ is a request then $o.(j+l)$ is required to contain the n'th data element received. Note that for any j such that $i.j \neq \odot$, nothing is said about the component's behavior at time $j+l$ except that the message output at time $j+l$ is an element of D.

5.2 Specification of the Software Queue

The software queue is specified in the time dependent format (see Figure 8). The assumption imposes the usual constraint on requests via its second conjunct. The first conjunct makes sure that not more than one message is received per time unit. This assumption is okay since the communication on x is synchronous and INT forwards the messages it receives on x along r with a delay of exactly one time unit. The first conjunct of the commitment states that the software queue replies in a FIFO manner (if it replies at all). The second conjunct insists that for any j, the number of requests in r until time j is less than or equal to the number of replies in s until time $j+k$. This guarantees that any reply is received within k time units. Note that if $k > 1$ this constraint does not restrict the component from replying in less than k time units. This in contrast to the specification T_NET which requires the reply to come after exactly l time units.
Note also that the first conjunct of the assumption together with the commitment guarantee that if SWQ for example forwards two data elements within one time unit then the last of these is forwarded with a delay of less than k time units. Due to this fact we may prove that the software part of the network stays within the $k+2$ time units requirement.

$$\mathrm{SWQ} \stackrel{\mathsf{td}}{\equiv} (r \in {M_+}^{\overline{\infty}} \rhd s \in {D_+}^{\overline{\infty}}) ::$$

ass :

$$\forall j \in \mathsf{N} : \#r{\downarrow_{(j+1)}} \leq \#r{\downarrow_j} + 2 \wedge \forall m \in M^{\overline{*}} : m \sqsubseteq r \Rightarrow \#(\odot \copyright m) \leq \#(D \copyright m)$$

com :

$$D \copyright s \sqsubseteq D \copyright r \wedge \forall j \in \mathsf{N} : \#(\odot \copyright (r{\downarrow_j})) \leq \#(D \copyright (s{\downarrow_{(j+k)}}))$$

Figure 8: Specification of the Software Queue

5.3 Specification of the Interface

$$\mathrm{INT} \stackrel{\mathsf{td}}{\equiv} (x \in M^{\overline{\infty}}, s \in {D_+}^{\overline{\infty}} \rhd y \in D^{\overline{\infty}}, r \in {M_+}^{\overline{\infty}}) ::$$

ass :

$$\mathsf{sy}(x) \wedge \forall m \in M^{\overline{*}} : m \sqsubseteq x \Rightarrow \#(\odot \copyright m) \leq \#(D_+ \copyright m)$$

com :

$$\mathsf{sy}(y) \wedge D_+ \copyright y \sqsubseteq D \copyright s \wedge$$

$$\forall j \in \mathsf{N} : \#(D_+ \copyright (y{\downarrow_j})) < \#(D \copyright (s{\downarrow_j})) \Rightarrow y.(2j+1) \in D_+ \wedge$$

$$r = \langle \surd \rangle \frown (M_+ \cup \{\surd\}) \copyright x$$

Figure 9: Specification of the Interface

The interface INT is specified in the time dependent format (see Figure 9). Since the input along x comes from the hardware queue the communication history of x can be assumed to be synchronous. This explains the first conjunct of the assumption. Similarly, since the output along y is to be received by the hardware queue, it has to be synchronous. Thus, the first conjunct of the commitment is needed. The second conjunct of the commitment ensures that only data elements received on s is output along y.
The third conjunct of the commitment insists that the interface forwards the data elements it receives from the software queue as fast as it can without breaking the invariant imposed on y that not more than one data element is sent within the same time unit. The second and third conjunct together guarantee that any data element sent by SWQ eventually will reach HWQ.
Note that the third conjunct cannot be simplified to

$$\forall j \in \mathsf{N} : \#(D_+ \copyright (y{\downarrow_{(j+1)}})) = \#(D \copyright (s{\downarrow_j})).$$

The reason is that the software queue may send more than one data element between two consecutive ticks. Thus, because the communication along y is required to be synchronous, if the interface receives three data elements between two consecutive ticks on s, then it needs at least four time units (remember that there is at least a delay of one time unit) before all three data elements have been forwarded along y. The fourth conjunct of the commitment makes sure that the data elements and requests received on x are forwarded with a delay of exactly one time unit.

5.4 Specification of the Hardware Queue

$$\mathrm{HWQ} \overset{\mathsf{sy}}{\equiv} (i \in M^\infty, y \in D^\infty \rhd o \in D^\infty, x \in M^\infty) ::$$

ass :

$$\forall m \in M^* : m \sqsubseteq i \Rightarrow \#(\odot \copyright m) \leq \#(D_+ \copyright m) \wedge$$

$$D_+ \copyright y \sqsubseteq D_+ \copyright x \wedge \forall j \in \mathsf{N} : \#(\odot \copyright (x|_j)) \leq \#(D_+ \copyright (y|_{(j+k+2)}))$$

com :

$$\forall j \in \mathsf{N}_+ : \mathsf{let}\ n = \#(\odot \copyright (i|_j))\ \mathsf{in}\ i.j = \odot \Rightarrow o.(j+l) = (D_+ \copyright i).n \wedge$$

$$\forall m \in M^* : m \sqsubseteq x \Rightarrow \#(\odot \copyright m) \leq \#(D_+ \copyright m)$$

Figure 10: Specification of the Hardware Queue

The hardware queue is specified in the synchronous format (see Figure 10). The first conjunct of the assumption is similar to the previous cases. Clearly the hardware queue can only be required to behave correctly as long as the software part behaves as a FIFO queue. This is stated by the second and third conjunct of the assumption. The first conjunct of the commitment is equal to that of T_NET. In addition we must make sure that the hardware queue uses the software queue correctly, i.e., it should only send requests when the software queue is not empty. This requirement is stated by the second conjunct of the commitment.

5.5 Verification

The correctness of this decomposition can be verified in the same way as for the earlier example. Firstly, translate all specifications into the time dependent format. Secondly, use the adaptation rule to make implicit causalities explicit. Thirdly, use the parallel rule to prove the decomposition step. We leave the details as an exercise for the reader.

6 Conclusions

This paper has introduced a formal method supporting hardware/software co-design. We have presented a uniform, common semantic basis for all specification formats, composition and refinement. Our method is compositional and well-suited for top-down design. We can handle both synchronous and asynchronous communication. Real-time constraints can be specified, and we have specially designed rules for the verification of refinement steps. This allows hardware/software co-design to be conducted in a very elegant way.

When we claim that our method supports hardware/software co-design, we obviously consider only those aspects related to specification and verification. There are of course many other interesting research directions inside the area of hardware/software co-design that we have not considered here.

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 corresponds 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]. For more details about the semantic model we refer to [GS95].

Specification Formats: The formats for time dependent and time independent specifications are inspired by [BS94]. The format for synchronous specifications is for example related to the approach in [Tuc92].

The one-step-longer-than semantics used by us is strongly inspired by [AL93]. [Col94] 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]. The composition rule introduced above does not require such restrictions. The main reason, as we see it, is that the rule makes a clear distinction between induction hypotheses and assumptions, i.e., it does not employ the usual trick of using the environment assumptions as induction hypotheses.

An assumption/commitment rule handling general liveness properties in the assumptions can also be found in [Pnu85] (related rules are proposed in [Sta85], [Pan90]). However, this rule is based on a $\Rightarrow$ semantics for assumption/commitment specifications. Our rules 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, with respect to our approach, we have at least shown that, from a technical point of view, there is no reason why such constraint should be imposed.
Soundness and completeness proofs for our rules can be found in [Stø95].

7 Acknowledgments

Financial support has been received from the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen". Manfred Broy, Pierre Collette and Christian Facchi have read a draft of this paper and provided many helpful comments.

References

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

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

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

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

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

[BS94] M. Broy and K. Stølen. Specification and refinement of finite dataflow networks — a relational approach. In *Proc. FTRTFT'94, Lecture Notes in Computer Science 863*, pages 247–267, 1994.

[Buc94] K. Buchenrieder, editor. *Third International Workshop on Hardware/Software Codesign.* IEEE Computer Society Press, 1994.

[Col94] P. Collette. An explanatory presentation of composition rules for assumption-commitment specifications. *Information Processing Letters*, 50:31–35, 1994.

[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.

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

[Hoa72] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–282, 1972.

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

[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.

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

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

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

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

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

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

[Stø95] K. Stølen. Assumption/commitment rules for data-flow networks — with an emphasis on completeness. Technical Report SFB 342/09/95 A, Technische Universität München, 1995.

[Tuc92] J. Tucker. Equational specification of synchronous concurrent algorithms and architectures. In *Proc. Programming and Mathematical Method, Summerschool, Marktoberdorf*. Springer, 1992.

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

Reihe A

342/22/90 A Manfred Broy, Claus Dendorfer: Functional Modelling of Operating System Structures by Timed Higher Order Stream Processing Functions

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

Reihe A

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

Reihe A

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

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

Reihe A

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

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 Gauš Elimination

342/13/93 A Christoph Pflaum, Ulrich Rüde: Gauš' 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

Reihe A

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

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 Emphasize on Completeness

342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design

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