A compositional proof system for an occam-like real-time language

J. Hooman

87/14

december 1987
A Compositional Proof-System for an OCCAM-like Real-Time Language

by

Jozef Hooman

CSN 87/14

November 1987
Computing Science Notes

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science of the Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.
A COMPOSITIONAL PROOF SYSTEM FOR AN OCCAM-LIKE REAL-TIME LANGUAGE

Jozef Hooman *

Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. box 513
5600 MB Eindhoven
The Netherlands

November 1987

ABSTRACT

A compositional proof system is given for an Occam-like real-time programming language to specify and verify distributed synchronous message passing. Concurrency is modelled as "maximal parallelism"; that is, if a process can proceed it will do so immediately. A process only waits when no local action is possible and no partner is available for communication. Terminating and nonterminating processes can be specified from the viewpoint of an external observer with his own clock. This leads to a global notion of time. Furthermore we take a dense time domain.

A specification of a process consists of a Hoare triple (pre and post condition) extended with two invariants: an assumption about expected behaviour of the environment, and a commitment which is guaranteed by the process itself, as long as the environment satisfies the assumption. In deviation of earlier work [H], assumption and commitment may refer directly to the global time. This makes it possible to specify (and verify) that something must happen at a certain point of time.

In the proof system emphasis is put on an easy way of reasoning at parallel composition. The parallel composition rule deals with checking and removing assumptions only. Maximal parallelism can be used locally by making suitable assumptions and applying a separate "strengthen" rule which models the assumption/commitment reasoning.

Electronic-mail address: mcvax!eutrc3!wsinjh.UUCP (or wsdcj@heidhe5.BITNET).
0. NOTATIONS

This section contains a number of notations used in subsequent chapters. Let \( \text{TIME} \) be a given (nonempty) time domain with addition, subtraction, equality \( = \), and an ordering \( \leq \). Intervals of this domain are denoted by
\[
< l, u > = \{ t \in \text{TIME} \mid l < t < u \} \text{ for an open interval,}
\]
\[
[l, u ] = \{ t \in \text{TIME} \mid l \leq t \leq u \} \text{ for a closed interval,}
\]
\[
[l, u > \text{ and } < l, u ] \text{ for half-open half-closed intervals.}
\]

Let \( \text{SET} \) be a collection of sets.
Consider functions which map a point of \( \text{TIME} \) to an element of \( \text{SET} \), i.e. to some set.

For two functions \( f, f' : \text{TIME} \rightarrow \text{SET} \) we define
\[
* \text{ the pointwise union of } f \text{ and } f', \text{ notation } f \cup f', \text{ as}
\]
\[
(f \cup f')(t) = f(t) \cup f'(t), \text{ for all } t \in \text{TIME}.
\]
\[
* \text{ the pointwise subtraction of a set } \text{set} \in \text{SET} \text{ from a function } f, \text{ notation } f \ominus \text{set}, \text{ as}
\]
\[
(f \ominus \text{set})(t) = f(t) - \text{set}, \text{ for all } t \in \text{TIME}.
\]

The restriction \( f \downarrow \alpha \), of a function \( f \) to time \( \alpha \in \text{TIME} \), is defined as follows:
\[
(f \downarrow \alpha)(t) =
\begin{cases}
  f(t) & \text{if } t \leq \alpha \\
  \emptyset & \text{if } t > \alpha
\end{cases}
\]

Let \( \inf \) be the greatest lowerbound and \( \sup \) the smallest upperbound of a subset of \( \text{TIME} \).
For convenience we denote \( \inf \text{TIME} \) by \( -\infty \) and \( \sup \text{TIME} \) by \( \infty \).

Note: \( \inf \emptyset = \infty \), and \( \sup \emptyset = -\infty \).

Definition of the minimum and maximum of a function \( f \):
\[
\min(f) = \inf \{ t \mid f(t) \neq \emptyset \}, \text{ and } \max(f) = \sup \{ t \mid f(t) \neq \emptyset \}.
\]

The image of a set of time points (e.g. an interval) is defined as the pointwise union:
for \( T \subseteq \text{TIME} \), \( f(T) = \bigcup_{t \in T} f(t) \).

The inverse image of a set \( \text{set} \in \text{SET} \) under a function \( f \), notation \( f^{-1}(\text{set}) \), is defined as
\[
f^{-1}(\text{set}) = \{ t \in \text{TIME} \mid \text{set} \subseteq f(t) \}.
\]
$T \mapsto \text{set}$ denotes the function which assigns $\text{set} \in \text{SET}$ to all points in $T$,
and $\emptyset$ to all points outside $T$: $(T \mapsto \text{set})(t) = \begin{cases} \text{set} & \text{if } t \in T \\ \emptyset & \text{if } t \notin T \end{cases}$

The symbol for the empty set $\emptyset$ is also used to denote the function which assigns the empty set to all points of time.

We often use: $\forall x, P(x) : p$ as an abbreviation for $\forall x : P(x) \rightarrow p$.

$p[y/x]$ denotes the substitution of $y$ for all free occurrences of $x$ in $p$.

For a set $A$, $\mathcal{P}(A)$ denotes the powerset of $A$, i.e. the set of all subsets of $A$.

1. INTRODUCTION

In the area of real-time systems (such as process control, telecommunication, life support systems in hospitals and avionics systems) there is a growing need for formal specification and verification techniques. Concurrency and hard time limits make the design and development of real-time embedded systems very complex, and certainly testing is not sufficient to validate a program. Also, in many real-time applications failure is very expensive and can have disastrous consequences. As a result of our work in Esprit project DESCARTES, which provides a context for investigating the theoretical background of real-time systems, this paper contains a proof system for the specification and verification of real-time embedded systems.

A simple language akin to Occam ([Occ]) is considered to capture the essential features of real-time in the context of distributed synchronous message passing. Communication occurs along unidirectional channels between pairs of processes. Nesting of parallel composition is allowed, and there is a hiding operator to hide internal communications. By the real-time statement $\text{DELAY } d$ the execution is suspended for the specified number of time units. Such a $\text{DELAY}$-statement may occur in the guard of an alternative command. Together with the underlying execution model this gives the opportunity to program a time-out. The execution model is that of "maximal parallelism". That is, if a process can proceed it will do so immediately. A process only waits when no local action is possible and no partner is available for communication. As soon as an action becomes possible execution must proceed.

For this language we formulate a proof theory which meets the following aims:
- specify communication and timing behaviour of terminating and nonterminating processes from the viewpoint of an external observer with his own clock,
- specify processes which have an intensive interaction with their environment, and where communications with the environment have a great influence on the behaviour of these processes (so called reactive processes [Pn]).
verification during the process of program development (as opposed to a-posteriori verification) should be possible. That is, during the design of a program we want to verify the design steps.

How to achieve the requirements above? Consider the first point. In order to deal with real-time, we want to express the timing behaviour of a system from the viewpoint of an external observer with his own clock. Therefore a special variable \textit{time} is introduced which refers to this external clock. So at the level of reasoning there is a conceptual global clock. Furthermore, we use a dense time domain; between every two points of time there exists a third point of time. Having dense time is important during the process of development and refinement. With a discrete time domain the notion of smallest time unit has to be redefined in general. For instance, if we take the parallel composition of two processes with different time units, a common time unit has to be found and the specifications of the components have to be adapted. Note that two unrelated communications may be happening arbitrarily close to each other in time.

Furthermore we may want to refine one action which is considered to be atomic on one level, into several actions on lower level. Then it is inconvenient to fix a certain indivisible time unit for the top-level specification. For instance, in our framework a communication takes a certain period of time and, at the level of abstraction considered in this paper, one can only see that a communication is being performed during this period of time. A closer look, however, may refine this uniform interval into different events happening, e.g. according to some handshake protocol.

Next we discuss the form of the specifications. Our formalism is based on Hoare triples, i.e. \{pre\} "program" \{post\}. Meaning: if we start the "program" in a state satisfying assertion \textit{pre} and if the program terminates then assertion \textit{post} holds for the termination state.

Using the special variable \textit{time}, which refers to the global notion of time, the timing behaviour of a program can be specified:
\{time = 3\} "program" \{7 ≤ time < 12\}.

For a proof of such statements we have to know the bounds on the execution time of atomic statements and the overhead associated with composite constructs.

With pre and post conditions only terminating processes can be specified. A usual approach to deal with nonterminating processes is to add an invariant, called \textit{commitment} in our formalism, to specify communication and timing behaviour of nonterminating processes. This commitment should hold at all points of time during and after the execution of a process, and it represents the real-time communication behaviour of that process. So the commitment must not refer to any internal state of the process during execution.

Still the framework explained so far is not satisfactory. In the specification of a process there is no information about the behaviour of its environment, whereas, in general, the behaviour of a process depends heavily on its environment. Especially when specifying reactive processes (see the second requirement above) we want to specify a process relative to knowledge about its environment.
Consider, as a simple example, the statement \( D?x \): synchronous input via channel \( D \) where the value received is stored in variable \( x \). \( D!e \) denotes synchronous output of the value of \( e \) via \( D \).

In the specification \( \{ \text{time} = 5 \} D?x \{ \cdots \} \) the values of \( \text{time} \) and \( x \) in the post condition depend on the behaviour of the environment: when is the environment enabled to do a \( D \)-communication, and which value will be sent.

In general, knowledge about the environment is an important factor in the design of a real-time process. Therefore, in the tradition of [MC] and [ZRE84], we add an assumption to the specification by which we can strengthen post condition and commitment.

Our formulae, called correctness formulae, are of the form \( (A, C) : \{p\} L \{q\} \),

where \( L \) is a program in our programming language,

\( A \) is an assumption describing the expected behaviour of the environment of \( L \), and

\( C \) is a commitment which is guaranteed by process \( L \) itself, as long as the environment does not violate the assumption.

When two processes are composed in parallel, we have to verify that assumptions of one process about joint communications are justified by commitments of the other process for these joint communications.

What should be in the assertion language; which expressions can be used in \( A, C, p \) and \( q \)?

The special variable \( \text{time} \) has been mentioned already, and program variables can be used in the pre and post conditions. Since we are interested in the communication behaviour of processes there is a denotation for the communication actions being performed at a particular point of time. Furthermore remember that the maximal parallelism constraint imposes certain restrictions on the waiting for a communication. In order to use this constraint in a compositional way, and derive certain timing properties from it, it is intuitively clear that we need some denotation for this waiting. Formally the need for such a denotation follows from the full abstraction result of [HGR]: if termination, communication along channels and progress of time are the observables of a process then it is not possible to characterise real-time distributed message passing in a compositional way without some denotation for this waiting. Observe that for a process and its environment the waiting period for a joint communication may be different. Hence a distinction is made between the waiting for input and the waiting for output. Finally, in commitments the special variable \( \text{fin} \) can be used to denote termination of a process.

For a general impression of our specifications we give some simple examples. Since the full assertion language has not been given up to now we write informal sentences in assumption and commitment. In chapter 3, however, these sentences will be expressed formally, and the examples below can be verified with the proof system of chapter 4. It is assumed here that a communication (without waiting) takes one time unit.

In the specification of a process there are two important assumptions which can be made about the behaviour of its environment: the values sent by this environment, and when the environment is ready to do a communication. Given such assumptions the timing and communication behaviour of a process can be determined.
Examples

- Make assumptions about the values sent by the environment:

  \[(\text{env sends 3 via D, true }) : \{\text{true} \} D?x \ {x = 3}\].

  Use this assumption for a commitment about the next communication:

  \[(\text{env sends 3 via D, send 4 via B }) : \{\text{true} \} D?x \ ; B!x + 1 \ {x = 3}\].

- Make assumptions about the waiting of the environment for communications.

  For instance, when the environment is ready to start a communication. Then we can determine when the communication must take place, and determine the termination time in the post condition.

  \[(\text{env waits for D! from 5 \ no D comm from 3 till 5} ,
  \text{wait for D? from 3 \ (time } \geq 6 \rightarrow \ D \text{ comm from 5 till 6 }) : \{\text{time = 3} \} D?x \ {time = 6}\).

  \[(\text{env waits for D! from 5, wait for D? from 5 \ (time } \geq 8 \rightarrow \ D \text{ comm from 7 till 8} ) :
  \{\text{time = 7} \ \no D \text{ comm from 5} \} D?x \ {time = 8}\).

  In the formulae above "\text{env waits for D! from 5}" means:

  wait for output via D until the actual communication takes place.

  These assumptions can be used to commit something about the next communication, e.g.:

  \[(\text{env waits for D! from 5 \ no D comm from 3 till 5} ,
  \text{wait for B? from 6} ) : \{\text{time = 3} \} D?x \ ; B?y \ {true}\).

  Due to a time-out the environment can restrict the waiting period. This is reflected in assumptions such as: "\text{env waits for D! from 5 at most until 8}".

\[\square\]

Observe that in our proof system for safety properties of real-time programs we can express that a program must do something at a certain point of time. The examples above demonstrate that the commitment can express that, given a suitable assumption, the program must communicate. Without making assumptions it is possible to specify that a process must start waiting for a communication at a certain point of time: \[(\text{true} , \text{wait for D? from 2} ) : \{\text{time = 2}\} L \ {\cdots}\].

The last requirement, verify-while-develop, imposes some constraints on the proof system, in which rules and axioms are given to relate specifications and programs. A proof system, which is suitable for integration in the design process, should be compositional, that is: each composite program construct has a rule in which a specification of the construct can be derived from specifications of its constituents, without any further knowledge about the structure of these constituents. As a consequence every component can be developed in isolation according to its specification.

--- 5 ---
Important in our compositional proof system is the rule for parallel composition of two processes. In this rule assumptions about shared channels of the two processes are verified and then removed. Assumptions about external channels are maintained in the new assumption of the network. In principle the conjunction of commitments, and the conjunction of post conditions is taken, except for some renaming to deal with different termination times. In our proof system it is not necessary to add the maximal parallelism constraint globally at parallel composition. Maximal parallelism can be used locally for input and output commands, because knowledge about the waiting of the environment can be expressed in the assumption. By a separate "strengthen rule" this assumption can be used together with maximal parallelism to derive a stronger commitment. There is, however, no obligation to make assumptions and use maximality locally. It is possible to restrict the waiting of processes after parallel composition by using the consequence rule, which also includes maximal parallelism.

This paper is organized as follows. Chapter 2 contains the syntax of the real-time language and its intuitive semantics. In chapter 3 we explain the interpretation of our correctness formulae and the assertion language. The compositional proof system is formulated in chapter 4. The conclusion and a discussion of future work can be found in chapter 5. The appendices contain a lot of technical details: in appendix A a formal denotational semantics is given, a formal interpretation of correctness formulae can be found in appendix B. In appendix C soundness of the proof system from chapter 4 w.r.t. the semantics given in appendix A is proved. Appendix D contains some details about the semantics of the iteration construct, namely that the fixed point equation, given in appendix A, has a unique solution. References can be found in appendix E.

ACKNOWLEDGEMENTS

I like to thank Willem Paul de Roever for introducing me into the theory of proof systems in general, and into the work of Job Zwiers in particular, which to a large extend influenced my own work. Furthermore the members of ADCAD and the EUT involved in ESPRIT project Descartes are acknowledged for clarifying discussions. Especially the remarks of Rob Gerth and Amir Pnueli have led to a number of improvements.
2. SYNTAX

In this chapter we give the syntax of a real-time programming language for distributed synchronous message-passing. Communication takes place through unidirectional channels which connect exactly two processes. There is a delay-statement, which may appear in the guard of an alternative statement, too. Such a delay-branch makes it possible to program a time-out, i.e. to restrict the waiting period for certain communications. We separate the concepts of parallel composition and hiding of internal communications by introducing an explicit hiding operator [..].

In the syntax below \( D \) will stand for a channel name, \( d \) and \( e \) for expressions, \( b \) for a boolean expression, and \( x \) for a program variable.

**Language construction**

\[
L ::= S \mid N
\]

**Statement**

\[
S ::= \text{SKIP} \mid x := e \mid IO \mid \text{DELAY} d \mid S_1; S_2 \mid [N] \mid G \mid *G
\]

**Guarded command**

\[
G ::= \left[ \bigotimes_{i=1}^{n_1} b_i \rightarrow S_i \bigotimes_{i=1}^{n_2} b_i' ; IO_i \rightarrow S_i' \bigotimes_{i=1}^{n_3} b_i'' ; \text{DELAY} d_i \rightarrow S_i''' \right]
\]

**Input/Output**

\[
IO ::= D!e \mid D?x
\]

**Network**

\[
N ::= S_1 \parallel S_2
\]

A boolean expression \( b_i \) or \( b_i'' \) is omitted if it is \( TRUE \).

2.1 Informal semantics

**Atomic commands**

- **SKIP**
  skip: the only effect of this statement is that it takes some time to execute it.

- **\( x := e \)**
  assignment: the value of expression \( e \) is assigned to the variable \( x \), and there is some progress of time.

- **\( D!e \)**
  output: send the value of expression \( e \) through channel \( D \);
  first a waiting period for a corresponding input command, and
  when a partner is available a (synchronous) communication takes place, which takes some time.

- **\( D?x \)**
  input: receive a value via channel \( D \) and assign this value to the variable \( x \);
  first a waiting period for a corresponding output command, and
  when a partner is available a (synchronous) communication takes place, which takes some time.

- **\( \text{DELAY} d \)**
  delay: suspends the execution for \( (\text{the value of}) \ d \) time units. A delay statement with a negative value is equivalent to a delay statement with a zero value.
**Composite commands**

- \( S_1; S_2 \) sequential composition: execute \( S_2 \) after having executed \( S_1 \).
- \([N]\) hiding: the internal channels of network \( N \) are no longer visible.
- \( G \) guarded command: A guard is open if the boolean part evaluates to true.

Following [KSRGA] we give priority to purely boolean guards. So if at least one of the \( b_i \) is true then select non-deterministically one of the open purely boolean guards and execute the corresponding branch. If none of the purely boolean guards is open and none of the other guards is open execution aborts. Otherwise, let \( \text{mindelay} \) be the minimum of the delay-values of the open delay-guards (infinite if there are no open delay-guards). If within \( \text{mindelay} \) time units at least one IO-command of the open IO-guards can be executed, select non-deterministically one of them and execute the guard and the corresponding branch. Otherwise, if no IO-guard can be taken within \( \text{mindelay} \) time units, one of the open delay-guards with delay value equal to \( \text{mindelay} \) is selected.

- \(*G\) iteration: repeated execution of guarded command \( G \) as long as at least one of the guards is open. When none of the guards is open execution terminates.
- \( S_1 \parallel S_2 \) network: parallel execution of \( S_1 \) and \( S_2 \), based on the maximal parallelism model; no process ever waits unnecessarily, if execution can proceed it will do so immediately.

We assume given a lower bound and an upper bound on the execution time of atomic constructs, and bounds on the overhead needed for composite constructs. Furthermore, it is assumed that communications take a positive amount of time, and that there exists an \( \epsilon > 0 \) such that the overhead associated with the guarded command is at least \( \epsilon \).

### 2.2 Example

The following program \( P \) illustrates syntax and informal semantics:

\[
P \equiv \ast[H?m \rightarrow \text{counter} := 0; \ast[\text{counter} < 3; D!m \rightarrow [A?x \rightarrow \text{counter} := 4
\]

\[
\begin{align*}
\square & \text{DELAY 30} \rightarrow \text{counter} := \text{counter} + 1 \\
\square & \text{counter} = 3 \quad \rightarrow E!0; \text{counter} := 4 
\end{align*}
\]

Program \( P \) consists of an outer iteration of receiving a message \( m \) via channel \( H \) from a host. This message \( m \) is sent via channel \( D \), and \( P \) starts waiting for an acknowledgement via channel \( A \). This waiting for \( A \) is bound by a time-out: the guard "DELAY 30" is taken if no \( A \) communication is possible within 30 time units. Then \( \text{counter} \) is updated and the inner iteration is executed again. When \( \text{counter} = 3 \) the second branch is taken and some error signal is sent to the host via channel \( E \). Note that when an \( A \) or \( E \) communication has been performed, \( \text{counter} \) is set to 4 which causes termination of the inner iteration (because then there are no open guards), and \( P \) is ready for a next message via channel \( H \).
2.3 Syntactic restrictions

First some definitions, let $\text{CHAN}$ be set of channel names. $D!$ and $D?$ are called directed channels, for $D \in \text{CHAN}$.

$\text{var}(L)$ denotes the program variables occurring in language construction $L$, $\text{chan}(L)$ denotes the set of visible channel names and directed channels in $L$, and $\text{type}(\text{IO})$ denotes the directed channel of an IO-command;

$\text{type}(D!e) = D!$ and $\text{type}(D?x) = D?$. 

\[ \text{ex. } \text{chan} (E!5; D?x \parallel D!2; F?3) = \{ D,D!,D?,E,E!,E,F,F? \}, \text{ and} \]

\[ \text{chan} ([E!5; D?x \parallel D!2; F?3]) = \{ E,E!,E,F,F? \}. \]

In a network $S_1 \parallel S_2$ the concurrent processes $S_1$ and $S_2$ are not allowed to have shared variables. Thus $\text{var}(S_1) \cap \text{var}(S_2) = \emptyset$.

Furthermore it is required that $S_1$ and $S_2$ do not have joint input channels or joint output channels. Thus $\text{chan}(S_1) \cap \text{chan}(S_2) \subseteq \text{CHAN}$.

Note that the joint channels of $S_1 \parallel S_2$, i.e. $\text{chan}(S_1) \cap \text{chan}(S_2)$, are exactly those channels through which $S_1$ and $S_2$ may communicate with each other.

Throughout this paper we use $\equiv$ to denote syntactic equality.

3. SPECIFICATION LANGUAGE

In this chapter the specification language is defined. In section 3.1 the correctness formulae for specifying and verifying programs are introduced. A number of expressions used in the assertion language are listed in section 3.2. Restrictions on assertions are formulated in section 3.3. More details can be found in appendix B which contains the formal interpretation of assertions and correctness formulae.

3.1 Correctness formulae

In this section the correctness formulae used to specify real-time processes are introduced. These formulae should be suitable to specify timing and communication behaviour of terminating and nonterminating processes. Furthermore it should be possible to specify the behaviour of a process relative to assumptions about the behaviour of its environment. Therefore Hoare triples are extended with an invariant, which is split up in two parts, an

- assumption specifying the expected behaviour of the environment, and a
- commitment, which is guaranteed to hold by the process itself, as long as the assumption concerning earlier behaviour has not been violated by the environment.
Important is that assumption and commitment reflect the externally visible behaviour of environment and process, resp. That is, they refer to (the timing of) termination, communications along externally visible channels and waiting concerning these channels. Consequently, assumption and commitment must not contain program variables or internal channels.

We use the notation: \((A, C) : \{p\} L \{q\}\), where \(A\) is the assumption, \(C\) the commitment, \(p\) the pre condition, and \(q\) the post condition, with the following meaning:

there exists a constant \(\delta > 0\) such that

if \(p\) holds for the initial state in which \(L\) starts executing, then

for all points of time \(t\) during or after execution of \(L\):

(1) \(C\) holds at \(t\) (during or after execution of \(L\)), and

(2) if \(L\) terminates at \(t\) then \(q\) holds for the termination state.

The use of \(A\) is restricted to all points of time up to a \(\delta\) distance (in time) from the point where \(C\) has to hold. A motivation for this interpretation can be found in the conclusion (chapter 5).

3.2 Assertion language

In this section we list a number of expressions which can be used in the assertions of a correctness formula. In order to determine what is needed in this assertion language, we first list what we want to specify and verify about a real-time process with communication via synchronous message passing:

- the values of program variables at the start of the execution,
- whether the program terminates or not, and if the program terminates:
  - when does it terminate, and what are the values of program variables at termination,
  - the communications which are performed, when they are performed, and the values transmitted. So it should be possible to specify at every point of time which communications are being performed.
- the waiting for a communication via a particular channel at a certain point of time.

Some denotation of the waiting for a communication is needed, because for a compositional treatment it should be possible to combine the specifications of two processes and derive a specification for their parallel composition without knowing the structure of these processes. The timing behaviour of this parallel composition depends on the maximal parallelism constraint, i.e. there is no unnecessary waiting. Especially: two processes never both wait for a communication via the same channel. So this constraint imposes restrictions on the waiting for a communication. The full abstraction result of [HGR] for a similar language implies that indeed some denotation for this waiting is required in order to achieve compositionality.
Reasoning about timing properties of programs requires a way to refer to the time in the assertions. We adopt a global notion of time, that is, at the level of reasoning there is a conceptual global clock. Because two arbitrary communications may be arbitrarily close to each other in time, we take a dense time domain \(\text{TIME}\).

In this paper we choose \(\text{TIME}\) equal to the rational numbers, \(\text{TIME} = \mathbb{Q}\), with the usual addition operation \(+\), ordering \(<\) and equality \(=\). For simplicity (to avoid an elaborate distinction), let the domain \(\text{VAL}\) for values of identifiers be such that \(\text{VAL} = \text{TIME}\).

3.2.1 Assertions

Given the list of observables above, it is possible in assertions to refer to the

- **communication function**, i.e. a function \(\pi\) from the time domain \(\text{TIME}\) to sets of communication records \((D,v)\), with \(D\in\text{CHAN}\) and \(v\in\text{VAL}\).

\((D,v)\in\pi(t)\), for \(t\in\text{TIME}\), denotes that at time \(t\) a communication via channel \(D\) with value \(v\) is being performed.

Notation: \(\pi\).

**example**

For \(t\in\text{TIME}\),

\[\pi(t) = \{(E,4),(D,7)\}\]

denotes that at time \(t\) two communications are being performed.

\[\pi(t) = \emptyset\]

denotes that there is no communication at time \(t\).

Note that because of the syntactic restrictions on channels (i.e. they connect exactly two processes) at most one \(D\)-communication can happen at any point of time.

- **wait function**, i.e. a function \(W\) from the time domain to sets of directed channels, \(W: \text{TIME} \rightarrow \mathcal{P}(\{D!,D? \mid D\in\text{CHAN}\})\). At every point of time it denotes the waiting for a communication via a directed channel in the associated set.

Notation: \(W\).

**example**

An output command \(D!e\) starting at time 5 first has to wait for a corresponding partner. A waiting period of \(v\) time units is represented by a wait function \(<5,5+v>\rightarrow\{D!\}\).

Thereafter the communication takes place, denoted by a communication function (assume the communication takes \(t\) time units): \(<5+v,5+v+t>\rightarrow\{(D,e)\}\).

- **program variables**

- **time**

Notation: the special variable \(\text{time}\).

- **termination** of a process. Only in commitments a special boolean variable \(\text{fin}\) can be used which is true iff the process has terminated.

Notation: the special variable \(\text{fin}\).

The special variable \(\text{fin}\) is used in commitments in order to be able to write a specification which distinguishes between programs with different behaviour in sequential composition. For instance, without \(\text{fin}\) we can not give a specification which distinguishes the programs.
S_{11} \equiv [D!0 \square D!0 \rightarrow *[TRUE \rightarrow SKIP]] \text{ and } S_{12} = D!0

(both programs do not perform any communication or waiting after the D communication).

A compositional rule for sequential composition requires a distinction, however, since we can already (without \texttt{fin}) give a distinguishing specification for $S_{11};D!1$ and $S_{12};D!1$ (viz. the first composition can do nothing after the first D communication, whereas the second composition always starts waiting for the second D communication).

In assertions we use \textit{logical variables} to relate assumption, commitment, precondition and postcondition. These variables do not occur in the program text, so the value they denote is not affected by program execution. In order to apply correct substitutions distinguish between four types of logical variables:

- logical communication variables : $c$, denoting a communication function,
- logical wait variables : $w$, denoting a wait function,
- logical \textit{VAL} (or \textit{TIME}) variables : $v$ or $t$, denoting a value from \textit{VAL} (=TIME),
- logical boolean variables : $b$, denoting \textit{true} or \textit{false}.

Quantification is allowed over logical variables only.

Communication functions must occur \textit{projected}, that is, within the scope of a projection \([...]_D\), for $D \in \text{CHAN}$.

Let $ce$ be a communication expression, i.e. an expression denoting a communication function, then $[ce]_D$ denotes the communication function which is the restriction of $ce$ to communication records with channel name $D$, so for $t \in \text{TIME}$: $[ce]_D(t) = \{(D,e) | (D,e) \in ce(t)\}$.

Similar wait functions must occur projected on a directed channel:

$[we]_dc$ denotes the restriction a wait expression $we$ to a directed channel $dc$ ($dc \equiv D!$ or $dc \equiv D?$ for $D \in \text{CHAN}$): $[we]_dc(t) = \{dc \in dc \epsilon we(t)\}$, for all $t \in \text{TIME}$.

### 3.2.2 Abbreviations

The following abbreviations are often used:

$\pi_D \equiv [\pi]_D$, $W_D \equiv [W]_D$, $W_D' \equiv [W]'_D$.

Furthermore we can project on more than one (directed) channel, e.g. $\pi_{DE} \equiv \pi_D \cup \pi_E$.

The \textit{restriction} of an assertion $p$ to a time $t \in \text{TIME}$ is defined as the following substitution:

$p \upharpoonright t \equiv p[\pi_D^{\uparrow} / \pi, \pi_W^{\uparrow} / \pi_W, \epsilon \uparrow \upharpoonright \epsilon]$.

The inverse of a function has been defined already in chapter 0, we use also:

$\pi_D^{-1} \equiv \pi_D^{-1}((D)) \equiv \{t \in \text{TIME} | \pi_D(t) \neq \emptyset\}$, and

$W_D^{-1} \equiv W_D^{-1}((D!)) \equiv \{t \in \text{TIME} | W_D(t) \neq \emptyset\}$.

Below we give a formal expression of assertions used in examples of chapter 1 and chapter 4. Note that there is no distinction between an expression denoting the behaviour of a process and an expression denoting the behaviour of an environment, the only difference is their place in the correctness formula. So in the expressions below \textit{wait} may be replaced by \textit{env waits}, and \textit{send} by \textit{env sends}.
\[ \text{time first } D \text{ after } t \equiv \inf \{ v > t \mid \pi_D(v) \neq \emptyset \} \]

\[ \text{send } v \text{ via } D \equiv \pi_D(TIME) \subseteq \{(D,v)\} \]

\[ \text{wait for } D \text{ from } t \equiv <t, \min(\text{time first } D \text{ after } t, \text{time})] \subseteq W_{\bar{D},1}^{-1} \]

\[ \text{wait for } D \text{ from } t_1 \text{ at most until } t_2 \equiv <t_1, \min(\text{time first } D \text{ after } t_1, \text{time }, t_2)] \subseteq W_{\bar{D},1}^{-1} \]

\[ D \text{ comm from } t_1 \text{ till } t_2 \equiv <t_1,t_2>] \subseteq \pi_D^{-1} \]

\[ \text{no } D \text{ comm from } t \equiv \pi_D(<t,\infty>) = \emptyset \]

\[ \text{no } D \text{ comm from } t_1 \text{ till } t_2 \equiv \pi_D(<t_1,t_2]) = \emptyset \]

### 3.3 Restrictions on the assertion language

Let \( \text{var}(p) \) be the set of program variables occurring in assertion \( p \).

\( \text{uchan}(p) \) is defined as the set of all undirected channels occurring in projections of \( \pi \) in \( p \).

\( \text{dchan}(p) \) is defined as the set of all directed channels occurring in projections of \( W \) in \( p \).

Furthermore, define \( \text{chan}(p) = \text{uchan}(p) \cup \text{dchan}(p) \).

\[ \text{ex. chan}(F!\#W_B(0) \land (E,1)!\#\pi_D(5)) = \{D,B!\} \]

For a correctness formula \( (A,C):\{p\} L \{q\} \) the following restrictions are imposed upon the assertions \( A, C, p \) and \( q \):

- \( \text{var}(A,C) = \emptyset \); program variables must not occur in \( A \) and \( C \), since \( A \) and \( C \) should express the externally observable behaviour only.

- \( \pi \) and \( W \) must occur projected in \( A, C, p \) and \( q \).

- the special boolean variable \( \text{fin} \) is allowed in \( C \) only.
4. PROOF SYSTEM

In this chapter we give a proof system, that is, rules and axioms to relate programs (according to the syntax in chapter 2) and specifications (as described in chapter 3). The proof system will be compositional: each composite program construct has a rule in which a specification of the construct can be derived from specifications of its constituents, without any further knowledge about the structure of these constituents (see [HdeR] for a survey of compositionality in proof systems). In the formulation below we assume that we know the channels and variables which occur (syntactically) in the components. This information could be added to the specifications easily, for instance by using a "basis" (see [PJ]).

In order to prove timing properties of programs, we need some knowledge about the execution time of atomic constructs, and about the overhead of composite constructs. In appendix A we give the semantics of our real-time language using a function $T$ which assigns to atomic statements the bounds on their execution time, and which gives for composite commands the bounds on the overhead for these commands. Instead of giving a proof system for such a general function, we take one specific function and formulate a proof system for it. Soundness of the rules, however, will not depend on this specific function. Furthermore it is easy to modify the system for a general $T$-function; see the skip-statement for an example of such a modification.

In the proof system of this chapter we make the following assumptions about the execution time of commands. Atomic actions: $\text{SKIP}$ and assignment take one time unit, $\text{DELAY} \ d$ takes exactly $d$ time units (if $d$ is positive, otherwise 0 time units), and for an input or output command we assume that the actual communication (i.e. without waiting) takes one time unit. A guarded command requires one time unit of overhead (e.g. for evaluation of boolean guards, select an open guard, etc.). We assume that there is no overhead for the other composite constructs.

The structure of this chapter is as follows. First the rule for parallel composition is presented in section 4.1. The consequence rule, which includes the use of maximal parallelism, is presented in section 4.2. The assumption/commitment reasoning is modelled by a separate rule, the strengthen rule, which is formulated in section 4.3. The remaining rules and axioms of our proof system are given in three groups. Section 4.4 contains the rules and axioms related to atomic statements of our language. In section 4.5 those related to composite constructs, and in section 4.6 general axioms and rules related to all language constructions are given. (Soundness of the system is proved in the appendix C.)
4.1 The rule for parallel composition

In this section the rule for parallel composition is formulated. In this rule assumptions about
shared channels of the two processes involved are verified and removed. Consider, for the parallel
composition of $S_1$ and $S_2$, the assumption $A_2$ of $S_2$:
- $A_2$ may contain assumptions about joint channels of $S_1$ and $S_2$, and these assumptions must
  be verified by commitment $C_1$ of $S_1$.
- $A_2$ may contain assumptions about external channels of $S_2$. These assumptions are
  maintained in the new network assumption $A$ for $S_1 \parallel S_2$.

This leads to the following proof obligation in the rule: $A \land C_1 \to A_2$, and similar $A \land C_2 \to A_1$.

The post condition of $S_1 \parallel S_2$ is in principle the conjunction of both post conditions, except for
some renaming due to the possibility of different termination times. Similar, in principle a
conjunction of commitments is taken.

Let $jchan = \{ D, D', D ? \mid D \in chan(S_1) \cap chan(S_2) \}$.

(parallel composition)

\[
(A_i, C_i) : \{p_i\} S_i \{q_i\}, \quad i = 1, 2 \\
q_1^{\{q_1:v_{time}\}} \land q_2^{\{q_2:v_{time}\}} \to q^{\text{max}(t_1, t_2) / v_{time}} \\
C_1^{\{b_1:v_{fin}\}} \land C_2^{\{b_2:v_{fin}\}} \to C^{b_1 \land b_2 / v_{fin}} \\
A \land C_1 \to A_2, \quad A \land C_2 \to A_1 \\
(A, C) : \{p_1 \land p_2\} S_1 \parallel S_2 \{q\}
\]

with $t_1$ and $t_2$ logical TIME variables not occurring free in $q_1, q_2$ or $q$, $b_1$ and $b_2$ logical boolean variables not occurring free in $C_1, C_2$ or $C$, and provided:

(1) $\text{chan}(p_i, q_i, C_i) \subseteq \text{chan}(S_i)$,
(2) $\text{var}(p_i, q_i) \subseteq \text{var}(S_i)$,
(3) $\text{uchan}(A_i) \subseteq \text{chan}(S_i)$ and
(4) $\text{dchan}(A) \cap \text{dchan}(C_i) = \emptyset$, for $i \in \{1, 2\}$.

Restrictions (1) and (2) denote that pre condition, post condition and commitment of a process
must refer to program variables or channels of that process only. According to (3) the
assumption $A_i$ of a process should mention communications via channels of that process only.
Requirement (4) expresses that the network assumption $A$ of $S_1 \parallel S_2$ does not refer to waiting for
channels which are mentioned in projections on wait functions in the commitment of one of the
two processes. In appendix C, after the soundness proof of the parallel composition rule,
examples are given to show the need for restrictions (3) and (4).

Note that there is no maximal parallelism constraint in the rule for parallel composition. In our
proof system this constraint is axiomatized in two ways:
in the strengthen rule, which models the assumption/commitment reasoning, where the assumption combined with maximal parallelism restricts the behaviour of a process, and
- in the consequence rule, which states that maximal parallelism can be used for every implication between assertions. So it is possible to use maximality after parallel composition by applying the consequence rule.

The example below illustrates the reasoning with assumptions and commitments at parallel composition.

**example**

Consider the following specifications (*DELAY* is used to represent internal actions):

\[
(A_1 \equiv \text{env waits for } B \text{? from } 2 \land \text{no } B \text{ from } 0 \text{ till } 2 \land \\
\text{env waits for } D \text{? from } 6 \land \text{no } D \text{ from } 3 \text{ till } 6, \\
C_1 \equiv (\text{time} \geq 3 \rightarrow \text{B comm from } 2 \text{ till } 3) \land \text{wait for } D ! \text{ from } 3 \land \text{send } 5 \text{ via } D): \\
\{\text{time} = 0\} S_1 \equiv B ! 1 ; D ! 5 ; \text{DELAY } 2 \{\text{time} = 9\}.
\]

\[
(A_2 \equiv \text{env waits for } D ! \text{ from } 3 \land \text{send } 5 \text{ via } D, \\
C_2 \equiv \text{wait for } D ? \text{ from } 6 \land \text{no } D \text{ from } 3 \text{ till } 6 \land (\text{time} \geq 7 \rightarrow \text{D comm from } 6 \text{ till } 7)): \\
\{\text{time} = 0\} S_2 \equiv \text{DELAY } 6 ; D ? x \{\text{time} = 7 \land x = 5\}.
\]

Take for \(S_1 \parallel S_2\) the assumption: \(A \equiv \text{env waits for } B ? \text{ from } 2 \land \text{no } B \text{ from } 0 \text{ till } 2,\) then clearly: \(A \land C_1 \rightarrow A_2\) and \(A \land C_2 \rightarrow A_1,\) so parallel composition leads to

\[
(A, C_1 \land C_2) : \{\text{time} = 0\} S_1 \parallel S_2 \{\text{time} = 9 \land x = 5\}.
\]

Using the consequence rule (see next section) we can derive the following commitment for \(S_1 \parallel S_2:\)

\[
(\text{time} \geq 3 \rightarrow \text{B comm from } 2 \text{ till } 3) \land (\text{time} \geq 7 \rightarrow \text{D comm from } 6 \text{ till } 7).
\]

\[\square\]

### 4.2 The consequence rule

Important in the proof system is the treatment of maximal parallelism. This maximal parallelism constraint requires that never two processes both wait for the same communication. So there is never waiting for input and waiting for output via a particular channel at a certain point of time. In the proof system this is axiomatized in two ways: in the strengthen rule (see section 4.3), where the assumption combined with maximal parallelism restricts the behaviour of a process, and in the consequence rule, which will be formulated in this section.

When assumptions aren't used locally, parallel composition yields waiting behaviour of both processes in post condition and commitment. By using maximal parallelism in the consequence rule we can then derive stronger timing properties.
Maximal parallelism can be expressed as follows:

\[ MP_D \equiv \forall t : \neg (D? \in W(t) \land D! \in W(t)) \]

where \( t \) is a logical TIME variable, and \( D \in \text{CHAN} \).

\( MP_D \) expresses that there is no point of time with waiting for input and waiting for output via channel \( D \).

We can use this maximality property for every implication between assertions because every point in a computation will satisfy it. An other property of every point in a computation is that there is no communication or waiting in the future yet. For a particular point of time communication function and wait function are empty for future points of time. So we can also use for every implication the following expression \( FUT \), which states that a process does no communications or waiting for communications in the future:

\[ FUT \equiv \forall t > 0 : W(time + t) = \pi(time + t) = \emptyset \]

where \( t \) is a logical TIME variable.

The following example shows the need for \( FUT \).

**example**

The correctness formula below cannot be derived without using some information about the domain of interpretations, where the communication function and the wait function are empty after the current moment of time.

\[ (true, \forall t > 0 : \pi_D(time + t) = \emptyset) \land \{true\} \land \{\forall t > 0 : \pi_D(time + t) = \emptyset\}. \]

\[ \square \]

Now the consequence rule is a straightforward extension of the usual rule in Hoare logic:

\[
\frac{A \land MP_D \land FUT \rightarrow A', C' \land MP_D \land FUT \rightarrow C, p \land MP_D \land FUT \rightarrow p', q' \land MP_D \land FUT \rightarrow q}{(A, C) \land \{p\} \land \{q\}}
\]

where \( D \) is a channel name.

### 4.3 Strengthen rule for assumption-commitment reasoning

Before the rest of the proof system is given, we formulate in this section a general rule which models the assumption/commitment reasoning. By incorporating such a rule in the proof system, the rules and axioms for language constructs can be formulated without using the assumption. So by these rules and axioms we can first derive a commitment of the program without using any assumption. By applying the strengthen rule below, we can add new assumptions and strengthen commitment and post condition.

Let \( w \) be a logical wait variable and \( D \) a channel name, then define

\[ MP_D(w, W) = \forall t : \neg (D? \in w(t) \land D! \in W(t)) \land \neg (D! \in w(t) \land D? \in W(t)). \]
4.4 Rules and axioms for atomic statements

In this section we give rules and axioms for skip, assignment, delay and i/o-commands. In these rules and axioms the assumption will not be used for the commitment and the post condition. Stronger commitments and post conditions can be derived by applying the strengthen rule above.

**skip**

The only effect of the skip-statement is that it takes some time to execute it. Observe that we have to check the commitment for every point of time during and after execution, because C may refer to the time.

\[(A', C') : \{ p \land time = t_0 \} L \{ q' \}\]

\[\frac{MP_D(w, W) \land ( \forall t \in [t_0, time - \delta]) : (A \downarrow t) [\tau/w] \rightarrow ((q' \rightarrow q) \land (C' \rightarrow C))}{(A' \land A, C') : \{ p \land time = t_0 \} L \{ q \}}\]

for \(\delta > 0\), \(D \in CHAN\), and where \(w\) and \(t\) are a logical wait variable and a logical TIME variable, resp., not occurring free in \(q', q, C', C\) or \(A\).

Note that we use a substitution in \(A\) for the wait function, in order to distinguish the waiting of the environment (in \(A\)) from the waiting behaviour of \(L\) (in \(C'\) and \(q'\)).

The axiom above can be adapted easily for a general \(T\) function which assigns to atomic statements an interval such that the execution time is in this interval, and which gives for composite commands the bounds on the overhead for these commands. Then the skip-axiom can be formulated as:

\[(A, C) : \{ q[time + 1/time] \land \forall t \geq 0 : C[time + 1/time, \geq 1/pm] \} SKIP \{ q \}\]

provided \(t\) does not occur free in \(C\).

The assignment and delay axiom are similar to the skip axiom:

**assignment**

\[(A, C) : \{ q[\tau/x, time + 1/time] \land \forall t \geq 0 : C[time + 1/time, \geq 1/pm] \} x := e \{ q \}\]

provided \(t\) does not occur free in \(C\).

**delay**

Remember that a negative delay value yields a zero delay, so the function \(\text{nonneg}\) is applied, which is defined as follows: \(\text{nonneg}(d) = \begin{cases} 0 & \text{if } d < 0, \\ d & \text{if } d \geq 0. \end{cases}\)
(delay) \( (A, C): \{ q [ time + \text{nonneg}(d)/\text{time}] \land \forall t \geq 0: C \{ time + t /\text{time}, t \geq \text{nonneg}(d)/\text{fin} \} \} \) \DELAY d \{ q \}

provided \( t \) does not occur free in \( C \).

output

The rule for the output command reflects what happens in time. We have to check that commitment \( C \) is valid for all points of time after the start, that is,
- for all points of time during the waiting period (of \( t_w \) time units),
- for all points of time during the communication,
- for all points of time after the communication.

Furthermore we have to prove that the post condition holds in the termination state.

Note that in general we do not know the length of the waiting period for this communication, thus we have to prove commitment and postcondition for all possible wait values \( t_w \).

The implications in the rule represent a reasoning in the initial state. In order to reason about future states, after some waiting or after communication, certain substitutions are applied such that if an assertion with substitution is true in the initial state, then this assertion is also true without substitution in a particular point of time in future.

Define the following substitution which characterizes the state of \( W \) and \( \pi \) immediately after execution of the output statement \( D!e \):

\[
\text{termin} \equiv \forall t_w \geq 0 \forall t \geq 0: (C [ time + t ]\{ \text{termin}, t \geq t_w + 1/\text{fin} \}) \land q [ \text{termin}, \text{true}/\text{fin}, \text{time} + t_w + 1/\text{time} ]
\]

\( (A, C): \{ p \} \) \DELAY d \{ q \}

where \( t_w \) and \( t \) are logical TIME variables not occurring free in \( A, C, p \) or \( q \).

input

The input rule has the same structure as the output rule. Since the value received is not known, we have to prove commitment and postcondition for all possible input values \( v \). First a substitution is defined which characterizes the state of \( W, \pi \) and \( x \) after termination of \( D?x \).

\[
\text{termin} \equiv \forall t_w \geq 0 \forall v \forall t \geq 0: (C [ time + t ]\{ \text{termin}, t \geq t_w + 1/\text{fin} \}) \land q [ \text{termin}, \text{true}/\text{fin}, \text{time} + t_w + 1/\text{time} ]
\]

\( (A, C): \{ p \} \) \DELAY d \{ q \}

where \( t_w \) and \( t \) are logical TIME variables not occurring free in \( A, C, p \) or \( q \), and \( v \) is a logical VAL variable not occurring free in \( A, C, p \) or \( q \).

The example below shows how we can prove the examples from the introduction (chapter 1).
example
In this example we use the following notation:
\[ t_D(v) \equiv \text{time first } D \text{ after } v \]
\[ \pi_D(v) \equiv \inf \{ v > v_1 | \pi_D(v) \neq \emptyset \} \]. Note that
\[ \min (t_D(5), t) = \min (\inf \{ v > 5 | \pi_D(v) \neq \emptyset \}, t) = \min (t_D(5), t) \].

We will prove the following formula for \( D \uparrow D \):
\[ (A = \text{env waits for } D? \text{ from } 5 \land \text{no } D \text{ comm from } 3 \text{ till } 5, \]
\[ C = \text{time} \geq 6 \rightarrow D \text{ comm from } 5 \text{ till } 6 \land \text{fin} \): \{ \text{time } = 3 \} D!0 \{ \text{time } = 6 \}. \]
Using the formalization of the abbreviations given in chapter 3, we have to prove:
\[ (A = \left\{ < 5 , \min (t_D(5), \text{time} ) \right\} \subseteq W_D^{-1} \land \pi_D(\left\{ < 3,5 \right\}) = \emptyset, \]
\[ C = \text{time} \geq 6 \rightarrow \left\{ < 5,6 \right\} \subseteq \pi_D^{-1} \land \text{fin} \): \{ \text{time } = 3 \} D!0 \{ q \equiv \text{time } = 6 \}. \]
First take the following commitment and post condition:
\[ C' = (1) \quad < 3 , \min (t_D(3), \text{time} ) \right\} \subseteq W_D^{-1} \land \]
\[ (2) \quad \text{time} \geq t_D(3)+1 \rightarrow \left\{ t_D(3), t_D(3)+1 \right\} \subseteq \pi_D^{-1} \land \text{fin} \]
and
\[ q' \equiv \text{time} = t_D(3)+1 \land < 3, t_D(3) \right\} \subseteq W_D^{-1} \]
Then by the output rule we can prove: \( (\text{true}, C') : \{ \text{time } = 3 \} D!0 \{ q' \}. \)
With the invariance rule (see section 4.6) we can prove: \( (\text{true}, t_0 = 3 ) : \{ t_0 = 3 \} D!0 \{ t_0 = 3 \}. \)
Thus conjunction (see section 4.6) leads to
\[ (\text{true}, C' \land t_0 = 3 ) : \{ \text{time } = 3 \land t_0 = 3 \} D!0 \{ q' \land t_0 = 3 \}. \]
Since \( \text{time } = 3 \land \text{time} = t_0 \rightarrow \text{time } = 3 \land t_0 = 3, \) we can derive by the consequence rule:
\[ (\text{true}, C' \land t_0 = 3 ) : \{ \text{time } = 3 \land \text{time } = t_0 \} D!0 \{ q' \land t_0 = 3 \}. \]
in order to apply the strengthen rule, take \( \delta = 0.5, \) and assume
\[ (3) \quad MP_D(w,W) \text{ and } \]
\[ (4) \quad \forall t \in [t_0, t time \leq 0.5] : (A \downarrow t ) \]
Prove: a) \( C' \land t_0 = 3 \rightarrow C \) and b) \( q' \land t_0 = 3 \rightarrow q. \)
Since \( t_0 = 3 \) may be assumed in order to prove \( C \) or \( q, \) we can derive from (4):
\[ \forall t \in [3, \text{time } \leq -0.5] : (A \downarrow t ) \]
Using (3) and (0) we obtain:
\[ (5) \quad \forall t \in [3, \text{time } \leq -0.5] : W_D(\left\{ < 5 , \min (t_D(5), t ) \right\}) = \emptyset. \]
In order to prove \( C \) we assume \( \text{time } \geq 6. \)
and from \( q' \) we can also derive: \( \text{time} = t_D(5) + 1 \geq 5 + 1 = 6. \)

So suppose \( \text{time} \geq 6 \), then \( \pi_D(<3,5>) = \emptyset \), which leads to

\[
(6) \quad t_D(3) = \inf \{ v > 3 \mid \pi_D(v) \neq \emptyset \} = \inf \{ v > 5 \mid \pi_D(v) \neq \emptyset \} = t_D(5).
\]

a) Assume \( C' \land t_0 = 3 \). Prove \( C \) as follows.

Let \( \text{time} \geq 6 \), then from (5):

\[
\forall t \in [3, 5.5) : W_D(\langle <5, \min(t_D(5), t) \rangle) = \emptyset, \text{ thus } W_D(\langle <5, \min(t_D(5), 5.1) \rangle) = \emptyset.
\]

From (1) and (6): \( <3, \min(t_D(5), 6) \rangle \subseteq W_D^{-1}. \) Thus \( t_D(5) \leq 5. \)

By definition \( t_D(5) \geq 5 \), so \( t_D(5) = 5 = t_D(3). \)

Hence from \( C' \) and \( \text{time} \geq t_D(3) + 1 : <5, 6 > \subseteq \pi_D^{-1} \land \text{fin}. \)

b) Assume \( q' \land t_0 = 3 \). Prove \( q \) as follows.

From \( q' \) and (6): \( \text{time} = t_D(3) + 1 = t_D(5) + 1. \) Then from (5) (take \( t = t_D(5) + 0.1) : W_D(\langle <5, \min(t_D(5), t_D(5) + 0.1) \rangle) = \emptyset, \text{ so } W_D(\langle <5, t_D(5) \rangle) = \emptyset.
\]

From \( q' : <3, t_D(5) \rangle \subseteq W_D^{-1}, \text{ thus } t_D(5) \leq 5. \)

By definition \( t_D(5) \geq 5 \), so \( t_D(5) = 5 = t_D(3). \) Hence from \( q': \text{time} = 6. \)

Now the strengthen rule leads to \( (A, C) ; \{ \text{time} = 3 \land \text{time} = t_0 \} \) D\( \emptyset \{ \text{time} = 6 \}. \)

Since \( t_0 \) does not occur in \( A, C \) or in the post condition, we can use the substitution rule (see section 4.6), and substitute \( \text{time} \) for \( t_0 \) in the precondition. This leads to the desired formula:

\[
(A, C) ; \{ \text{time} = 3 \} \) D\( \emptyset \{ \text{time} = 6 \}. \]

\( \square \)

4.5 Rules for composite constructs

Next we give rules for sequential composition, hiding, guarded command and iteration. Since we give a compositional proof system, to each composite construct corresponds a rule in which a specification of the construct can be derived from its constituents without any further knowledge of the structure of these components.

**sequential composition**

The rule for sequential composition is different from such a rule in [ZRE84], where the two components should have the same assumption/commitment pair. In the rule below for \( S_1 ; S_2 \) the assumption/commitment pairs for \( S_1 \) and \( S_2 \) may be different. The new commitment of \( S_1 : S_2 \) is the commitment of \( S_1 \) as long as \( S_1 \) has not terminated, or (after termination of \( S_1 \), during execution of \( S_2 \)) the commitment of \( S_2 \).

\[
(A_1, C_1) ; \{ p \} S_1 \{ r \} , \ (A_2, C_2) ; \{ r \} S_2 \{ q \} \equiv (A_1 \land A_2, (C_1 \land \neg \text{fin}) \lor C_2) ; \{ p \} S_1 ; S_2 \{ q \}
\]

The two examples below should demonstrate the use of this rule for sequential composition.
example 1
Consider the sequential composition of $D!0$ and $B!1$, first prove for $D!0$:

(A$_1$ $\equiv$ env waits for $D$? from $5$ $\land$ no $D$ from $3$ till $5$,)

$C_1$ $\equiv$ \textit{time} $\geq$ $6$ $\rightarrow$ \textit{fin} : \{ \textit{time} $=$ $3$ \} $D!0$ \{ \textit{time} $=$ $6$ $\land$ $D$ comm from $5$ till $6$ \},

and for $B!1$:

(A$_2$ $\equiv$ env waits for $B$? from $6$,)

$C_2$ $\equiv$ \textit{D} comm from $5$ till $6$ $\land$ (\textit{time} $\geq$ $7$ $\rightarrow$ \textit{B} comm from $6$ till $7$) : \{ \textit{time} $=$ $6$ $\land$ \textit{D} comm from $5$ till $6$ \} $B!1$ \{ \textit{time} $=$ $7$ \}.

The rule for sequential composition yields:

(A$_1$ $\land$ A$_2$, (C$_1$ $\land$ $\neg$ fin) $\lor$ C$_2$) : \{ \textit{time} $=$ $3$ \} $D!0$ ; $B!1$ \{ \textit{time} $=$ $7$ \}.

The commitment ((\textit{time} $\geq$ $6$ $\rightarrow$ fin) $\land$ $\neg$ fin) $\lor$ C$_2$ leads to \textit{time} $<$ $6$ $\lor$ C$_2$, thus $\textit{time} \geq$ $6$ $\rightarrow$ C$_2$.

So we obtain (by the consequence rule) the following commitment for $D!0$ ; $B!1$:

(\textit{time} $\geq$ $6$ $\rightarrow$ \textit{D} comm from $5$ till $6$) $\land$ (\textit{time} $\geq$ $7$ $\rightarrow$ \textit{B} comm from $6$ till $7$).

$\square$

example 2
In the following example of sequential composition $S_1$ has terminating and nonterminating executions, take $S_1$ $\equiv$ $[\textit{TRUE} \rightarrow D!3$ $\Box$ $\textit{TRUE} \rightarrow B!4$ $; * \rightarrow \textit{SKIP}]$, and $S_2$ $\equiv$ $D!5$.

For $S_1$ we can derive (remember that a guarded command requires one time unit overhead):

(A$_1$ $\equiv$ env waits for $D$? from $1$ $\land$ env waits for $B$? from $1$,)

$C_1$ $\equiv$ (C$_{\text{term}}$ $\equiv$ \textit{time} $\geq$ $2$ $\rightarrow$ \textit{D} comm from $1$ till $2$ $\land$ fin) $\lor$

(C$_{\text{nonterm}}$ $\equiv$ \textit{\neg fin} $\land$ (\textit{time} $\geq$ $2$ $\rightarrow$ \textit{B} comm from $1$ till $2$)) : \{ \textit{time} $=$ $0$ \} $S_1$ \{ \textit{time} $=$ $2$ $\land$ C$_{\text{term}}$ \}.

For $S_2$ we can derive

(A$_2$ $\equiv$ env waits for $D$? from $4$ $\land$ no $D$ from $2$ till $4$,)

$C_2$ $\equiv$ C$_{\text{term}}$ $\land$ (\textit{time} $\geq$ $5$ $\rightarrow$ \textit{D} comm from $4$ till $5$) : \{ \textit{time} $=$ $2$ $\land$ C$_{\text{term}}$ \} $S_2$ \{ \textit{time} $=$ $5$ \}.

Then sequential composition yields:

(A$_1$ $\land$ A$_2$, (C$_1$ $\land$ $\neg$ fin) $\lor$ C$_2$) : \{ \textit{time} $=$ $0$ \} $S_1$ ; $S_2$ \{ \textit{time} $=$ $5$ \}.

The commitment (C$_1$ $\land$ $\neg$ fin) $\lor$ C$_2$ implies (C$_{\text{term}}$ $\land$ $\neg$ fin) $\lor$ (C$_{\text{nonterm}}$ $\land$ $\neg$ fin) $\lor$ C$_2$.

Thus \textit{time} $<$ $2$ $\lor$ C$_{\text{nonterm}}$ $\lor$ C$_2$, and hence C$_{\text{nonterm}}$ $\lor$ (\textit{time} $\geq$ $2$ $\rightarrow$ C$_2$) $\equiv$

C$_{\text{nonterm}}$ $\lor$ (\textit{time} $\geq$ $2$ $\rightarrow$ (C$_{\text{term}}$ $\land$ (\textit{time} $\geq$ $5$ $\rightarrow$ \textit{D} comm from $4$ till $5$)))).

This leads to the commitment: C$_{\text{nonterm}}$ $\lor$ (C$_{\text{term}}$ $\land$ (\textit{time} $\geq$ $5$ $\rightarrow$ \textit{D} comm from $4$ till $5$)).

$\square$
hiding
The hiding rule allows us to encapsulate internal communications.
\[(A, C) : \{ p \land \pi_{ujchan} = \emptyset \land W_{djchan} = \emptyset \} S_1 \parallel S_2 \{ q \} \]
\[(A, C) : \{ p \} [ S_1 \parallel S_2 ] \{ q \} \]
where \(ujchan = \text{chan}(S_1) \cap \text{chan}(S_2)\), i.e. the undirected joint channels of \(S_1\) and \(S_2\),
\(djchan = \{ D! , D? \mid D \in \text{chan}(S_1) \cap \text{chan}(S_2)\}\), and
provided \(\text{chan}(A, C, p, q) \cap (ujchan \cup djchan) = \emptyset\).

guarded command
For the guarded command construct we have two rules corresponding to the following two cases:
- at least one of the purely boolean guards is true; then one of the branches with a true purely boolean guard is taken because these guards have priority, or
- none of the purely boolean guards is true.

Let \(G = [ \square b_i \rightarrow S_i, \square b_i' ; IO_i \rightarrow S_i', \square b_i'' ; DELAY d_i \rightarrow S_i'' ]\).
The first rule is applied if one of the purely boolean guards evaluates to true.
Assertion \(\overline{p}\) holds after evaluation of the purely boolean guards (which takes one time unit) and before execution of a \(S_i\)-branch.
\[(\text{guard}1) \quad p \rightarrow \overline{p}[^{\text{time}+\text{time}}] \land \forall i \in [0,1> : C[^{\text{time}+\text{time}, \text{false/fin}}], (A, C) : \{ p \land b_i \} S_i \{ q \}, i = 1,..,n_1 \]
\[(A, C) : \{ p \land \bigvee_{i=1}^{n_1} b_i \} G \{ q \} \]
provided \(t\) does not occur free in \(C\).

In the second rule none of the purely boolean guards is true.
Then we take one of the open delay branches with minimal delay if there was no communication available for the open communication guards within this delay period. This last restriction is denoted by a wait function for the channels of open i/o-guards, with interval length equal to the minimal delay period.
Another possibility is a communication before the minimal delay period has elapsed. Then we include the usual communication record and wait functions for all open i/o-guards.
In order to define the minimal delay period and the set of "open" IO-guards, we have to know which booleans are true. So we have to guess the set of true boolean guards:
\(S\) is the set of indices of \(b_i'\) which are true,
\(T\) is the set of indices of \(b_i''\) which are true.
Define for sets \(S \subseteq \{1,..,n_2\}\) and \(T \subseteq \{1,..,n_3\}\):
\(\text{mindelay} = \min\{ \text{nonneg}(d_i) \mid i \in T\}\), \((\min(\emptyset) = \infty)\) and \(\text{ioset} = \{ \text{type}(IO_i) \mid i \in S\}\).
Expression $B_{S,T}$ checks the guess, represented by $S$ and $T$, for booleans:

$$B_{S,T} \equiv \bigwedge_{k \in S} b_k' \land \bigwedge_{k \in T} \neg b_k'. \land \bigwedge_{k \in S} b_k'' \land \bigwedge_{k \in T} \neg b_k''.$$ For a wrong guess $B_{S,T}$ yields FALSE in the premiss of an implication in the rule, thus satisfying this implication trivially.

In the rule we use auxiliary assertions $\overline{p}$ and $p_i$;
assertion $\overline{p}$ holds after a DELAY-guard and before a $S_i$'-'branch, assertion $p_i$ holds after the $IO_i$-guard and before the $S_i$'-branch.

The following three substitutions represent the state of affairs immediately after an output, after an input and after a delay statement with value $mindelay$, resp.
(in this proof system the overhead associated with a guarded command is one time unit)

For all $S \subseteq \{1, \ldots, n_2\}$, $T \subseteq \{1, \ldots, n_3\}$:

$$p \rightarrow \forall n_1 \forall b_i$$

$$B_{S,T} \land p \rightarrow \forall t \in [0, mindelay + 2] : (C \uparrow time + t) \text{outp} \land p_i \text{outp} \land (time + tw + 2) \Rightarrow \text{false/fin}$$

if $IO_i = D!e$, $i = 1, \ldots, n_2$

$$B_{S,T} \land p \rightarrow \forall t \in [0, mindelay + 2] : (C \uparrow time + t) \text{inp} \land p_i \text{inp} \land (time + tw + 2) \Rightarrow \text{false/fin}$$

if $IO_i = D?x$, $i = 1, \ldots, n_2$

$$(A, C) : \{ p_i \land b_i \} \land S_i \{ q \}, i = 1, \ldots, n_2$$

$$(A, C) : \{ \overline{p} \land \text{nonneg} (d_i) \} \land \text{mindelay} \land b_i'' \} \land S_i \{ q \}, i = 1, \ldots, n_3$$

$$(A, C) : \{ p \} \land G \{ q \}$$

where $v, t$ and $tw$ are logical variables not occurring free in $A, C, p$ or $q$.  

--- 24 ---
iteration
Let \( \Pi \equiv \bigwedge_{i=1}^{n_1} b_i \rightarrow S_i \quad \bigwedge_{i=1}^{n_2} b_i' \rightarrow IO_i \rightarrow S_i' \quad \bigwedge_{i=1}^{n_3} b_i'' \rightarrow \text{DELAY} \quad a_i \rightarrow S_i'' \), and

\[
\text{define } b \equiv \bigvee_{i=1}^{n_1} b_i \quad \bigvee_{i=1}^{n_2} b_i' \quad \bigvee_{i=1}^{n_3} b_i''.
\]

In the following rule \( p \) is an invariant for the guarded command.

Note that when the iteration terminates \( (p \land \neg b) \), one time unit elapses due to the overhead of this statement.

\begin{align*}
(A, C_1) : \{ p \land b \} & G \{ p \} \\
 & \frac{p \land \neg b \rightarrow q \land \forall t \geq 0 : C_2 [r \land \forall t \geq 0 : f / \forall t \geq 0 : g]}{(A, (C_1 \land \neg \text{fin}) \lor C_2) : \{ p \} \ast G \{ q \}}
\end{align*}

provided \( t \) does not occur free in \( C_2 \).

4.6 General rules and axioms

The following rules and axioms are applicable to every language construction. We formulate the usual rules for substitution and conjunction, and an invariance axiom.

\begin{align*}
\text{(substitution)} & \quad (A, C) : \{ p \} L \{ q \} \\
(A, C) & \quad \frac{(A, C) : \{ p / \forall \forall \forall \} \ast L \{ q \}}{(A, C) : \{ p \} L \{ q \}}
\end{align*}

where \( v, c \) and \( w \) are a logical \text{VAL} variable, a logical communication variable and a logical wait variable, resp., and where \( e, f \) and \( g \) are an arbitrary \text{VAL} expression, communication expression and wait expression, resp., and provided \( v, c \) and \( w \) do not occur free in \( A, C \) or \( q \).

\begin{align*}
\text{(conjunction)} & \quad (A_i, C_i) : \{ p_i \} L \{ q_i \}, \quad i = 1, 2 \\
(A_1 \land A_2, C_1 \land C_2) & \quad \frac{(A_1 \land A_2, C_1 \land C_2) : \{ p_1 \land p_2 \} \ast L \{ q_1 \land q_2 \}}{(A_1 \land A_2, C_1 \land C_2) : \{ p_1 \land p_2 \} L \{ q_1 \land q_2 \}}
\end{align*}

\begin{align*}
\text{(invariance)} & \quad (A, C) : \{ p \land C \} L \{ p \}
\end{align*}

provided \( \text{var} (p) \cap \text{var} (L) = \emptyset \), \( \text{chan} (C, p) \cap \text{chan} (L) = \emptyset \),

the special variable \text{time} does not occur in \( p \), and \( \text{fin} \) does not occur in \( C \).

The invariance axiom is needed in order to prove

\[
\{ \text{true} \land \pi_D = \emptyset \} \ast [D ! D \land D ? x] \{ \pi_D = \emptyset \},
\]

which is valid because communications via internal channel \( D \) are hidden by the hiding operator \([..] \).

In appendix A we give a formal semantics of our real-time language, and in appendix C we prove that all rules and axioms of the above given proof system are sound w.r.t. this semantics.
5. CONCLUSION AND FUTURE WORK

In this paper a compositional proof theory has been formulated for a real-time programming language with synchronous message passing. Communication and timing behaviour of terminating and nonterminating processes can be specified relative to assumptions about the environment. Concurrency is modelled as maximal parallelism, and this maximality property can be used locally by using the strengthen rule, which axiomatizes the assumption/commitment reasoning. The relation between assumption and commitment, used in this paper, is motivated in the next paragraph.

Starting point is the rule for parallel composition which should deal with checking and removing assumptions only (see our rule in section 4.1). Then the interpretation of \((A, C) : \{p\} \rightarrow \{q\}\) cannot be a simple implication between \(A\) and \(C\). Because this would lead to validity of \((A_1 \equiv I, C_1 \equiv I) : \{\ldots\} S_1 \{\ldots\}\) and
\[(A_2 \equiv I, C_2 \equiv I) : \{\ldots\} S_2 \{\ldots\},\] for arbitrary \(I, S_1\) and \(S_2\).

Now the commitment of \(S_1\) justifies the assumption of \(S_2; C_1 \rightarrow A_2\), and also \(C_2 \rightarrow A_1\).

So by the intended rule for parallel composition we can derive \((true, I) : \{\ldots\} S_1 \parallel S_2 \{\ldots\}\), which is certainly not true for every \(I\).

In order to avoid such circular reasoning there must be some difference between the point for which we prove \(C\) and the points for which \(A\) can be used. A straightforward idea is to use \(A\) up to (but not including) the point for which we want to prove \(C\). But then the above mentioned circular reasoning is still possible, because \((time > 0 \rightarrow false, time > 0 \rightarrow false) : \{\ldots\} S \{\ldots\}\) would be valid for every program \(S\) (remember that the time domain is dense).

Therefore our interpretation of \((A, C) : \{p\} \rightarrow \{q\}\) requires that there is a positive distance in time between the points where \(A\) can be used and the point for which we prove \(C\).

The main difference with earlier work [H], is that now the special variable \(time\), which refers to the global time, is allowed in assumption and commitment. This makes it possible to specify that something must happen at a certain point of time. More work on large, realistic, examples is needed to show how the proof system can be used. Also important is a comparison with the work of others, such as Zwarico and Lee [ZL], Shankar and Lam [UL], Barbacci and Wing [BW], and Lee and Davidson [LD].

In this paper a rather primitive assertion language is used to investigate the essential features of specifying and verifying real-time systems. For an easy use of the proof system it is important to define suitable abbreviations upon this assertion language, and to give rules for reasoning with these abbreviations. An alternative is to use a simpler assertion language. As a first attempt in this direction, a proof system based on Real-Time Temporal Logic (see [KR],[KVR]) has been -- 26 --
formulated, together with Jennifer Widom, for a simple real-time language [HJ].

The maximal parallelism model represents a possible implementation of parallelism, but during our project it has become clear that a lot of typical real-time applications deal with uniprocessor implementation. Therefore we plan to study uniprocessor implementations with different scheduling policies, priorities and interrupts. Asynchronous communication and broadcast also deserve attention, but it will be rather easy to incorporate these features. Finally relative completeness of the proof systems developed has to be proven formally. It is expected to proceed along the same lines as the relative completeness proof of the [ZRE] system; it will be considered as soon as [Z] becomes available.
A. SEMANTICS

In this appendix a formal denotational semantics is given for the real-time programming language defined in chapter 2. As a starting point served the semantics given in [KSRGA] for CSP-R, a real-time language based on Hoare's CSP [Hoare]. That semantics uses a linear history semantics with a discrete time domain. In our semantics functions are used to denote the events happening at a certain point of time, and we take a dense time domain.

In the next section we describe 4-tuples, which form the basis of our semantic domain of denotations. In section A.2 the domain of denotations is defined, and the particular function defining the semantics is given in section A.3.

A.1 Our basic 4-tuples

In this section we define our basic 4-tuples, which form the basis of the semantic domain. A real-time system will be described from the viewpoint of an external observer with his own clock. So in our semantics there is a global notion of time, that is, at the level of reasoning there is a conceptual global clock. Because two arbitrary communications may be arbitrarily close to each other in time, we take a dense time domain $TIME$.

In this paper we choose $TIME$ equal to the rational numbers, $TIME = \mathbb{Q}$, with the usual addition operation $+$, ordering $<$ and equality $=$. For simplicity (to avoid an elaborate distinction), let the domain $VAL$ for values of identifiers be such that $VAL = TIME$.

The basic domain of denotations for the semantics of a process consists of sets of tuples $(\tau, W, \sigma, \alpha)$, where:

- $\tau$ is a communication function;
  a function from a point in $TIME$ to a set of communication records $(D, v)$, with $D$ a channel name and $v \in VAL$.
  Informal meaning of $(D, v) \in \tau(t)$: at time $t$ a communication via channel $D$ is being performed and $v$ is the communicated value.
  So this record is assigned to every point of time during the communication.

- $W$ is a wait function;
  a function from $TIME$ to a set of directed channels, $W : TIME \rightarrow IP(\{ D, D? \mid D \in \text{CHAN} \})$.
  Informal meaning of $D \in W(t)$: wait at point of time $t$ for an output via channel $D$, and $D? \in W(t)$ means: wait at point of time $t$ for an input via channel $D$.
- $\sigma$ is a state; a mapping from identifiers to values ($\sigma \in \text{STATE}$) or $\bot$, indicating an unfinished computation, or $(\sigma, \top)$, with $\sigma \in \text{STATE}$, modeling progress of time after a computation which has terminated in state $\sigma$, $(\sigma, \top) \in T\text{STATE} \equiv \text{STATE} \times \{\top\}$. Note that the state is local, and cannot be changed by any other process.

- $\alpha$ denotes the time; the tuple represents the state of affairs at time $\alpha$, for $\alpha \in \text{TIME}$.

In the sequel $s$ will stand for the tuple $(\tau, W, \sigma, \alpha)$, and similar $s' = (\tau', W', \sigma', \alpha')$, $s = (\tau, \hat{W}, \hat{\sigma}, \hat{\alpha})$, etc. Such a 4-tuple indicates a "point" in a computation, i.e., it reflects the state of affairs in a computation at a certain point of time.

A tuple $(\tau, W, \sigma, \alpha)$ with $\sigma \in \text{STATE}$ models a finished computation, which terminates at time $\alpha$ in state $\sigma$, with communication function $\tau$ and wait function $W$ produced during the computation.

Tuples $(\tau, W, \sigma, \alpha)$ with $\sigma = \bot$ modeling unfinished computations, are needed to model infinite computations through an infinite chain of approximations.

A tuple $(\tau, W, (\sigma, \top), \alpha)$, with $\sigma \in \text{STATE}$, models a point of time after a terminated computation (which terminates in a state $\sigma$). It is added to the semantics to obtain a straightforward interpretation of correctness formulae because in these formulae it is possible to refer to points of time after the termination time.

Now every computation is modeled by an infinite chain of tuples. For a nontermination computation all states are $\bot$, for a terminating computation there is a state $\sigma$ from $\text{STATE}$ which is preceded, in time, by $\bot$ states, and followed by an infinite chain of tuples with state $(\sigma, \top)$, increasing time component and the same communication and wait function as in the termination tuple.

A.2 Domain of denotations

In this section the tuples are used to define the semantic domain of denotations. This semantic domain $D$ is restricted to those tuples that satisfy the maximal parallelism constraint, that is, never two processes both wait for the same communication. For the wait functions this means that for every channel $D$ there is never a waiting for input via $D$ and a waiting for output via $D$ at the same instant of time.

Let $w$ and $w'$ be wait functions. Then we formulate this constraint as follows:

$$\text{MP}(w, w') \iff \forall t \in \text{TIME} : \neg(D? \in w(t) \land D! \in w'(t)) \land \neg(D? \in w'(t) \land D! \in w(t)).$$

Note: if $\text{max}(w) < \text{min}(w')$ then $\text{MP}(w, w')$. 

---

-29-
The wait function $W$ occurring in a tuple $s$ in the semantics of a program must satisfy this maximality constraint. So in the sequel we restrict us to the following set of tuples:
$$IB = \{(\tau, W, \sigma, \alpha) \mid MP(W, W) \land \max(\tau, W) \leq \alpha\}.$$  
The last clause denotes that nothing has happened in the future yet.

In order to obtain tuples for all points of time after and including the starting time in the semantics, a closure w.r.t. a starting time is applied. After a termination tuple we add tuples with increasing time component and the same trace and wait function as in the termination tuple.

Let $U$ be a set of tuples.

The closure w.r.t. a time $\alpha$ is defined as
$$Close_\alpha(U) = \{s_u \in U \mid \alpha_u \geq \alpha\}$$
$$\cup \{(\tau_u \downarrow \alpha', W_u \downarrow \alpha', \bot, \alpha') \mid \exists \sigma_u, \alpha_u : (\tau_u, W_u, \sigma_u, \alpha_u) \in U \land \alpha \leq \alpha' < \alpha_u\}$$
$$\cup \{(\tau_u, W_u, (\sigma_u, \top), \alpha') \mid \exists \alpha_u : (\tau_u, W_u, \sigma_u, \alpha_u) \in U \land \alpha' > \alpha_u \geq \alpha\}$$

$U$ is called **closed** iff there exists an $\alpha$ such that $Close_\alpha(U) = U$.

The basic domain of denotations is the set of all nonempty closed subsets of $ IB$,  
$$ID = \{D \mid D \subseteq IB \land D \neq \emptyset \land \exists \alpha : Close_\alpha(U) = U\}.$$  

A.3 The function defining the semantics

Finally the particular function defining the semantics is given.

Assume a function $T$ has been given, which assigns to every atomic statement $S$ (i.e. skip, assignment, io, delay) and state $\sigma$ an interval $T_\sigma(S)$, such that the execution time of this statement in this state is an element of the given interval. For the guarded command statement $G$, $T_\sigma(G)$ denotes the overhead needed to execute this statement (e.g. evaluation of boolean guards, selection of an open guard, etc.). Assume there exists a constant $\epsilon > 0$ such that $t \in T_\sigma(G) \rightarrow t \geq \epsilon$. There is no overhead for the other composite constructs. Furthermore, it is assumed that communications take a positive amount of time, i.e $t \in T_\sigma(IO) \rightarrow t > 0$. Parallel composition will be defined such that corresponding io-actions take the same execution time.

Assume the existence of semantic functions $[.]$ for VAL expressions $e$ and boolean expressions $b : [e]_\sigma, [b]_\sigma$.

The variant of a state $\sigma \neq \bot$, $\sigma[\cdot/x]$, is defined as
$$[\sigma[\cdot/x]](x) = v$$
$$[\sigma[\cdot/x]](y) = \sigma(y), \text{ if } y \neq x.$$  

The concatenation of two tuples $s_1$ and $s_2$, notation $s_1 \cdot s_2$ or $s_1s_2$, is defined by
$$s_1 \cdot s_2 = (\tau_1 \cup \tau_2, W_1 \cup W_2, \sigma_2, \alpha_2).$$
The semantics is now defined as a function $M$ which maps a language construction $L$, from an initial state $\sigma \in \text{STATE}$ and an initial time $\alpha$ to an element of $\text{ID}$:

$$M : L \rightarrow (\text{STATE} \times \text{TIME} \rightarrow \text{ID}).$$

**skip**

The semantics of a skip statement shows that the time component is updated with the execution time of this statement; all possible execution times between the bounds given by the $T$-function are included. By taking the closure an element of $\text{ID}$ is obtained.

$$M(\text{SKIP})(\sigma, \alpha) = \text{Close}_\alpha(\{(\sigma, \delta, \alpha + t) \mid t \in T_\sigma(\text{SKIP})\})$$

**assignment**

An assignment statement has a similar semantics, now also the state is updated.

$$M(x := e)(\sigma, \alpha) = \text{Close}_\alpha(\{(\sigma, \delta, \alpha [\delta[I(x)]/\alpha], \alpha + t) \mid t \in T_\sigma(x := e)\})$$

**delay**

A delay statement updates the time component $\alpha$ with the specified time given by the $T$-function. This $T$-function should be such that $t \in T_\sigma(\text{DELAY} d)$ implies $t \geq [d]_T$. Since a negative delay value yields a zero delay, the function $\text{nonneg}$, defined below, is applied to the delay value.

$$\text{nonneg}(v) = \begin{cases} 0 & \text{if } v < 0 \\ v & \text{if } v \geq 0 \end{cases}$$

$$M(\text{DELAY} d)(\sigma, \alpha) = \text{Close}_\alpha(\{(\sigma, \delta, \alpha + \text{nonneg}(t) \mid t \in T_\sigma(\text{DELAY} d)\})$$

**output**

For an output command we include a communication record in the semantics for every point of time during the communication. Assume the process has to wait $w$ time units and execution of the output command is started at time $\alpha$, then the actual communication is performed between $\alpha + w$ and $\alpha + w + t$ for $t \in T_\sigma(D!e)$. This is represented by the communication function: $<\alpha + w, \delta > \mapsto \{(D, \text{value of } e )\}$. Waiting for $w$ time units is denoted by wait function $<\alpha, \alpha + w > \mapsto \{D!\}$. Since waiting time $w$ depends on the other process, we take all possible values for $w$.

$$M(D!e)(\sigma, \alpha) = \text{Close}_\alpha(\{(\sigma, \delta, \alpha + w + t) \mid \text{nonneg}(t) \leq \text{TIME}, \delta \geq 0 \wedge t \in T_\sigma(D!e)\}$$

We use left open intervals, because we want to add something after the starting time, and not at the same point of time as the starting time (note that $w$ may be 0).

Right open intervals are used for the communication function because we want to distinguish separate communications via a particular channel. Otherwise it is possible that $D?x \parallel D!3; D!3$ does not deadlock for certain $T$-functions. For instance, take $T(IO) = [1,2]$, and assume $D?x$ takes two time units and $D!1$ one time unit.
input
The semantics of the input statement is similar to the output command, now the value received is not known, and all possible input values are included.

\[ M(D?x)(\sigma,\alpha) = \text{Close}_\alpha(\{(\alpha+w,\alpha+w+t) \mapsto [(D,v)], \langle \alpha,\alpha+w \rangle \mapsto \{D?\}, \sigma[\alpha/v],\alpha+w+t\} \mid w \in \text{TIME} \land w \geq 0 \land v \in \text{VAL} \land t \in T_\sigma(D?x)) \} ) \]

sequential composition
In order to define the semantics of sequential composition, the semantic function is extended to sets of tuples.

Define the extension of a function \( X : \text{STATE} \times \text{TIME} \rightarrow \text{ID} \) to a set \( U \in \text{ID} \), \( X^* : \text{ID} \rightarrow \text{ID} \):

\[ X^*(U) = \{ s_u \mid s_u \in U \land \sigma_u \in \text{STATE} \land s \in X(\sigma_u,\alpha_u) \} \cup \{ s_u \mid s_u \in U \land \sigma_u = \bot \}. \]

Then the semantics of \( S_1;S_2 \) is defined as

\[ M(S_1;S_2)(\sigma,\alpha) = M(S_2)^*(M(S_1)(\sigma,\alpha)). \]

hiding
Hiding of internal communications just means the projection on external channels:

\[ M([N])(\sigma,\alpha) = [M(N)(\sigma,\alpha)]_{\text{chan}(\{N\})} \]

with for \( U \in \text{ID} \) projection on a set \( \text{cset} \) is defined as follows:

\[ [U]_{\text{cset}} = \{ \{\tau\}_{\text{cset}},[W]_{\text{cset}},\sigma,\alpha \mid (\tau,W,\sigma,\alpha) \in U \} \]

where \([\tau]_{\text{cset}}\) denotes the restriction of \( \tau \) to records with channel name in \( \text{cset} \):

\[ [\tau]_{\text{cset}}(t) = \{(D,v) \mid (D,v) \in \tau(t) \land D \in \text{cset}\}. \]

\([W]_{\text{cset}}\) is defined as \([W]_{\text{cset}}(t) = W(t) \cap \text{cset} \), for every \( t \in \text{TIME} \).

guarded command
For the semantics of the guarded command construction consider two cases:
- at least one of the purely boolean guards is true; then, because of priority for these branches, take the union of the semantics of all branches with a true purely boolean guard.
- none of the purely boolean guards is true:
  - then we take one of the open delay branches with minimal delay if there was no communication available for the open communication guards within this delay period. This last restriction is denoted by wait functions for the channels of open i/o-guards, with interval length equal to the minimal delay period.
  - another possibility is a communication before the minimal delay period has elapsed. Then we include the usual communication record and wait functions for all open i/o-guards.
- Again the wait functions of the environment are restricted in order to satisfy the maximal parallelism constraint.

\( T_\sigma(G) \) represents the time needed to deduce which i/o-branches are open, to compute delays, to select a branch, etc.
Let \( G \equiv \left[ \bigotimes_{i=1}^{n_1} b_i \rightarrow S_i \bigotimes_{i=1}^{n_2} b_i ; IO_i \rightarrow S_i \bigotimes_{i=1}^{n_3} b_i ; \text{DELAY} \ d_i \rightarrow S_i \right] \),

define

\[
\text{mindelay} = \min \{ \text{nonneg} (\llbracket d_i \rrbracket_\sigma) \mid \llbracket b_i \rrbracket_\sigma \} \quad (\min(\sigma) = \infty) \quad \text{ioset} = \{ \text{type}(IO_i) \mid \llbracket b_i \rrbracket_\sigma \}.
\]

\[
M(G)(\sigma, \alpha) = \bigcup_{i=1}^{n_1} [M(S_i)(\sigma, \alpha+t) \mid \llbracket b_i \rrbracket_\sigma \wedge t \in T_\sigma(G)], \quad \text{if } \bigvee_{i=1}^{n_1} \llbracket b_i \rrbracket_\sigma,
\]

and otherwise

\[
\bigcup_{i=1}^{n_2} M(S_i)(\sigma, \alpha+t) \mid \llbracket b_i \rrbracket_\sigma \wedge t \in T_\sigma(G), \quad \text{if } \bigvee_{i=1}^{n_2} \llbracket b_i \rrbracket_\sigma,
\]

i.e.

\[
\bigcup_{i=1}^{n_3} M(S_i)(\sigma, \alpha+t) \mid \llbracket b_i \rrbracket_\sigma \wedge t \in T_\sigma(G) \wedge t' \in T_\sigma(\text{DELAY } d_i) \wedge w \in \text{TIME} \wedge
\]

\[
0 \leq w < \text{mindelay} \wedge \begin{cases}
\text{IO}_i \equiv D x \rightarrow v = [e]_\sigma \wedge \sigma = \sigma \\
\text{IO}_i \equiv \text{D } x \rightarrow v \in \text{VAL} \wedge \sigma = \sigma \left[ \frac{t}{t^'} \right]_\sigma
\end{cases}
\]

Define

\[
M^*(G)(\sigma, \alpha) = \bigcup_{i=0}^\infty \phi_i(\sigma, \alpha),
\]

where \( \phi_i \) are functions from \( \text{STATE} \times \text{TIME} \) to \( \text{ID} \) defined inductively by

\[
\phi_0(\sigma, \alpha) = \{ (\emptyset, \emptyset, \perp, \alpha) \},
\]

\[
\phi_{i+1}(\sigma, \alpha) = \begin{cases}
\phi_i(M(G)(\sigma, \alpha)), & \text{if } \bigvee_{i=1}^{n_1} \llbracket b_i \rrbracket_\sigma \vee \bigvee_{i=1}^{n_2} \llbracket b_i \rrbracket_\sigma \vee \bigvee_{i=1}^{n_3} \llbracket b_i \rrbracket_\sigma \\
\text{Close}_\alpha(\sigma, \sigma, \alpha+t) \mid t \in T_\sigma(G), & \text{otherwise}.
\end{cases}
\]

Note that there exists an \( \epsilon > 0 \) such that \( t \in T_\sigma(G) \rightarrow t \geq \epsilon \) for all \( \sigma \in \text{STATE} \).

Define

\[
M^*(G)(\sigma, \alpha) = \bigcup_{i=0}^\infty \phi_i(\sigma, \alpha),
\]

where \( \phi_i \) are functions from \( \text{STATE} \times \text{TIME} \) to \( \text{ID} \) defined inductively by

\[
\phi_0(\sigma, \alpha) = \{ (\emptyset, \emptyset, \perp, \alpha) \},
\]

\[
\phi_{i+1}(\sigma, \alpha) = \begin{cases}
\phi_i(M(G)(\sigma, \alpha)), & \text{if } \bigvee_{i=1}^{n_1} \llbracket b_i \rrbracket_\sigma \vee \bigvee_{i=1}^{n_2} \llbracket b_i \rrbracket_\sigma \vee \bigvee_{i=1}^{n_3} \llbracket b_i \rrbracket_\sigma \\
\text{Close}_\alpha(\sigma, \sigma, \alpha+t) \mid t \in T_\sigma(G), & \text{otherwise}.
\end{cases}
\]

Note that there exists an \( \epsilon > 0 \) such that \( t \in T_\sigma(G) \rightarrow t \geq \epsilon \) for all \( \sigma \in \text{STATE} \).

Then the semantics of the iteration statement \( M^*(G) \) can also be defined as the unique fixed point over \( \{ X : \text{STATE} \times \text{TIME} \rightarrow \text{ID} \mid s \in X(\sigma, \alpha) \rightarrow \alpha \geq \alpha \} \) of the following equation. This will be proven in appendix D.
\[ X = \lambda_{\sigma, \alpha}. \text{if } \bigvee_{i=1}^{n_1} [b_i] \sigma \lor \bigvee_{i=1}^{n_2} [b_i] \sigma \lor \bigvee_{i=1}^{n_3} [b_i] \sigma \]

then \( X'(M(G)(\sigma, \alpha)) \)

else \( \text{Close}_\alpha \{(0,0,0,0) \mid t \in T_\sigma(G)\} \).

**parallel composition**

For the **parallel composition** \( S_1 \parallel S_2 \) the semantics includes for a certain point of time a combination of two tuples from \( M(S_1) \) and \( M(S_2) \) representing that point of time. We take the pointwise union of communication and wait functions, provided the communication functions of \( S_1 \) and \( S_2 \) are identical for the joint communications of \( S_1 \) and \( S_2 \). Furthermore the states are combined; \( S_1 \parallel S_2 \) only terminates if both processes have terminated. Remember that there are no shared variables.

Let \( j\text{chan} = \{ D, D!, D? \mid D \in \text{chan}(S_1) \cap \text{chan}(S_2) \} \).

\[ M(S_1 \parallel S_2)(\hat{\sigma}, \hat{\alpha}) = \text{Close}_\alpha \{(\tau_1 \cup \tau_2, W_1 \cup W_2, \sigma, \alpha) \mid \]

\[ \forall i \in \{1,2\} \exists \sigma_i, \alpha_i : (\tau_i, W_i, \sigma_i, \alpha_i) \in M(S_i)(\hat{\sigma}, \hat{\alpha}) \land \alpha = \alpha_i \land \]

\[ [\tau_1]_{\text{chan}} = [\tau_2]_{\text{chan}} \land MP(W_1, W_2) \land \]

\[
\begin{cases}
\sigma = \bot, & \text{if } \sigma_1 = \bot \lor \sigma_2 = \bot \\
\sigma \in \text{TSTATE}, & \text{if } \sigma_1 \in \text{TSTATE} \land \sigma_2 \in \text{TSTATE} \\
\sigma \in \text{STATE}, & \text{if otherwise}
\end{cases}
\]

\[ (\sigma_1 \neq \bot \land \sigma_2 \neq \bot) \rightarrow \text{state}(\sigma)(x) = \begin{cases} 
\text{state}(\sigma_i)(x), & x \in \text{var}(S_i) \\
\hat{\sigma}(x), & x \notin \text{var}(S_1, S_2) 
\end{cases} \]

where \( \text{state}(\sigma) = \begin{cases} 
\sigma & \text{if } \sigma \in \text{STATE} \\
\overline{\sigma} & \text{if } \sigma = (\sigma, \top)
\end{cases} \).

The tuple in the semantics of \( S_1 \parallel S_2 \) is an element of \( \mathcal{B} \), because

- \( MP(W_1 \cup W_2, W_1 \cup W_2) \), since \( MP(W_1, W_1), MP(W_2, W_2) \) and \( MP(W_1, W_2) \),
- \( \max(\tau_1, \tau_2, W_1, W_2) \preceq \alpha_1 = \alpha_2 = \alpha \).
B. FORMAL INTERPRETATION

In this appendix the formal interpretation of the assertion language is described in section B.1. Section B.2 contains a formal definition of the interpretation of our correctness formulae \((A, C) : \{p\} L \{q\}\).

B.1 Interpretation of assertions

This section concerns the interpretation of the assertion language. An assertion \(p\) is interpreted in a logical variable environment \(\gamma\), which assigns values to logical variables, and a tuple \(s = (\tau, W, \sigma, \alpha)\).

Notation: \([[p]]_{\gamma}s\).

If \(p\) contains free program variables \((\text{var}(p) \neq \emptyset)\) then \(p\) is interpreted in tuples \(s\) with \(\sigma \in \text{STATE}\). The interpretation is straightforward, some examples:

\([[c]]_{\gamma}s = \gamma(c), \quad [[w]]_{\gamma}s = \gamma(w), \quad [[b]]_{\gamma}s = \gamma(b), \quad [[v]]_{\gamma}s = \gamma(v),

\([[\text{fin}]]_{\gamma}s = (\sigma \neq \bot),

\([[\tau]]_{\gamma}s = \tau, \quad [[W]]_{\gamma}s = W, \quad [[\text{fin}]]_{\gamma}s = (\sigma \neq \bot), \quad [[\text{time}]]_{\gamma}s = \alpha.

Let \(ce\) denote a communication expression, i.e. an expression denoting a communication function, and \(we\) denotes a wait expression.

\([[ce]]_{D}lD_{s} = [[ce]]_{s}lD_{s}, \quad [[we]]_{D}lD_{s} = [[we]]_{s}lD_{s}.

If \(\sigma \in \text{STATE}\) then \([[x]]_{\gamma}s = \sigma(x)\).

An assertion \(p\) is called valid, denoted by \(\models p\), iff \(\forall \gamma \forall s : \sigma \in \text{STATE} \rightarrow [[p]]_{\gamma}s\).

B.2 Formal definition of a correctness formula

A formal definition of the interpretation of a correctness formula is given in this section. Again we use the abbreviation \(s = (\tau, W, \sigma, \alpha)\), \(\hat{s} = (\hat{\tau}, \hat{W}, \hat{\sigma}, \hat{\alpha})\) etc.

Furthermore we define the set of all wait functions satisfying maximal parallelism:

\(WFUN = \{W \mid W :\text{TIME} \rightarrow \mathcal{P}(\{D \mid D? \mid D \in \text{CHAN}\}) \land MP(W, W)\}\).

A correctness formula is called valid, denoted by \(\models (A, C) : \{p\} L \{q\}\), iff

\(\exists \delta > 0 \forall \gamma \forall s \in B, \delta \in \text{STATE} : [[p]]_{\gamma}s \rightarrow \forall s \in M(L)(\delta, \hat{\delta}) \forall w \in WFUN : MP(w, \hat{W} \cup W) \land \forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta : [[A]]_{\gamma}(\hat{\tau} \cup \tau \downarrow \alpha', w \downarrow \alpha', \bot, \alpha') \rightarrow ( [[C]]_{\gamma}s \land (\sigma \in \text{STATE} \rightarrow [[q]]_{\gamma}s s))\).
C. SOUNDNESS OF THE PROOF SYSTEM

In this appendix soundness of the proof system, as formulated in chapter 4, is proven. That is, every correctness formula \((A, C) : \{p\} L \{q\}\) which is derivable in the proof system of chapter 4 is also valid: \(\models (A, C) : \{p\} L \{q\}\), as defined in appendix B, using the semantics of appendix A. This is achieved by proving the following theorem.

**Theorem:**
All rules and axioms of the proof system of chapter 4 are sound w.r.t. the semantics given in appendix A.

**proof:**
We have to prove that every axiom is valid, and that the conclusion of a rule is valid given the validity of the premisses.

Let \(s\) denote the tuple \((\tau, W, \sigma, \alpha)\), and \(\hat{s} = (\hat{\tau}, \hat{W}, \hat{\sigma}, \hat{\alpha})\) etc., and define the set of all wait functions:

\[
WFUN = \{ W \mid W:TIME \rightarrow IP(\{D!, D? \mid D \in CHAN\}) \land MP(W, W) \}.
\]

Proving \(\models (A, C) : \{p\} L \{q\}\) is by the definition of a correctness formula equivalent to:

there exists a \(\delta > 0\) such that

\[
\text{given } y, \hat{s} \in \mathcal{B}, \hat{\sigma} \in \text{STATE} \text{ with } [p]y\hat{s}, \text{ we can prove}
\]

for all \(s \in M(L)(\delta, \hat{\sigma})\) and for all \(w \in WFUN\) with

\[
MP(w, \hat{W} \cup W) \land \forall \alpha', \hat{\sigma} \leq \alpha' < \alpha - \delta : \[A\]y((\hat{\tau} \sqcup \tau \downarrow \alpha'; w \downarrow \alpha', \bot, \alpha'))
\]

the following:

i) \([C]y\hat{s} s\)

ii) \(\sigma \in \text{STATE} \rightarrow [q]y\hat{s} s\).

Note that, because of the restrictions on the assertion language (see section 3.3), \([A]y(\tau, W, \sigma, \alpha)\) does not depend on state \(\sigma\), so we often write \([A]y(\tau, W, \ldots, \alpha)\).

**parallel composition**

According to the parallel composition rule for \(S_1 \parallel S_2\), let \(jchan = \{ D, D!, D? \mid D \in \text{chan}(S_1) \cap \text{chan}(S_2) \}\), and assume the following:

1. \(\models (A_i, C_i) : \{p_i\} S_i \{q_i\}, \quad i = 1, 2\)
2. \(\models q_1[\tau_{\text{time}}] \land q_2[\tau_{\text{time}}] \rightarrow q[\max(\tau_1, \tau_2)_{\text{time}}]\)
\( t_1 \) and \( t_2 \) are logical TIME variables not occurring free in \( q_1, q_2 \) or \( q \).

\( b_1 \) and \( b_2 \) are logical boolean variables not occurring free in \( C_1, C_2 \) or \( C \).

\( \text{chan} \left( p_i, q_i, C_i \right) \subseteq \text{chan} \left( S_i \right) \),

\( \text{var} \left( p_i, q_i \right) \subseteq \text{var} \left( S_i \right) \),

\( \text{uchan} \left( A_i \right) \subseteq \text{chan} \left( S_i \right) \) and

\( \text{dchan} \left( A \right) \cap \text{dchan} \left( C_i \right) = \emptyset \), for \( i \in \{1, 2\} \).

We have to prove: \( \models (A, C): \{p_1 \land p_2\} S_1 \parallel S_2 \{q\} \).

\text{proof:}

There exist \( \delta_1 > 0 \) and \( \delta_2 > 0 \) such that for \( i \in \{1, 2\} \):

\[
\forall \gamma \forall s \in \mathcal{B}, \delta \in \text{STATE} : \left( [p_i] \gamma s \rightarrow \forall s_i \in \mathcal{M}(S_i)(\delta, \alpha') \forall w \in \text{WFUN} : \right.
\]

\[
MP(w, \hat{W} \cup W) \land \forall \alpha', \beta \leqslant \alpha' < \alpha_i - \delta_i : \left( [A_i] \gamma (\hat{\tau} \cup \tau_i \downarrow \alpha', \gamma, \downarrow \alpha', \downarrow) \right)\]

\[
\rightarrow \left( [C_i] \gamma s_i \land (s_i \in \text{STATE} \rightarrow [q_i] \gamma s_i \right)).
\]

Take \( \delta = \max (\delta_1, \delta_2) \).

Choose \( \gamma, \hat{s} \in \mathcal{B}, \delta \in \text{STATE} \) arbitrary. Assume

\( \{p_1 \land p_2\} \gamma \hat{s} \).

For the proof we first introduce the following notation (given \( \hat{s} \)), for two tuples \( s_1 \) and \( s_2 \):

\[
\text{par} \left( s_1, s_2 \right) = \{(\tau_1 \cup W_1 \cup W_2, \sigma, \alpha) : \forall i \in \{1, 2\} : s_i \in \mathcal{M}(S_i)(\delta, \alpha) \land \alpha = \alpha_i \land \]

\[
[\tau_i]_{\text{chan}} = [\tau_2]_{\text{chan}} \land MP(W_1, W_2) \land \]

\[
\begin{cases}
\sigma = \bot & \text{if } \sigma_1 = \bot \land \sigma_2 = \bot \\
\sigma \in \text{STATE} & \text{if } \sigma_1 \in \text{STATE} \land \sigma_2 \in \text{STATE} \\
\sigma \in \text{STATE} & \text{if otherwise}
\end{cases}
\]

\[
\left( \sigma_1 \neq \bot \land \sigma_2 \neq \bot \rightarrow \text{state} \left( \sigma \right)(x) = \left( \begin{array}{ll}
\text{state} \left( \sigma_1 \right)(x), x \in \text{var} \left( S_i \right) \\
\delta \left( x \right), x \notin \text{var} \left( S_1, S_2 \right)
\end{array} \right) \right) \}
\]

where \( \text{state} \left( \sigma \right) = \left( \begin{array}{ll}
\sigma & \text{if } \sigma \in \text{STATE} \\
\overline{\sigma} & \text{if } \sigma = (\overline{\sigma}, \top)
\end{array} \right) \).

First prove by induction on \( n \in \mathbb{N} \) the following statement.
\[ P(n) \equiv \begin{cases} \text{if } s \in M(S_1 \uplus S_2)(\hat{\alpha}, \hat{\alpha}), \text{ with } s \in \text{par}(s_1, s_2) \text{ for certain } s_1, s_2, \end{cases} \]

\[
\alpha \in [\hat{\alpha} + n \delta, \hat{\alpha} + (n+1)\delta], \quad w \in WFUN \text{ with }
\]

\[
MP(w, \hat{W} \cup W) \text{ and } \forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta : ![A]y(\hat{T} \uparrow \alpha', \hat{W} \uparrow \alpha', \bot, \alpha').
\]

Then there exist \( w_i \in WFUN \) with \( MP(w_i, \hat{W} \cup W_i) \) and

\[
\forall \alpha', \hat{\alpha} \leq \alpha' < \alpha_i - \delta_i : ![A_i]y(\hat{T} \uparrow \alpha', w_i \uparrow \alpha', \bot, \alpha')
\]

(and thus \( ![C_i]y_{s_i} \), for \( i \in \{1, 2\} \)).

- Basic step: \( n = 0 \), so \( \alpha \in [\hat{\alpha}, \hat{\alpha} + \delta] \), i.e. \( \alpha \in [\hat{\alpha}, \hat{\alpha} + \min(\delta_1, \delta_2)] \).

But then there are no \( \alpha' \) with \( \hat{\alpha} \leq \alpha' < \alpha_i - \delta_i \) \((\alpha_i = \alpha \) by definition of \( \text{par}(..) \)), so \( P(0) \) holds for \( w_1 = \emptyset \).

- Induction step: let \( n > 0 \) and suppose \( P(k) \) holds for \( k < n \). Prove: \( P(n) \).

Suppose \( s \in M(S_1 \uplus S_2)(\hat{\alpha}, \hat{\alpha}) \), with \( s \in \text{par}(s_1, s_2) \) for certain \( s_1, s_2 \).

1. \( \alpha \in [\hat{\alpha} + n \delta, \hat{\alpha} + (n+1)\delta] \),
2. \( w \in WFUN \) with \( MP(w, \hat{W} \cup W) \) and
3. \( \forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta : ![A]y(\hat{T} \uparrow \alpha', w \uparrow \alpha', \bot, \alpha') \).

Let \( (i, j) \in \{(1, 2), (2, 1)\} \).

Take \( w_i = [W_i \uplus W_j \uparrow \text{chan}(C_i)] \uplus w_{\text{chan}(A)} \).

Then \( MP(w_i, \hat{W} \uplus W) \) because
- \( MP(\hat{W}, \hat{W}) \) from \( \hat{\alpha} \in B \),
- \( MP(\hat{W}, W_i) \) from \( \max(\hat{W}) \leq \hat{\alpha} \leq \min(W_i) \) and \( W_i(\hat{\alpha}) = \emptyset \),
- \( MP(W_j, \hat{W}) \) from \( \max(\hat{W}) \leq \hat{\alpha} \leq \min(W_j) \) and \( W_j(\hat{\alpha}) = \emptyset \),
- \( MP(W_j, W_i) \) from definition of \( \text{par}(s_1, s_2) \),
- \( MP(w, \hat{W} \uplus W) \) from (12): \( MP(w, \hat{W} \uplus W_1 \uplus W_2) \).

Let \( \alpha' \) be such that \( \hat{\alpha} \leq \alpha' < \alpha_i - \delta_i \).

Define \( s_i' = (T_i \downarrow \alpha', W_i \downarrow \alpha', s_i', \alpha') \) for \( i \in \{1, 2\} \),

where we choose \( \sigma_1' \) and \( \sigma_2' \) such that \( \text{par}(s_1, s_2) \neq \emptyset \).

Take \( s' \in \text{par}(s_1, s_2) \). Then \( s' \in M(S_1 \uplus S_2)(\hat{\alpha}, \hat{\alpha}) \), and

\( \hat{\alpha} \leq \alpha' < \alpha_i - \delta_i = \alpha - \delta_i < \alpha - \min(\delta_1, \delta_2) = \alpha - \delta \), i.e. \( \hat{\alpha} \leq \alpha' < \alpha - \delta \).

Hence (from (11)): \( \alpha' \in [\hat{\alpha} + k \delta, \hat{\alpha} + (k+1)\delta] \), for \( k < n \).

From (13): \( \forall \alpha'', \hat{\alpha} \leq \alpha'' < \alpha' - \delta : ![A]y(\hat{T} \uparrow \alpha'', w \uparrow \alpha'', \bot, \alpha'') \).

Thus from (12) and the induction hypothesis: \( ![C_i]y_{s_i} s_i' \).
So $\llbracket C \rrbracket_\gamma(\tau \cup \tau', [W \cup W_j]_{\text{chan}(C_j)}, \sigma_j', \alpha_j').$

From (6): $\text{chan}(C_j) \subseteq \text{chan}(S_j)$,
from $s \in \text{par}(s_1, s_2)$: $\tau_j' = [\tau']_{\text{chan}(S_j)}$, and $\alpha_j' = \alpha'$, thus
$\llbracket C_j \rrbracket_\gamma(\tau \cup \tau', [W \cup W_j]_{\text{chan}(C_j)}, \sigma_j', \alpha').$

By (9): $\llbracket C_j \rrbracket_\gamma(\tau \cup \tau', [W \cup W_j]_{\text{chan}(C_j)} \cup [w \downarrow \alpha']_\text{chan}(A), \sigma_j', \alpha').$

Note $W \cup W_j \downarrow \alpha' = W \cup W_j$, thus

(I) $\llbracket C_j \rrbracket_\gamma(\tau \cup \tau', w_i \downarrow \alpha', \sigma_j', \alpha').$

From (13): $\llbracket A \rrbracket_\gamma(\tau \cup \tau', w \downarrow \alpha', \tau', \alpha')$. From $s \in \text{par}(s_1, s_2)$: $\tau \downarrow \alpha' = \tau$. So by (9):
$\llbracket A \rrbracket_\gamma(\tau \cup \tau', W \cup W_j]_{\text{chan}(C_j)} \cup [w \downarrow \alpha']_\text{chan}(A), \alpha').$

(II) $\llbracket A \rrbracket_\gamma(\tau \cup \tau', w_i \downarrow \alpha', \tau', \alpha').$

So by (4), (I) and (II): $\llbracket A_i \rrbracket_\gamma(\tau \cup \tau', w_i \downarrow \alpha', \tau', \alpha').$

Since $[\tau']_{\text{chan}(S_j)} = \tau_i'$, and from (8): $\text{uchan}(A_i) \subseteq \text{chan}(S_j)$ we derive
$\llbracket A_i \rrbracket_\gamma(\tau \cup \tau_i \downarrow \alpha', \alpha_i')$.

Note $\tau_i' = \tau_i \downarrow \alpha'$, thus $\llbracket A_i \rrbracket_\gamma(\tau \cup \tau_i \downarrow \alpha', \alpha_i')$, thus $P(n)$.

Now let $s \in M(S_1 \uplus S_2)(\tilde{\sigma}, \tilde{\alpha})$, $w \in \text{WFUN}$ with

$MP(w, W \cup W)$ and $\forall \alpha', \tilde{\alpha} \leq \alpha' < \alpha - \delta: \llbracket A \rrbracket_\gamma(\tau \downarrow \alpha', w \downarrow \alpha', \tau', \alpha')$.

Then there are $s_1, s_2$ with $s \in \text{par}(s_1, s_2)$.

Furthermore, there exists an $n$ with $\alpha \in [\tilde{\alpha} + n \delta, \tilde{\alpha} + (n + 1) \delta)$ (since $\delta > 0$).

i) From $P(n)$: $\frac{\sum_{i=1}^{2} \llbracket C_i \rrbracket_\gamma s_i}{i}.$

Take, temporarily, a new environment $\gamma' = \gamma^{\sigma_1 \downarrow \alpha_i, \sigma_2 \downarrow \alpha_i}$, then (by 5):
$\frac{\sum_{i=1}^{2} \llbracket C_i \rrbracket_\gamma s_i}{i} = \frac{\sum_{i=1}^{2} \llbracket C_i \rrbracket_\gamma s_i}{i}$, that is, $\frac{\sum_{i=1}^{2} \llbracket C_i \rrbracket_\gamma s_i}{i}.$

By (6) and $\alpha = \alpha_i$ (see definition $\text{par}(..)$):
$\frac{\sum_{i=1}^{2} \llbracket C_i \rrbracket_\gamma s_i}{i}.$

i.e., $\frac{\sum_{i=1}^{2} \llbracket C_i \rrbracket_\gamma s_i}{i}.$

Thus, by (3): $\llbracket C \rrbracket_\gamma s.$

Note that $\llbracket \text{par} = b_1 \land b_2 \rrbracket_\gamma s$ (see definition $\text{par}(..)$), so $\llbracket C \rrbracket_\gamma s.$

hence, using (5): $\llbracket C \rrbracket_\gamma s.$

\hfill -- 39 --
Let $\sigma \in \text{STATE}$, then

$$\sigma_1 \in \text{STATE} \land (\sigma_2 \in \text{STATE} \lor \sigma_2 \in T\text{STATE})$$

or $\sigma_2 \in \text{STATE} \land (\sigma_1 \in \text{STATE} \lor \sigma_1 \in T\text{STATE})$.

So there are $s_1', s_2'$ with $s_i' \in M(S_i)(\delta, \alpha)$, and $\sigma_i \in \text{STATE}$.

Observe that $\alpha_1' = \alpha_1$ or $\alpha_2' = \alpha_2$ so max $(\alpha_1', \alpha_2') = \alpha_1 = \alpha_2 = \alpha$.

From $P(n)$ and (1): $\bigwedge_{i=1}^{2} \llbracket q_i \rrbracket y \hat{s} s_i$.

Take $y' = y[\sigma_1' \theta_1, \sigma_2' \theta_2]$, then $\bigwedge_{i=1}^{2} \llbracket q_i \rrbracket [\theta_1, \theta_2] y \hat{s} s_i$.

Since $s_i'$ and $s_i$ only differ in the time component and a possible addition of T to the state, we obtain by (6) and (7): $\bigwedge_{i=1}^{2} \llbracket q_i \rrbracket [\theta_1, \theta_2] y \hat{s} s$.

Then (2) leads to $\llbracket q \rrbracket [\max(t_1, t_2), \theta_1, \theta_2] y \hat{s} s$. Also $\llbracket \text{time} = \max(t_1, t_2) \rrbracket y \hat{s} s$.

Thus $\llbracket q \rrbracket y \hat{s} s$, so by using (6): $\llbracket q \rrbracket y \hat{s} s$.

\[\square\]

The need for requirement (8) is demonstrated by the following example: consider

$(\pi_E(5) \neq \emptyset, ...) : \{\text{time} = 4\} \text{ DELAY } 2 \{\text{false}\}$

(Which can be proven as follows:

first prove $(\text{true, ...}) : \{\text{time} = 4 \land \pi_E(5) = \emptyset\} \text{ DELAY } 2 \{\text{time} = 6 \land \pi_E(5) = \emptyset\}$,

use the strengthen rule to derive

$(\pi_E(5) \neq \emptyset, ...), \{\text{time} = 4 \land \pi_E(5) = \emptyset\} \text{ DELAY } 2 \{\text{time} = 6 \land \pi_E(5) = \emptyset \land \pi_E(5) \neq \emptyset\}$).

Parallel composition with $(..., ...), \{\text{time} = 4\} E!0 \{...\}$ leads to

$(\pi_E(5) \neq \emptyset, ...), \{\text{time} = 4\} \text{ DELAY } 2 \parallel E!0 \{\text{false}\}$, which is not valid.

The following example shows why requirement (9) is needed: consider

$(A_1 \equiv \text{false}, C_1 \equiv \text{false}) : \{\text{time} = 4\} \text{ SKIP } \{\cdots\}$, and

$(A_2 \equiv \text{true}, C_2 \equiv W_D(5) = \emptyset) : \{\text{time} = 4\} \text{ DELAY } 2 \{\cdots\}$.

Take $A \equiv W_D(5) \neq \emptyset$, then $A \land C_1 \rightarrow A_2$ and $A \land C_2 \rightarrow A_1$, so parallel composition would lead to $(W_D(5) \neq \emptyset, \text{false}) : \{\text{time} = 4\} \text{ SKIP } \parallel \text{ DELAY } 2 \{\cdots\}$, which is certainly not valid.

consequence

Let $D \in \text{CHAN}$, define $MP_D \equiv \forall t : (D \in W(t) \land D \notin W(t))$ and

$FUT \equiv \forall t > 0 : W(t + t) = \pi(t + t) = \emptyset$.

-- 40 --
Assume
(1) \( \models (A', C') : \{ p' \} L \{ q' \} \)
(2) \( \models A \land MP_D \land FUT \rightarrow A' \)
(3) \( \models C' \land MP_D \land FUT \rightarrow C \)
(4) \( \models p \land MP_D \land FUT \rightarrow p' \)
(5) \( \models q' \land MP_D \land FUT \rightarrow q \)
Prove: \( \models (A, C) : \{ p \} L \{ q \} \).

proof
From (1): there exists a \( \delta > 0 \) such that

(6) \( \forall \gamma \forall \hat{s} \in \mathcal{IB}, \hat{\sigma} \in \text{STATE} : \llbracket p \rrbracket_\gamma \hat{s} \rightarrow \forall s \in \text{M}(L)(\hat{\sigma}, \hat{\omega}) \forall w \in \text{WFUN} : \\
MP(w, \hat{W} \cup W) \land \forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta : \llbracket A \rrbracket_\gamma (\hat{\tau} \omega \tau \downarrow \alpha', w \downarrow \alpha', \bot, \alpha') \\
\rightarrow ( \llbracket C \rrbracket_\gamma \hat{s} s \land ( \sigma \in \text{STATE} \rightarrow \llbracket q \rrbracket_\gamma \hat{s} s ) ). \)

Take this \( \delta \), and choose \( \gamma, \hat{s} \in \mathcal{IB}, \hat{\sigma} \in \text{STATE} \) arbitrary. Assume

(7) \( \llbracket p \rrbracket_\gamma \hat{s}. \)

Let \( s \in \text{M}(L)(\hat{\sigma}, \hat{\omega}) \) and \( w \in \text{WFUN} \) with \( MP(w, \hat{W} \cup W) \) and

(8) \( \forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta : \llbracket A \rrbracket_\gamma (\hat{\tau} \omega \tau \downarrow \alpha', w \downarrow \alpha', \bot, \alpha'). \)

Remember that \( \mathcal{IB} \) consists of all tuples \((\tau, w, \sigma, \alpha)\) with \( MP(W, W) \) and \( \max(\tau, W) \leq \alpha \), thus \( \llbracket MP_D \land FUT \rrbracket \gamma \hat{s} \), since \( \hat{s} \in \mathcal{IB}. \)

Hence (7) and (4) lead to \( \llbracket p \rrbracket_\gamma \hat{s}. \) Since \( w \in \text{WFUN} : \llbracket MP_D \rrbracket_\gamma (\hat{\tau} \omega \tau \downarrow \alpha', w \downarrow \alpha', \bot, \alpha'). \)

Furthermore: \( \llbracket FUT \rrbracket_\gamma (\hat{\tau} \omega \tau \downarrow \alpha', w \downarrow \alpha', \bot, \alpha'). \)

Thus using (2) we can derive from (8): \( \forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta : \llbracket A \rrbracket_\gamma (\hat{\tau} \omega \tau \downarrow \alpha', w \downarrow \alpha', \bot, \alpha'). \)

Then (6) leads to \( \llbracket C \rrbracket_\gamma \hat{s} s \) and \( \sigma \in \text{STATE} \rightarrow \llbracket q \rrbracket_\gamma \hat{s} s. \)

By the definition of the semantic function: \( s \in \mathcal{IB}, \) so \( \llbracket MP_D \land FUT \rrbracket \gamma \hat{s} s. \)

Hence, using (3) and (5), we derive: \( \llbracket C \rrbracket_\gamma \hat{s} s \) and \( \sigma \in \text{STATE} \rightarrow \llbracket q \rrbracket_\gamma \hat{s} s. \)

\( \square \)

strengthen

Let \( D \) be a channel name, and

\( MP_D(w, W) = \forall t : \neg (D? \in w(t) \land D!t \in W(t)) \land \neg (D!t \in w(t) \land D? \in W(t)). \)

Let \( \delta > 0, \) and assume

(1) \( \models (A', C') : \{ p \land time = t_0 \} L \{ q' \} \)
(2) \( \models MP_D(w, W) \land ( \forall t \in [t_0, time - \delta) : (A \downarrow t)[w] ) \rightarrow ( (q' \rightarrow q) \land (C' \rightarrow C) ) \)
(3) \( w \) and \( t \) are a logical wait variable and a logical TIME variable, resp.,
not occurring free in $q'$, $q$, $C'$, $C$ or $A$.

Prove: $\models (A \land A, C) : \{p \land \text{time} = t_0\} \ L \ \{q\}$.

proof:
From (1): there exists a $\delta_1 > 0$ such that

(4) $\forall \gamma \forall \delta \in \text{IB}, \delta \in \text{STATE} : [[p \land \text{time} = t_0]] \gamma \delta \rightarrow \forall \delta \in M(L)(\delta, \hat{\alpha}) \forall w \in \text{WFUN} :

\[\text{MP}(w, \hat{w} \cup W) \land \forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta_1 : [[A]] \gamma (\hat{\alpha} \cup \tau \downarrow \alpha', \hat{w} \downarrow \alpha', \perp, \alpha') \rightarrow (([[C]] \gamma \delta s \land (\sigma \in \text{STATE} \rightarrow [[q]] \gamma \delta s))$.

Take $\delta' = \min (\delta, \delta_1)$. Choose $\gamma, \delta \in \text{IB}, \delta \in \text{STATE}$ arbitrary, assume

(5) $[[p \land \text{time} = t_0]] \gamma \delta$. Then

(6) $\gamma(t_0) = \hat{\alpha}$.

Let $s \in M(L)(\delta, \hat{\alpha}), w \in \text{WFUN}$. Assume

(7) $\text{MP}(w, \hat{w} \cup W)$ and

(8) $\forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta : [[A \land A]] \gamma (\hat{\alpha} \cup \tau \downarrow \alpha', \hat{w} \downarrow \alpha', \perp, \alpha')$.

Take $\gamma' = \gamma^{\tau \downarrow \alpha'}$. Then from (7):

(9) $[[\text{MP}(w, W)]] \gamma' \delta s$.

Since $\delta' \leq \delta$, by (8): $\forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta : [[A \land \text{time} = t_0]] \gamma'^{\tau \downarrow \alpha'}(\hat{\alpha} \cup \tau \downarrow \alpha', \hat{w} \cup W, \sigma, \alpha)$.

And thus, using $\gamma'(t_0) = \hat{\alpha}$ (from (6)):

(10) $[[\forall t \in [t_0, \text{time} - \delta > : (A \land t)]^{\tau \downarrow \alpha'}]] \gamma' \delta s$.

Since $\delta' \leq \delta_1$, we obtain from (8): $\forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta_1 : [[A]] \gamma (\hat{\alpha} \cup \tau \downarrow \alpha', \hat{w} \downarrow \alpha', \perp, \alpha')$.

Note that, because of (3), we can take $\gamma'$ instead of $\gamma$ in the line above, and also in (5).

Together with (4) and (5) this leads to $[[C]] \gamma' \delta s$ and $\sigma \in \text{STATE} \rightarrow [[q]] \gamma' \delta s$.

Now using (9) and (10) this leads by (2) to $[[C]] \gamma' \delta s$ and $\sigma \in \text{STATE} \rightarrow [[q]] \gamma' \delta s$.

By (3) we can take $\gamma$ instead of $\gamma'$, thus $[[C]] \gamma \delta s$ and $\sigma \in \text{STATE} \rightarrow [[q]] \gamma \delta s$.

\[\square\]

skip
We have to prove:

$\models (A, C) : \{q \uparrow \text{time} + 1 / \text{time} \land \forall t \geq 0 : C \uparrow \text{time} + t / \text{time} \rightarrow \uparrow \text{fails}} \ SKIP \ \{q\}$

where $t$ does not occur free in $C$.

proof:
Since this axiom doesn't use $A$ we can take $\delta$ arbitrary greater than 0.

Choose $\gamma, \delta \in \text{IB}, \delta \in \text{STATE}$ arbitrary. Assume
Let $s \in M(SKIP)(\hat{\sigma},\hat{\alpha})$. Then there are three possibilities:

a) $s = (\hat{\tau},\hat{W},\hat{\alpha} + \tau)$ for $0 < \tau < 1$.

b) $s = (\hat{\tau},\hat{W},\hat{\alpha} + \tau)$ for $\tau = 1$.

c) $s = (\hat{\tau},\hat{W},(\hat{\alpha},\hat{T}),\hat{\alpha} + \tau)$ for $\tau > 1$.

i) Prove $\llbracket C \rrbracket y \hat{s}$ as follows: given $\llbracket C [time + t e_{time}] \rrbracket y \hat{s}$, thus

$\llbracket C \rrbracket y(\hat{\tau},\hat{W},\hat{\alpha} + \tau)$ for $0 < \tau < 1$ (remember $\llbracket \text{fin} \rrbracket y \hat{s}$ if $\sigma \neq \bot$).

Similar $\llbracket C \rrbracket y(\hat{\tau},\hat{W},(\hat{\alpha},\hat{T}),\hat{\alpha} + \tau)$ for $\tau > 1$ and $\sigma \neq \bot$.

ii) If $\sigma \in \text{STATE}$ then $s = (\hat{\tau},\hat{W},\hat{\alpha} + 1)$. Hence, from (1),

$\llbracket q [time + t e_{time}] \rrbracket y(\hat{\tau},\hat{W},\hat{\alpha} + 1)$ thus $\llbracket q \rrbracket y(\hat{\tau},\hat{W},\hat{\alpha} + 1) \Rightarrow \llbracket q \rrbracket y \hat{s}$.

Soundness of the assignment and the delay axiom requires a similar proof.

output

Define

$\text{termin} \equiv \forall \gamma \in \{D \uparrow t e \}_{\gamma}, \forall \gamma \in \{D \uparrow t e + 1 \}_{\gamma}$

and assume

(1) $\vdash p \Rightarrow \forall \nu \geq 0 \forall t \geq 0 : (C \downarrow time + t \{\text{termin}, \tau \uparrow t e + 1 \}_{\gamma} ) \land q [\text{termin}, \tau \uparrow t e, \tau \uparrow t e + 1 ]$

(2) $t e$ and $t$ are logical TIME variables not occurring free in $A, C, p$ or $q$.

Prove: $\vdash (A, C) : \{p \} D \uparrow \{ p \}$.

proof:

Take $\delta > 0$ arbitrary. Choose $\gamma, \hat{s} \in B$, $\delta \in \text{STATE}$ arbitrary. Assume

(3) $\llbracket p \rrbracket y \hat{s}$.

Let $s \in M(D \uparrow \{p \})(\hat{\delta},\hat{\alpha})$.

Then, by definition of $M(D \uparrow \{p \})$, there exists a $\nu_{0} \in \text{TIME}$, $\nu_{0} \geq 0$ (the waiting time) and

$s = (\langle \hat{\alpha} + \nu_{0} \rangle, \hat{\alpha} + \nu_{0} + 1) \mapsto (\{D, [\gamma_{t e} \hat{\delta}])\rangle, \hat{\alpha} + \nu, (\langle \hat{\alpha}, \hat{\alpha} + \nu_{0} \rangle \mapsto \{D ! \}) \downarrow \hat{\alpha} + \nu, \downarrow, \hat{\alpha} + \nu )$

for $0 \leq \nu < \nu_{0} + 1$,

or $s = (\langle \hat{\alpha} + \nu_{0} \rangle, \hat{\alpha} + \nu_{0} + 1) \mapsto (\{D, [\gamma_{t e} \hat{\delta}])\rangle, \hat{\alpha}, \hat{\alpha} + \nu_{0} + 1) \mapsto \{D ! \}, \hat{\alpha}, \hat{\alpha} + \nu_{0} + 1 )$

or $s = (\langle \hat{\alpha} + \nu_{0} \rangle, \hat{\alpha} + \nu_{0} + 1) \mapsto (\{D, [\gamma_{t e} \hat{\delta}])\rangle, \langle \hat{\alpha}, \hat{\alpha} + \nu_{0} \rangle \mapsto \{D ! \}, (\hat{\delta}, \hat{T} ), \hat{\alpha} + \nu )$ for $\nu > \nu_{0} + 1$.

Use (1), (3) and take $\gamma' = y^{\{t e_{\nu_{0}} \}}$, then:

for all $t \geq 0 : \llbracket (C \downarrow time + t \{\text{termin}, \tau \uparrow t e + 1 \}_{\gamma}) \rrbracket y \hat{s}$.

Observe $\llbracket (C \downarrow time + t \{\text{termin}, \tau \uparrow t e + 1 \}_{\gamma}) \rrbracket y \hat{s}$

$\llbracket C [time + t e_{time} \gamma, \tau \uparrow t e + 1 ]_{\gamma} \rrbracket \llbracket \text{termin}, \tau \uparrow t e + 1 \gamma \rrbracket$
Soundness of the input rule proceeds along the same lines.

**sequential composition**

Let

(1) \( \vdash (A_1, C_1): \{ p \} S_1 \{ r \} \), and

(2) \( \vdash (A_2, C_2): \{ r \} S_2 \{ q \} \).

Prove: \( \vdash (A_1 \land A_2, (C_1 \land \neg \text{fin}) \lor C_2): \{ p \} S_1 ; S_2 \{ q \} \).

**proof:**

For the validity of (1) and (2) there exist \( \delta_1 > 0 \) and \( \delta_2 > 0 \). Take \( \delta = \min(\delta_1, \delta_2) > 0 \).

Choose \( \gamma, \delta \in \mathcal{B}, \delta \in \text{STATE} \) arbitrary. Assume

(3) \( \llbracket p \rrbracket \gamma s \).

Let \( s \in M(S_1 ; S_2)(\delta, \alpha) \), thus (see semantics sequential composition):

a) \( s \in M(S_1)(\delta, \alpha) \) and \( \sigma = \bot \), or

b) \( s = s_1 s_2 \) with \( s_1 \in M(S_1)(\delta, \alpha), \sigma_1 \in \text{STATE} \) and \( s_2 \in M(S_2)(\sigma_1, \alpha_1) \).

Let \( w \in WFUN \), and assume

(4) \( \text{MP}(w, \hat{w} \cup \hat{w}) \) and

(5) \( \forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta : \llbracket A_1 \land A_2 \rrbracket (\hat{\tau} \cup \tau \downarrow \alpha', w \downarrow \alpha', \bot, \alpha') \).

Prove

i) \( \llbracket (C_1 \land \neg \text{fin}) \lor C_2 \rrbracket \gamma s \) and

ii) \( \sigma \in \text{STATE} \rightarrow \llbracket q \rrbracket \gamma s \).

Consider the two cases above for \( s \):

a) \( s \in M(S_1)(\delta, \alpha) \) and \( \sigma = \bot \).

Then from (5) and \( \delta = \min(\delta_1, \delta_2) \): \( \forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta_1 : \llbracket A_1 \rrbracket (\hat{\tau} \cup \tau \downarrow \alpha', w \downarrow \alpha', \bot, \alpha') \).

Together with (1), (3) and (4) this leads to \( \llbracket C_1 \rrbracket \gamma s \).

Since \( \sigma = \bot \) we have \( \llbracket \neg \text{fin} \rrbracket \gamma s \). Thus \( \llbracket C_1 \land \neg \text{fin} \rrbracket \gamma s \), which proves i).
ii) is trivial, since \( \sigma \not\in \text{STATE} \).

b) \( s = s_1 \cdot s_2 \) with \( s_1 \in M(S_1)(\hat{\alpha}, \hat{\alpha}) \), \( s_1 \in \text{STATE} \) and \( s_2 \in M(S_2)(\alpha_1, \alpha_1) \).

Then \( \alpha = \alpha_2 \geq \alpha_1 \) and \( \delta \leq \delta_1 \), so \( \alpha_1 - \delta_1 \leq \alpha - \delta \), thus we obtain from (5):

\[
\forall \alpha', \hat{\alpha} \leq \alpha' < \alpha_1 - \delta_1 : [A_1] y(\tau \cup \tau' \downarrow \alpha', w \downarrow \alpha', \bot, \alpha').
\]

Note: \( \tau = \tau_1 \cup \tau_2 \) and \( \min(\tau_2) \geq \alpha_1 \), thus

\[
\forall \alpha', \hat{\alpha} \leq \alpha' < \alpha_1 - \delta_1 : [A_2] y(\tau_1 \cup \tau_2 \downarrow \alpha', w \downarrow \alpha', \bot, \alpha').
\]

Together with (1), (3), (4) (note: \( MP(w, \hat{\hat{W}} \cup W_1 \cup W_2) \) implies \( MP(w, \hat{\hat{W}} \cup W_1) \)) and \( \sigma \in \text{STATE} \), this leads to \( [r] y \hat{\hat{s}} s_1 \).

Since \( \alpha_1 \geq \hat{\alpha} \), \( \alpha = \alpha_2 \) and \( \delta \leq \delta_2 \) we derive from (5):

\[
\forall \alpha', \alpha_1 \leq \alpha' < \alpha_2 - \delta_2 : [A_2] y(\tau_1 \cup \tau_2 \downarrow \alpha', w \downarrow \alpha', \bot, \alpha').
\]

Thus from (1), using \( [r] y \hat{\hat{s}} s_1 \), we derive \( [C_2] y \hat{\hat{s}} s_1 s_2 \), i.e. \( [C_2] y \hat{\hat{s}} s \), which proves i).

If \( \sigma \in \text{STATE} \), then \( \sigma_2 = \sigma \): \( \sigma_2 \in \text{STATE} \), so also from (2): \( [q] y \hat{\hat{s}} s_1 s_2 \). hence ii).

□

hiding

Given

(1) \( (A, C) : \{ p \land \pi_{ujchan} = \emptyset \land W_{djchan} = \emptyset \} S_1 \rightarrow S_2 \{ q \} \),

where \( ujchan = \text{chan}(S_1) \cap \text{chan}(S_2) \), \( djchan = \{ D \mid D \in \text{chan}(S_1) \cap \text{chan}(S_2) \} \), and

\( jchan = ujchan \cup djchan \), and provided

(2) \( \text{chan}(A, C, p, q) \cap jchan = \emptyset \).

We have to prove: \( (A, C) : \{ p \} [S_1 \rightarrow S_2] \{ q \} \).

proof:

Define for a tuple \( s \) and a set of (directed and undirected) channels \( cset \) the subtraction of \( cset \) from \( s \), notation \( s \setminus cset \), as the tuple obtained from \( s \) by deleting all records (in \( T \) and \( W \)) with channel in \( cset \).

From (1): there exists a \( \delta > 0 \) such that:

(3) \( \forall y \forall \hat{\hat{s}} \in IB, \hat{\alpha} \in \text{STATE} : [p \land \pi_{ujchan} = \emptyset \land W_{djchan} = \emptyset] y \hat{\hat{s}} \setminus jchan \rightarrow \)

\( \forall \hat{\hat{s}} \in M(S_1 \rightarrow S_2)(\hat{\alpha}, \hat{\alpha}) \forall \hat{w} \in WFUN :

\[
MP(\hat{w}, \hat{w} \setminus jchan \cup \hat{\hat{W}}) \land \forall \alpha', \hat{\alpha} \leq \alpha' < \alpha - \delta : [A] y(\hat{\hat{w}} \setminus jchan \cup \hat{\hat{w}} \downarrow jchan \downarrow \alpha', w \downarrow \alpha', \bot, \alpha')
\]

\( \rightarrow ( [C] y \hat{\hat{s}} \setminus jchan \check{S} \land ( \sigma \in \text{STATE} \rightarrow [q] y \hat{\hat{s}} \setminus jchan \check{S} ) ) \).

Choose \( y, \check{s} \in IB, \hat{\alpha} \neq \bot \), and \( \hat{\alpha} \neq \bot \) arbitrary. Assume

\(-- 45 --\)
(4) $[[p]]y\bar{s}$.

By using (2) this leads to $[[p]]y\bar{s}\backslash jchan$.

Observe that $[[\pi_{u_jchan} = \emptyset \land W_{d_jchan} = \emptyset]]y\bar{s}\backslash jchan$, so

(5) $[[p \land \pi_{u_jchan} = \emptyset \land W_{d_jchan} = \emptyset]]y\bar{s}\backslash jchan$.

Let $s \in M((S_1 \parallel S_2)(\emptyset,\emptyset))$, then there exists an $\bar{s}$ with $s = \bar{s}\backslash jchan$, and

(6) $\bar{s} \in M(S_1 \parallel S_2)(\emptyset,\emptyset)$.

Let $w \in WFUN$ such that

(7) $MP(w,\bar{W} \cup \bar{W})$ and

(8) $\forall \alpha', \vec{\alpha} \leq \alpha' < \delta : [[A]]y(\bar{\tau} \cdot w \downarrow \alpha', \downarrow, \downarrow, \downarrow, \downarrow)$

Since $\vec{\alpha} = \alpha$ and $\bigcap jchan = \emptyset$ (see (2)), (8) leads to

$\forall \alpha', \vec{\alpha} \leq \alpha' < \alpha - \delta : [[A]]y(\bar{\tau} \cdot w \downarrow \alpha', \downarrow, \downarrow, \downarrow, \downarrow)$

Since $W = \bar{W} \backslash jchan$, from (7):

$MP(w \backslash jchan, \bar{W} \backslash jchan \cup \bar{W} \backslash jchan)$, thus $MP(w \backslash jchan, \bar{W} \backslash jchan \cup \bar{W})$.

So take $\bar{w} = w \backslash jchan$ in (3), then $[[C]]y\bar{s}\backslash jchan \bar{s}$ and $\sigma \in STATE \rightarrow [[q]]y\bar{s}\backslash jchan \bar{s}$.

From (2): $[[C]]y\bar{s}\bar{s} \land (\sigma \in STATE \rightarrow [[q]]y\bar{s}\bar{s})$.

From (2) and $\vec{\sigma} = \sigma$ : $[[C]]y\bar{s}\bar{s}\backslash jchan$ and $\sigma \in STATE \rightarrow [[q]]y\bar{s}\bar{s}\backslash jchan$.

Thus $[[C]]y\bar{s}\bar{s}$ and $\sigma \in STATE \rightarrow [[q]]y\bar{s}\bar{s}$.

$\Box$

Proving soundness of the guarded command rule "guard1" is straightforward and omitted here.

The second rule "guard2" requires a rather long and tedious proof based on the same techniques used for soundness of the delay axiom and the rules for i/o and sequential composition. Also the soundness proof of the iteration rule is omitted, because it is very similar to the usual proof for such a rule.

Soundness of the substitution, and the conjunction rule, and the invariance axiom is straightforward.
D. SEMANTICS ITERATION

In this appendix we prove that the semantics of the iteration construct can be given as the unique solution of a fixed point equation.

Let \( G \equiv \left[ \bigwedge_{i=1}^{n_1} b_i \rightarrow S_i \bigwedge_{i=1}^{n_2} b_i' \rightarrow IO_i \rightarrow S_i' \bigwedge_{i=1}^{n_3} b_i'' \rightarrow \text{DELAY } d_i \rightarrow S_i'' \right] \),

\[ b \equiv \bigvee_{i=1}^{n_1} b_i \lor \bigvee_{i=1}^{n_2} b_i' \lor \bigvee_{i=1}^{n_3} b_i'' , \]

and for \( X : \text{STATE} \times \text{TIME} \rightarrow \text{ID} \) define \( F(X) \) as follows:

\[
F(X) = \lambda_{\sigma, \alpha}. \text{if } \left[ a \right] \sigma \text{ then } X'(M(G)(\sigma,0')) \text{ else } \text{Close}_\alpha((\emptyset, \emptyset, \sigma, \alpha+t) \mid t \in T_\sigma(G)) .
\]

The semantics of the iteration statement is defined as the limit of a chain of approximations.

Define \( M(*G)(\sigma, \alpha) = \bigcup_{i=0}^{\infty} \phi_i(\sigma, \alpha) \) where \( \phi_i \) are functions from \( \text{STATE} \times \text{TIME} \) to \( \text{ID} \) defined inductively by \( \phi_0(\sigma, \alpha) = \{ (\emptyset, \emptyset, \top, \alpha) \} \), and \( \phi_{i+1} = F(\phi_i) \) for \( i \geq 0 \).

Remember that we assume that there exists a constant \( \epsilon > 0 \) such that \( t \in T_\sigma(G) \rightarrow t \geq \epsilon \) for all \( \sigma \in \text{STATE} \).

Then the semantics of the iteration statement \( M(*G) \) can also be defined as the unique fixed point over \( \{ X : \text{STATE} \times \text{TIME} \rightarrow \text{ID} \mid s \in X(\sigma, \alpha) \rightarrow \alpha' \geq \alpha \} \) of the following equation: \( X = F(X) \).

In this appendix we prove two things:

a) \( \bigcup_{i=0}^{\infty} \phi_i \) is a solution of the equation above, and

b) \( X = F(X) \) has at most one solution.

These two points imply that the fixed point equation has a unique solution.
Proof:

a) First note that $s' \in \bigcup_{i=0}^\infty \phi_i (\sigma, \alpha)$ implies $\alpha' \geq \alpha$, because $s' \in M(G)(\sigma, \alpha)$ implies $\alpha' \geq \alpha$.

Furthermore, we have to prove: $\bigcup_{i=0}^\infty \phi_i (\sigma, \alpha) = F(\bigcup_{i=0}^\infty \phi_i ) (\sigma, \alpha)$ for all $\sigma \in \text{STATE}$, and $\alpha \in \text{TIME}$. Let $\sigma \in \text{STATE}$ and $\alpha \in \text{TIME}$, then this will be split up in two parts:

1) for all $j: \phi_j (\sigma, \alpha) \subseteq F(\bigcup_{i=0}^\infty \phi_i ) (\sigma, \alpha)$, and

2) if $s' \in F(\bigcup_{i=0}^\infty \phi_i ) (\sigma, \alpha)$ then there exists a $j$ such that $s' \in \phi_j (\sigma, \alpha)$.

proof:

1) Let $j = 0$, and $s' \in \phi_0 (\sigma, \alpha)$. Then $s' = (\emptyset, \emptyset, \bot, \alpha)$.

For all $X: (\emptyset, \emptyset, \bot, \alpha) \in X^*(M(G)(\sigma, \alpha))$ since $(\emptyset, \emptyset, \bot, \alpha) \in M(G)(\sigma, \alpha)$ (which follows from $t \in T_\sigma(G) \Rightarrow t > 0$).

Also $(\emptyset, \emptyset, \bot, \alpha) \in \text{Close}_\sigma[(\emptyset, \emptyset, \sigma, \alpha + t) \mid t \in T_\sigma(G)]$.

Thus $s' \in F(\bigcup_{i=0}^\infty \phi_i ) (\sigma, \alpha)$.

> Let $j > 0$.

If $\lnot [b]_\sigma$ then

$\phi_j (\sigma, \alpha) = \text{Close}_\sigma[(\emptyset, \emptyset, \sigma, \alpha + t) \mid t \in T_\sigma(G)] = F(\bigcup_{i=0}^\infty \phi_i ) (\sigma, \alpha)$.

If $[b]_\sigma$ then

$\phi_j (\sigma, \alpha) = \phi_{j-1}(M(G)(\sigma, \alpha) \subseteq (\bigcup_{i=0}^\infty \phi_i )^*(M(G)(\sigma, \alpha)) = F(\bigcup_{i=0}^\infty \phi_i ) (\sigma, \alpha)$.

2) Let $s' \in F(\bigcup_{i=0}^\infty \phi_i ) (\sigma, \alpha)$.

If $\lnot [b]_\sigma$ then $s' \in \text{Close}_\sigma[(\emptyset, \emptyset, \sigma, \alpha + t) \mid t \in T_\sigma(G)]$, so take $j = 1; s' \in \phi_1 (\sigma, \alpha)$.

If $[b]_\sigma$ then $s' \in (\bigcup_{i=0}^\infty \phi_i )^*(M(G)(\sigma, \alpha))$, that is, (use $\bigcup_{i=0}^\infty \phi_i )^* = \bigcup_{i=0}^\infty \phi_i$)

there exists a $k$ with $s' \in \phi_k^*(M(G)(\sigma, \alpha))$, thus $s' \in \phi_{k+1} (\sigma, \alpha)$, so take $j = k + 1$. □
b) Let $Y_1$ and $Y_2$ be two solutions of $X = F(X)$, such that $s' \in Y_1(\sigma, \alpha) \Rightarrow \alpha' \geq \alpha$.

We prove for $\sigma \in \text{STATE}$ and $\alpha \in \text{TIME}$: $Y_1(\sigma, \alpha) \subseteq Y_2(\sigma, \alpha)$.

If $\neg [b] \sigma$ then

\[ Y_1(\sigma, \alpha) = F(Y_1)(\sigma, \alpha) = \text{Close}_\alpha((\sigma, \alpha + t) \mid t \in T_{\sigma}(G)) = F(Y_2)(\sigma, \alpha) = Y_2(\sigma, \alpha). \]

If $[b] \sigma$ then $Y_1(\sigma, \alpha) = F(Y_1)(\sigma, \alpha) = Y'_1(\sigma, \alpha)$ and

\[ Y_2(\sigma, \alpha) = F(Y_2)(\sigma, \alpha) = Y'_2(\sigma, \alpha). \]

We prove with induction on $n$: for all $\sigma \in \text{STATE}$, $\alpha \in \text{TIME}$ and $s'$ with $[b] \sigma$:

if $\alpha' - \alpha \leq n \epsilon$ then $s' \in Y_1(\sigma, \alpha) \Rightarrow s' \in Y_2(\sigma, \alpha)$.

**proof:**

> Basic step: $n = 0$, so $\alpha' \leq \alpha$.

If $s' \in Y_1(\sigma, \alpha) = Y'_1(\sigma, \alpha)$, then $\alpha' \geq \alpha$, so $\alpha' = \alpha$.

Furthermore, $s' \in M(\sigma, \alpha)$ with $\sigma = \perp$,

because if there exists an $\overline{s} \in M(\sigma, \alpha)$ with $\overline{\sigma} \in \text{STATE}$, and $s'' \in Y_1(\overline{\sigma}, \overline{\alpha})$ such that $s' = s''$ then $\alpha' = \alpha'' \geq \overline{\alpha}$, and by our assumption: $\overline{\alpha} - \alpha \geq \epsilon > 0$.

Thus $\alpha' = \alpha'' \geq \overline{\alpha} > \alpha$; contradiction.

Conclusion: $s' \in M(\sigma, \alpha)$ and $\sigma = \perp$,

but then also $s' \in Y_2(\sigma, \alpha)$.

> Induction step: assume the statement above holds for all $k < n$.

Let $\alpha' - \alpha \leq n \epsilon$, $s' \in Y_1(\sigma, \alpha) = Y'_1(\sigma, \alpha)$ then there are two possibilities:

- $s' \in M(\sigma, \alpha)$ with $\sigma = \perp$, then $s' \in Y_2(\sigma, \alpha)$.
- there exists an $\overline{s} \in M(\sigma, \alpha)$ with $\overline{\sigma} \in \text{STATE}$, $s'' \in Y_1(\overline{\sigma}, \overline{\alpha})$ and $s' = s''$.

Then by our assumption: $\alpha - \alpha \geq \epsilon$, so $\alpha - \overline{\alpha} \leq -\epsilon$, and thus $\alpha'' - \overline{\alpha} = \alpha' - \overline{\alpha} = \alpha' - \alpha - \overline{\alpha} \leq n \epsilon - \epsilon = (n - 1) \epsilon$.

If $\neg [b] \overline{\sigma}$ then $s'' \in Y_1(\overline{\sigma}, \overline{\alpha}) = Y_2(\overline{\sigma}, \overline{\alpha})$ (see above).

If $[b] \overline{\sigma}$ then from $s'' \in Y_1(\overline{\sigma}, \overline{\alpha})$ the induction hypothesis yields $s'' \in Y_2(\overline{\sigma}, \overline{\alpha})$.

Together with $\overline{s} \in M(\sigma, \alpha)$, $\overline{\sigma} \in \text{STATE}$ we obtain

$s' = \overline{s} s'' \in Y'_2(\sigma, \alpha) = Y_2(\sigma, \alpha)$.

\[\square\]
E. REFERENCES


### Available Reports from the Theoretical Computing Science Group

<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Title</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIR82.1 R. Kuiper, W.P. de Roever</td>
<td>Fairness Assumptions for CSP in a Temporal Logic Framework</td>
<td>DESCARTES</td>
</tr>
<tr>
<td>TIR83.1 R. Koymans, J. Vytopil, W.P. de Roever</td>
<td>Real-Time Programming and Synchronous Message passing (2nd ACM PODC)</td>
<td>DESCARTES</td>
</tr>
<tr>
<td>TIR83.2 H. Barringer, R. Kuiper</td>
<td>Towards the Hierarchical, Temporal Logic, Specification of Concurrent Systems</td>
<td>DESCARTES</td>
</tr>
<tr>
<td>TIR84.1 R. Gerth, W.P. de Roever</td>
<td>A Proof System for Concurrent Ada Programs (SCP4)</td>
<td>DESCARTES</td>
</tr>
<tr>
<td>TIR84.2 R. Gerth</td>
<td>Transition Logic - how to reason about temporal properties in a compositional way (16th ACM FOCS)</td>
<td>DESCARTES</td>
</tr>
<tr>
<td>TIR84.3 H. Barringer, R. Kuiper, A. Pnueli</td>
<td>Now you may compose Temporal Logic Specifications (Proc. STOC84)</td>
<td>DESCARTES</td>
</tr>
<tr>
<td>TIR84.4 H. Barringer, R. Kuiper</td>
<td>Hierarchical Development of Concurrent Systems in a Temporal Logic Framework</td>
<td>DESCARTES</td>
</tr>
<tr>
<td>TIR85.1 W.P. de Roever</td>
<td>The Quest for Compositionality - a survey of assertion-based proof systems for concurrent programs, Part I: Concurrency based on shared variables (IFIP85)</td>
<td>DESCARTES</td>
</tr>
<tr>
<td>TIR85.2 O. Grünberg, N. Francez, J. Makowsky, W.P. de Roever</td>
<td>A proof-rule for fair termination of guarded commands (Inf.&amp; Control 1986)</td>
<td>DESCARTES</td>
</tr>
<tr>
<td>TIR85.3</td>
<td>F.A. Stomp, W.P. de Roever, R. Gerth</td>
<td>The $\mu$-calculus as an assertion language for fairness arguments (Inf.&amp; Control 1987)</td>
</tr>
<tr>
<td>TIR85.4</td>
<td>R. Koymans, W.P. de Roever</td>
<td>Examples of a Real-Time Temporal Logic Specification (LNCS207)</td>
</tr>
<tr>
<td>TIR85.5</td>
<td>H. Barringer, R. Kuiper, A. Pnueli</td>
<td>A Compositional Temporal Approach to a CSP-like Language</td>
</tr>
<tr>
<td>TIR86.1</td>
<td>R. Koymans</td>
<td>Specifying Message Passing and Real-Time Systems (extended abstract)</td>
</tr>
<tr>
<td>TIR86.2</td>
<td>J. Hooman, W.P. de Roever</td>
<td>The Quest goes on: A Survey of Proof Systems for Partial Correctness of CSP (LNCS227)</td>
</tr>
<tr>
<td>TIR86.3</td>
<td>R. Gerth, L. Shira</td>
<td>On Proving Communication Closedness of Distributed Layers (LNCS236)</td>
</tr>
<tr>
<td>TIR86.5</td>
<td>C. Huizing, R. Gerth, W.P. de Roever</td>
<td>Full Abstraction of a Real-Time Denotational Semantics for an OCCAM-like Language</td>
</tr>
<tr>
<td>TIR86.6</td>
<td>J. Hooman</td>
<td>A Compositional Proof Theory for Real-Time Distributed Message Passing</td>
</tr>
<tr>
<td>TIR86.7</td>
<td>W.P. de Roever</td>
<td>Questions to Robin Milner - A Responder’s Commentary (IFIP86)</td>
</tr>
<tr>
<td>TIR86.8</td>
<td>R. Gerth, A. Boucher</td>
<td>A Timed Failures Model for Extended Communicating Processes; extended abstract (ICALP87)</td>
</tr>
<tr>
<td>TIR86.9</td>
<td>R. Gerth, W.P. de Roever</td>
<td>Proving Monitors Revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)</td>
</tr>
</tbody>
</table>

- 56 -
In this series appeared:

<table>
<thead>
<tr>
<th>No.</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>85/01</td>
<td>R.H. Mak</td>
<td>The formal specification and derivation of CMOS-circuits</td>
</tr>
<tr>
<td>85/02</td>
<td>W.M.C.J. van Overveld</td>
<td>On arithmetic operations with M-out-of-N-codes</td>
</tr>
<tr>
<td>85/03</td>
<td>W.J.M. Lemmens</td>
<td>Use of a computer for evaluation of flow films</td>
</tr>
<tr>
<td>85/04</td>
<td>T. Verhoeff, H.M.J.L. Schols</td>
<td>Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate</td>
</tr>
<tr>
<td>86/01</td>
<td>R. Koymans</td>
<td>Specifying message passing and real-time systems</td>
</tr>
<tr>
<td>86/02</td>
<td>G.A. Bussing, K.M. van Hee, M. Voorhoeve</td>
<td>ELISA, A language for formal specifications of information systems</td>
</tr>
<tr>
<td>86/03</td>
<td>Rob Hoogerwood</td>
<td>Some reflections on the implementation of trace structures</td>
</tr>
<tr>
<td>86/04</td>
<td>G.J. Houben, J. Paredaens, K.M. van Hee</td>
<td>The partition of an information system in several parallel systems</td>
</tr>
<tr>
<td>86/05</td>
<td>Jan L.G. Dietz, Kees M. van Hee</td>
<td>A framework for the conceptual modeling of discrete dynamic systems</td>
</tr>
<tr>
<td>86/06</td>
<td>Tom Verhoeff</td>
<td>Nondeterminism and divergence created by concealment in CSP</td>
</tr>
<tr>
<td>86/07</td>
<td>R. Gerth, L. Shira</td>
<td>On proving communication closedness of distributed layers</td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>86/09</td>
<td>C. Huizing, R. Gerth, W.P. de Roever</td>
<td>Full abstraction of a real-time denotational semantics for an OCCAM-like language</td>
</tr>
<tr>
<td>86/10</td>
<td>J. Hooman</td>
<td>A compositional proof theory for real-time distributed message passing</td>
</tr>
<tr>
<td>86/11</td>
<td>W.P. de Roever</td>
<td>Questions to Robin Milner - A responders commentary (IFIP86)</td>
</tr>
<tr>
<td>86/12</td>
<td>A. Boucher, R. Gerth</td>
<td>A timed failures model for extended communicating processes</td>
</tr>
<tr>
<td>86/14</td>
<td>R. Koymans</td>
<td>Specifying passing systems requires extending temporal logic</td>
</tr>
<tr>
<td>87/01</td>
<td>R. Gerth</td>
<td>On the existence of a sound and complete axiomatizations of the monitor concept</td>
</tr>
<tr>
<td>87/02</td>
<td>Simon J. Klaver, Chris F.M. Verbene</td>
<td>Federatieve Databases</td>
</tr>
<tr>
<td>87/03</td>
<td>G.J. Houben, J. Paredaens</td>
<td>A formal approach to distributed information systems</td>
</tr>
<tr>
<td>87/04</td>
<td>T. Verhoef</td>
<td>Delay-insensitive codes - An overview</td>
</tr>
<tr>
<td>87/05</td>
<td>R. Kuiper</td>
<td>Enforcing non-determinism via linear time temporal logic specification</td>
</tr>
</tbody>
</table>
87/06  R. Koymans  Temporele logica specificatie van message passing en real-time systemen (in Dutch)

87/07  R. Koymans  Specifying message passing and real-time systems with real-time temporal logic

87/08  H.M.J.L. Schols  The maximum number of states after projecti...a

87/09  J. Kalisvaart  Language extensions to study structures for raster graphics
       L.R.A. Kessener
       W.J.M. Lemmens
       M.L.P van Lierop
       F.J. Peters
       H.M.M. van de Wetering

87/10  T. Verhoeff  Three families of maximally nondeterministic automata

87/11  P. Lemmens  Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model

87/12  K.M. van Hee  OR and AI approaches to decision support systems
       A. Lapinski

87/13  J. van der Woude  Playing with patterns, searching for strings

87/14  J. Hooman  A compositional proof system for an occam-like real-time language

87/15  G. Huizing  A compositional semantics for statecharts
       R. Gerth
       W.P. de Roever

87/16  H.M.M. ten Eikelder  Normal forms for a class of formulas
       J.C.F. Wilmont

87/17  K.M. van Hee  Modelling of discrete dynamic systems framework and examples
       G.J. Houben
       J.L.G. Dietz
<table>
<thead>
<tr>
<th>Year</th>
<th>Name</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>87/18</td>
<td>C.W.A.M. van Overveld</td>
<td>An integer algorithm for rendering curved surfaces</td>
</tr>
<tr>
<td>87/19</td>
<td>A.J. Seebregts</td>
<td>Optimalisering van file allocatie in gedistribueerde database systemen</td>
</tr>
<tr>
<td>87/20</td>
<td>G.J. Houben, J. Paredaens</td>
<td>The $R^2$-Algebra: An extension of an algebra for nested relations</td>
</tr>
<tr>
<td>87/21</td>
<td>R. Gerth, M. Codish, Y. Lichtenstein, E. Shapiro</td>
<td>Fully abstract denotational semantics for concurrent PROLOG</td>
</tr>
<tr>
<td>88/01</td>
<td>T. Verhoeff</td>
<td>A Parallel Program That Generates the Möbius Sequence</td>
</tr>
<tr>
<td>88/02</td>
<td>K.M. van Hee, G.J. Houben, L.J. Somers, M. Voorhoeve</td>
<td>Executable Specification for Information Systems</td>
</tr>
<tr>
<td>88/03</td>
<td>T. Verhoeff</td>
<td>Settling a Question about Pythagorean Triples</td>
</tr>
</tbody>
</table>