A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and completeness

Citation for published version (APA):

Document status and date:
Published: 01/01/1991

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 02. Oct. 2020
Eindhoven University of Technology
Department of Mathematics and Computing Science

A Compositional Proof System for Real-Time Systems Based on Explicit Clock Temporal Logic: Soundness and Completeness

by

P. Zhou  J. Hooman  R. Kuiper

Computing Science Note 91/25
Eindhoven, September 1991
COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science 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.

Copies can be ordered from:
Mrs. F. van Neerven
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

All rights reserved
editors: prof.dr.M.Rem
         prof.dr.K.M.van Hee.
A Compositional Proof System for Real-Time Systems
Based on Explicit Clock Temporal Logic:
Soundness and Completeness *

P. Zhou J. Hooman R. Kuiper

Dept. of Mathematics and Computing Science
Eindhoven University of Technology
P.O.Box 513
5600 MB Eindhoven, The Netherlands

October 8, 1991

Abstract
To specify and verify real-time systems, we consider a real-time version of temporal logic
which is called Explicit Clock Temporal Logic. Timing properties are specified by simply
extending the classical framework of temporal logic with a special variable which explicitly
refers to a global notion of time. Programs are written in an Occam-like real-time language
in which concurrent processes communicate by synchronous message passing along chan­
nels. A proof system is provided to formally verify that a program satisfies a specification
expressed in this logic. The proof system is compositional, that is, the verification of a
complex statement can be done on the basis of the verifications of its components, without
knowing the implementation of them. This allows splitting the verification of large systems
into the verification of subsystems. The proof system is shown to be sound and relatively
complete.

1 Introduction
In this paper we investigate the formal specification and verification of real-time systems. One
of our objectives was that the approach should be able to deal with a reasonably realistic
language. Therefore, we consider an Occam-like real-time programming language [Occ88] with
synchronous message passing along unidirectional channels between concurrent processes. An­
other aim was to build on existing formalisms and enable easy future extension or adaptation.
In general, to specify and verify real-time systems, an existing non-real-time method can be
extended with some notion of time. Here we also follow this approach, and use an extension of
linear time temporal logic [Pnu77,MP82,OL82]. The standard logic enables expressing safety
properties (such as “non-termination”, “no communication along channel c”, and “program
variable x is always greater than 5”) as well as liveness properties (such as “termination”,
eventually communicate along channel c”, and “eventually x has the value 8”). To specify
real-time properties (such as “termination within 7 time units”, “communicate on channel c
at time 4”, and “during the next 3 time units x has a positive value”) we have to extend

*This research was supported by ESPRIT-BRA project 3096 “Formal Methods and Tools for the Development
of Distributed and Real-Time Systems (SPEC)”.

1
temporal logic with a quantitative notion of time. As already observed in [PH88,HLP90], one can distinguish two main approaches.

In one approach, new temporal operators are introduced, by extending the standard ones with time bounds. Already in [BH81] a quantitative “leads to” operator has been introduced to verify real-time applications. In [SPE84] a temporal logic with statements about time intervals has been used to prove correctness of local area network protocols. A general discussion of this extension in the context of linear time, often called Metric Temporal Logic (MTL), can be found in [Koy90]. This logic has been applied to the specification of real-time communication properties of a transmission medium [KVR83]. In a similar way, branching time temporal logic, also called Computation Tree Logic (CTL), is extended to real-time by adding time bounds to the modal operators. See, for instance, [EMSS89] where algorithms for model checking and satisfiability analysis are also presented, for a logic with discrete time. It is shown in [ACD90] that the model checking results can be extended to CTL over a dense time domain.

We investigate the alternative approach, called Explicit Clock Temporal Logic (ECTL), in which temporal logic is extended with a distinguished variable that explicitly refers to clock time. Such a logic has been used in [Ost89] to reason about real-time discrete event systems. A similar real-time version can be found in [Har88]. This approach has the advantage of building upon a well-established classical temporal logic framework to which only axioms and rules for proving timing properties have to be added. Also, further extensions can be envisaged, like extra structure on the time domain, that leave the underlying logic unchanged. It is even possible to leave choices about the time domain, like this being discrete or dense, open till later. This allows for investigating the effect of different choices on, e.g., the possibilities for automatic verification without drastic changes to the formalism.

We mention some works that we feel are relevant to the present approach, though not directly related. Logics for reasoning about real-time systems are classified in [AI90] according to their complexity and expressiveness. A tableau-based decision procedure is given for a version of metric temporal logic. To obtain decidability a discrete time domain is used. Also a decidable version of the explicit clock approach (called TPTL) is considered in which there is a “freezing” quantification that binds a variable to the value of the clock at a certain state. In [HLP90] a decision procedure and a model checking algorithm are given for a suitably restricted version of ECTL. The expressibility of this logic is shown to be incomparable with TPTL.

Given the choice of programming language and the real-time version of temporal logic, the task is to develop a proof method to verify that a program satisfies a property expressed in this logic. In classical verification methods, such as [MP82] for temporal logic, the complete program text must be available. Global proof systems, based on the method of [MP82], and decision procedures for explicit clock versions of temporal logic can be found in [Har88,Ost89].

In contrast with these methods, we formulate a compositional proof system in which the specification of a compound programming language construct (such as sequential composition and parallel composition) can be deduced from specifications for its constituent parts without
any further information about the internal structure of these parts. Compositionality can be considered as a prerequisite for hierarchical, structured, program derivation. A separation of concerns is desired, implying a separation of the use of (and the reasoning about) a module from its implementation. By means of a compositional proof system the design steps can be verified during the process of top-down program construction. To obtain a compositional proof system, our starting point is the compositional system for classical temporal logic as described in [BKP84]. The present paper also builds on the approach to compositionality and real time introduced in [HW89]. With respect to that paper, the main difference is that the emphasis there was on achieving compositionality in as simple a setting as possible. Therefore, a quite sparse model of computation was used. The programming language did not contain data variables and consequently also transmission of values was not incorporated in the model. Furthermore, that paper forms a complement to the present one in the sense that the logic was MTL-styled rather than ECTL-styled. Related compositional proof systems based on MTL for different versions of the programming language have been formulated in [Ho09].

This paper is structured as follows. In Section 2 we describe our programming language, its informal semantics, and the basic timing assumptions. A formal denotational semantics can be found in Section 3. To specify real-time properties of programs we formulate in Section 4 the syntax and the interpretation of our version of ECTL. To verify that a program satisfies a specification written in this logic, Section 5 contains a compositional proof system. The soundness and completeness of the proof system are discussed in Section 6. Conclusion can be found in Section 7. The proofs of lemmas are given in Appendix A. All details of the soundness proof can be found in Appendix B. Precision proof which is used in the completeness proof of Section 6 is given in Appendix C.

2 Real-Time Programming Language

2.1 Syntax and Informal Semantics

We consider a real-time programming language which is akin to Occam [Occ88]. The language is based on a real-time extension of CSP with nested parallelism and synchronous message passing via channels [KSR+88]. A real-time statement delay $e$ is added which suspends an execution for the value of expression $e$ time units. Such a delay-statement may also occur in the guard of a guarded command. Processes communicate and synchronize by message passing via unidirectional channels which connect exactly two processes. Communication is synchronous.

Let $VAR$ be a nonempty set of program variables, $CHAN$ be a nonempty set of channel names, and $VAL$ be a denumerable domain of values of program variables. The syntax of our real-time programming language is given in Table 1, with $c, c_i \in CHAN$, $x, x_i \in VAR$, $\varnothing \in VAL$, and $n \in IN$, where $IN$ denotes the set of all natural numbers.

Define $\text{var}(S)$ as the set of variables occurring in $S$. Let $\text{wvar}(S)$ be the set of all write-variables of $S$, i.e., the set of variables occurring in the left-hand side of an assignment or as a variable in an input statement. Obviously, $\text{wvar}(S) \subseteq \text{var}(S)$. The set of (directional)
Table 1: Syntax of Programming Language

<table>
<thead>
<tr>
<th>Expression</th>
<th>$e ::= \varnothing \mid x \mid e_1 + e_2 \mid e_1 \times e_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean Expression</td>
<td>$b ::= e_1 = e_2 \mid e_1 &lt; e_2 \mid \neg b \mid b_1 \lor b_2$</td>
</tr>
<tr>
<td>Statement</td>
<td>$S ::= \text{skip} \mid x := e \mid \text{delay } e \mid c x \mid c?x \mid S_1 ; S_2 \mid G \mid *G \mid S_1</td>
</tr>
<tr>
<td>Guarded Command</td>
<td>$G ::= [[\bigotimes_{i=1}^n b_i \rightarrow S_i]] \mid [[\bigotimes_{i=1}^n b_i ; c_i?x_i \rightarrow S_i]] [b; \text{delay } e \rightarrow S]$</td>
</tr>
</tbody>
</table>

channels occurring in a statement $S$, denoted by $dch(S)$, is defined as the set containing all channels occurring in $S$ together with all directional channels $c!$ and $c?$ occurring in $S$. For instance, $dch(c!5; d?y|x) = \{c, c!, c?, d, d?\}$.

Informally, the statements have the following meanings:

**Atomic statements**
- **skip** terminates immediately.
- $x := e$ assigns the value of expression $e$ to the variable $x$.
- **delay** $e$ suspends execution for (the value of) $e$ time units.
- $c x$ sends the value of expression $e$ on channel $c$ as soon as a corresponding input statement is available. Since we assume synchronous communication, such an output statement is suspended until a parallel process executes an input statement of the form $c?x$.
- $c?x$ receives a value via channel $c$ and assigns this value to the variable $x$. Similar to the output statement, such an input statement has to wait for a corresponding output statement before a synchronous communication takes place.

**Compound statements**
- $S_1 ; S_2$ indicates sequential composition of $S_1$ and $S_2$.
- Guarded command $[[\bigotimes_{i=1}^n b_i \rightarrow S_i]]$ is executed as follows. If none of the $b_i$ evaluates to true then this command terminates after the evaluation of the booleans. Otherwise, non-deterministically select one of the $b_i$'s which evaluates to true and execute the corresponding statement $S_i$.
- Guarded command $[[\bigotimes_{i=1}^n b_i ; c_i?x_i \rightarrow S_i]] [b; \text{delay } e \rightarrow S]$ is executed as follows. A guard $b_i; c_i?x_i$ or $b; \text{delay } e$ is open if the boolean part evaluates to true. If none of the guards is open, the guarded command terminates after evaluation of the booleans. Otherwise, wait until an io-statement of the open io-guards can be executed and continue with the corresponding $S_i$. If the delay guard is open ($b$ evaluates to true) and no io-guard can be taken within $e$ time units (after the evaluation of the booleans), then $S$ is executed. If $b$ evaluates to true and $e$ yields 0 then $S$ is executed immediately after the evaluation of the booleans. Boolean expressions equivalent to true are often omitted in guards.

**Example 2.1** Observe that delay-values can be arbitrary expressions, for instance, $d?x; [d?y \rightarrow S_1] [c?x \rightarrow S_2] [[\text{delay } (x + 6) \rightarrow S_3]]$.
Example 2.2 By means of a guarded command, we can easily express a time-out. For instance, \([x > 0; c?y → x := y \parallel \text{delay } 10 \rightarrow \text{skip}]\) informally means that if \(x > 0\) and the input communication can take place within 10 time units then the assignment is executed, otherwise after 10 time units there is a time-out and the skip-statement is executed.

- \(\ast G\) indicates repeated execution of guarded command \(G\) as long as one of the guards is open. When none of the guards is open, \(\ast G\) terminates.
- \(S_1 || S_2\) indicates parallel execution of \(S_1\) and \(S_2\). No program variable should occur in both \(S_1\) and \(S_2\), i.e. \(\text{var}(S_1) \cap \text{var}(S_2) = \emptyset\).

Henceforth we use \(\equiv\) to denote syntactic equality.

### 2.2 Basic Timing Assumptions

In order to describe the real-time behavior of programs, we must make assumptions about the execution time of atomic statements and the extra time needed to execute compound constructs. In our proof system, the correctness of a program with respect to a specification, which may express timing properties, is verified relative to these assumptions. For simplicity, we assume in this paper that there is no overhead for compound statements and that a \(\text{delay } e\) statement takes exactly \(e\) time units. Furthermore we assume given positive constants \(K_a, K_c\) and \(K_g\) such that every assignment takes \(K_a\) time units, each communication takes \(K_c\) time units, and the evaluation of the guards in a guarded command takes \(K_g\) time units.

The most important assumption involves parallel composition. Observe that to determine the execution time of a simple program \(x := 0 || y := 1\) we need information about the number of available processors and the allocation of processes on processors. Note that an execution of \(x := 0 || y := 1\) terminates after \(K_a\) time units if both processes \(x := 0\) and \(y := 1\) have their own processor. If, however, the two processes are executed on a single processor then this program will take at least \(2 \times K_a\) time units, since then the processes have to be scheduled in some order. In general, we have to make an assumption about the assignment of processes to processors at parallel composition. In this paper we use the \(\text{maximal parallelism}\) model to represent the situation that each process has its own processor. Consequently any action is executed as soon as possible. Observe that maximal parallelism implies \(\text{minimal waiting}\): a process only waits when it tries to execute an input or output statement and the communication partner is not available. Hence it is never the case that one process waits to perform \(c!e\) and, simultaneously, another process waits to execute \(c?x\). This maximal parallelism assumption has been generalized in \([\text{Hoo91a}]\) to multiprogramming where several processes can be executed on a single processor and scheduling is based on priorities which are assigned to statements in the program.

### 3 Denotational Semantics

To formally define the meaning of a program, we give a denotational semantics for our programming language. In Section 3.1 we define a model to describe computations of programs.
This semantic model is used in Section 4 to interpret our assertion language. In Section 3.2 we present the denotational semantics which is used to define validity of correctness formulas, that is, to define formally when a program satisfies an assertion. Finally, in Section 3.3 we discuss some properties of the semantics.

3.1 Computational Model

In our semantics the timing behavior of a program is expressed from the viewpoint of an external observer with his own clock. Let this clock range over a time domain \( TIME \). Thus, although parallel components of a system might have their own, physical, local clock, the observable behavior of a system is described in terms of a single, conceptual, global clock.

To define the timing behaviour of a statement \( \text{delay } e \), we have to relate expressions in the programming language to our time domain. For simplicity we have assumed that \( \text{delay } e \) takes \( e \) time units and hence, implicitly, that \( VAL \subseteq TIME \). Furthermore we assume that the standard operators \( +, -, \times \) and \( \leq \) are defined on \( TIME \). Note that we allow our time domain to be dense (a domain is dense if between every two points there exists a third point).

Henceforth, we use \( i, j, \ldots \) to denote non-negative integers, and \( \tau, \hat{\tau}, \tau_0, \ldots \) to denote values from \( TIME \). For notational convenience, we use a special value \( \infty \) with the usual properties, such as \( \infty \notin TIME \), and for all \( \tau \in TIME \cup \{\infty\} : \tau \leq \infty, \tau + \infty = \infty + \tau = \infty \), etc.

A computation of a program is represented by a mapping which assigns to each point of time during this computation a pair consisting of a state and a set of communication records. The state represents the values of the program variables at that point of time. The communication records denote the state of affairs on the channels of the program. To denote the real-time communication behaviour, we use records of the form \((c, \vartheta)\), indicating a communication along channel \( c \) with value \( \vartheta \). In addition to the communications at any point of time, the model includes information about those processes waiting to send or waiting to receive messages on their channels at any given time. (The need for this additional information in a compositional framework follows from the fact that this information is present in the fully abstract semantics given in [HGR87] for a similar language.) Using this information, the formalism enforces minimal waiting in our maximal parallelism model by requiring that no pair of processes is ever simultaneously waiting to send and waiting to receive, respectively, on a shared channel.

The informal description above is formalized in the following definitions.

Definition 3.1 (States) The set of states \( STATE \) is defined as the set of mappings from \( VAR \) to \( VAL \), \( STATE = \{ s \mid s : VAR \rightarrow VAL \} \).

Thus a state \( s \in STATE \) assigns to each program variable \( x \) a value \( s(x) \).

Definition 3.2 (Variant) The variant of a state \( s \) with respect to a variable \( x \) and a value \( \vartheta \), denoted by \((s : x \mapsto \vartheta)\), is defined as \((s : x \mapsto \vartheta)(y) = \begin{cases} \vartheta & \text{if } y = x \\ s(y) & \text{if } y \neq x \end{cases}\).
Definition 3.3 (Communication Records) The set of communication records $COMM$ is defined as:

$$COMM = \{c! | c \in CHAN \} \cup \{c? | c \in CHAN \} \cup \{(c, \vartheta) | c \in CHAN \text{ and } \vartheta \in VAL \}.$$  

Let $[\tau_0, \tau_1]$ denote a closed interval of time points: $[\tau_0, \tau_1] = \{ \tau \in TIME | \tau_0 \leq \tau \leq \tau_1 \}$. Then a model, representing a real-time computation of a program, is defined as follows:

Definition 3.4 (Model) Let $\tau_0 \in TIME$, $\tau_1 \in TIME \cup \{ \infty \}$, and $\tau_1 \geq \tau_0$. A model $\sigma$ is a mapping $\sigma : [\tau_0, \tau_1] \to STATE \times \wp(COMM)$.

Definition 3.5 (Begin/End) For a model $\sigma$ with domain $[\tau_0, \tau_1]$, define $begin(\sigma) = \tau_0$ and $end(\sigma) = \tau_1$.

Consider a model $\sigma$ and a point $\tau$ with $begin(\sigma) \leq \tau \leq end(\sigma)$. Then $\sigma(\tau) = (state, comm)$ with $state \in STATE$, and $comm \subseteq COMM$. Henceforth we refer to the two fields of $\sigma(\tau)$ by $\sigma(\tau).state$ and $\sigma(\tau).comm$, respectively. Informally, if $\sigma$ models a computation of a program $S$, $begin(\sigma)$ and $end(\sigma)$ denote, resp., the starting time and the termination time of this computation of $S$ ($end(\sigma) = \infty$ if $S$ does not terminate). Furthermore, $\sigma(begin(\sigma)).state$ specifies the initial state of the computation, and if $end(\sigma) < \infty$ then $\sigma(end(\sigma)).state$ gives the final state. In general, $\sigma(\tau).state$ represents the values of program variables. For a channel name $c$, the set $\sigma(\tau).comm$ might contain a communication record $(c, iI)$, $c!$ or $c?$ with the following meaning:

- $(c, iI) \in \sigma(\tau).comm$ if the value $iI$ is transmitted along channel $c$ at time $\tau$;
- $c! \in \sigma(\tau).comm$ if $S$ is waiting to send along channel $c$ at time $\tau$;
- $c? \in \sigma(\tau).comm$ if $S$ is waiting to receive along channel $c$ at time $\tau$.

To make the model convenient for sequential composition, we do not look at the $comm$ field at the last point. Only $\sigma(end(\sigma)).state$ is interesting for the specification and reasoning. $\sigma(end(\sigma)).comm$ will have an arbitrary value.

Henceforth, we use the following definitions.

Definition 3.6 (Channels Occurring in Model) The set of (directional) channels occurring in a model $\sigma$, denoted by $dch(\sigma)$, is defined as

$$dch(\sigma) = \bigcup_{begin(\sigma) \leq \tau < end(\sigma)} \{c! | c \in \sigma(\tau).comm \} \cup \{c? | c \in \sigma(\tau).comm \} \cup \{c | \text{there exists a } \vartheta \text{ such that } (c, \vartheta) \in \sigma(\tau).comm \}.$$  

Definition 3.7 (Projection onto Channels) Let $cset \subseteq DCHAN$. Define the projection of a model $\sigma$ onto $cset$, denoted by $[\sigma]_{cset}$, as follows: $begin([\sigma]_{cset}) = begin(\sigma)$, $end([\sigma]_{cset}) = end(\sigma)$, for all $\tau$, $begin(\sigma) \leq \tau \leq end(\sigma)$, $[\sigma]_{cset}(\tau).state = \sigma(\tau).state$, and

$$[\sigma]_{cset}(\tau).comm = \{c! | c! \in \sigma(\tau).comm \land c! \in cset \} \cup \{c? | c? \in \sigma(\tau).comm \land c? \in cset \} \cup \{(c, \vartheta) | (c, \vartheta) \in \sigma(\tau).comm \land c \in cset \}.$$
Definition 3.8 (Projection onto Variables) Let \( \text{vset} \subseteq \text{VAR} \). Define the projection of a model \( \sigma \) onto \( \text{vset} \), denoted by \( \sigma \uparrow \text{vset} \), as follows: 
\[
\text{begin}(\sigma \uparrow \text{vset}) = \text{begin}(\sigma),
\]
\[
\text{end}(\sigma \uparrow \text{vset}) = \text{end}(\sigma),
\]
for all \( \tau \), \( \text{begin}(\sigma) \leq \tau \leq \text{end}(\sigma) \), \( (\sigma \uparrow \text{vset})(\tau).\text{comm} = \sigma(\tau).\text{comm} \), for all \( x \in \text{VAR} \), \( (\sigma \uparrow \text{vset})(\tau).\text{state}(x) = \begin{cases} 
\sigma(\tau).\text{state}(x), & \text{if } x \in \text{vset} \\
\sigma(\text{begin}(\sigma)).\text{state}(x), & \text{if } x \notin \text{vset} 
\end{cases} \).

Definition 3.9 (Concatenation) The concatenation of any two models \( \sigma_1 \) and \( \sigma_2 \), denoted by \( \sigma_1 \sigma_2 \), is a model \( \sigma \) such that
- if \( \text{end}(\sigma_1) = \infty \), then \( \sigma = \sigma_1 \);
- if \( \text{end}(\sigma_1) < \infty \), \( \text{end}(\sigma_1) = \text{begin}(\sigma_2) \), and \( \sigma_1(\text{end}(\sigma_1)).\text{state} = \sigma_2(\text{begin}(\sigma_2)).\text{state} \), then \( \sigma \) has domain \([\text{begin}(\sigma_1), \text{end}(\sigma_2)]\) and is defined as follows:
\[
\sigma(\tau) = \begin{cases} 
\sigma_1(\tau), & \text{for all } \tau, \text{begin}(\sigma_1) \leq \tau < \text{end}(\sigma_1) \\
\sigma_2(\tau), & \text{for all } \tau, \text{begin}(\sigma_2) \leq \tau \leq \text{end}(\sigma_2). 
\end{cases}
\]
- otherwise undefined.

Definition 3.10 (Concatenation of Sets of Models) The concatenation of any two sets of models \( \Sigma_1 \) and \( \Sigma_2 \) are defined as follows:
\[
\text{SEQ}(\Sigma_1, \Sigma_2) = \{ \sigma_1 \sigma_2 \mid \sigma_1 \in \Sigma_1 \text{ and } \sigma_2 \in \Sigma_2 \text{ such that } \sigma_1 \sigma_2 \text{ is defined} \}
\]

It is easy to prove that \( \text{SEQ} \) is associative.

3.2 Formal Semantics

A good starting point for the development of a compositional proof system is the formulation of a denotational, and hence compositional, semantics. In such a semantics the meaning of a statement must be defined without any information about the environment in which it will be placed. Hence, the semantics of a statement in isolation must characterize all potential computations of the statement. When composing this statement with (part of) its environment, the semantic operators must remove the computations that are no longer possible. To be able to select the correct computations from the semantics, any dependency of an execution on the environment must be made explicit in the semantic model.

To define such a compositional semantics, we first define the evaluation of expressions and boolean expressions in the programming language.

The evaluation of an expression \( e \), denoted by \( \mathcal{E}(e) \), is a function \( \mathcal{E}(e) : \text{STATE} \rightarrow \text{VAL} \), such that
- \( \mathcal{E}(\varnothing)(s) = \varnothing \),
- \( \mathcal{E}(x)(s) = s(x) \),
- \( \mathcal{E}(e_1 + e_2)(s) = \mathcal{E}(e_1)(s) + \mathcal{E}(e_2)(s) \),
- \( \mathcal{E}(e_1 \times e_2)(s) = \mathcal{E}(e_1)(s) \times \mathcal{E}(e_2)(s) \).

We define when a boolean expression \( b \) holds in a state \( s \), denoted by \( \mathcal{B}(b)(s) \), as follows:
- \( \mathcal{B}(e_1 = e_2)(s) \) iff \( \mathcal{E}(e_1)(s) = \mathcal{E}(e_2)(s) \),
The meaning of a program $S$, denoted by $\mathcal{M}(S)$, is a set of models representing the possible computations of $S$ starting at any arbitrary time.

**Skip**

A skip-statement terminates immediately without any state change.

$$\mathcal{M}(\text{skip}) = \{ \sigma | \text{begin}(\sigma) = \text{end}(\sigma) \}$$

**Assignment**

An assignment $x := e$ terminates after $K_a$ time units (recall that every assignment statement takes $K_a$ time units to execute). All intermediate states before termination are the same as the initial state. The state at termination also equals to the initial state except that the value of $x$ is replaced by the value of $e$ in the initial state. The $\text{comm}$ field is empty during the execution period since the assignment does not (try to) communicate.

$$\mathcal{M}(x := e) = \{ \sigma | \text{end}(\sigma) = \text{begin}(\sigma) + K_a, \text{for all } \tau, \text{begin}(\sigma) \leq \tau < \text{end}(\sigma), \text{begin}(\sigma).state = \text{end}(\sigma).state, \sigma(\tau).comm = \emptyset, \text{and} \sigma(\text{end}(\sigma)).state = (\sigma(\text{begin}(\sigma)).state : x \mapsto E(e)(\sigma(\text{begin}(\sigma)).state)) \}$$

**Delay $e$**

A delay $e$ statement terminates after exactly (the value of) $e$ time units.

$$\mathcal{M}(\text{delay } e) = \{ \sigma | \text{end}(\sigma) = \text{begin}(\sigma) + E(e)(\text{begin}(\sigma)).state, \text{for all } \tau, \text{begin}(\sigma) \leq \tau < \text{end}(\sigma), \sigma(\tau).state = \text{end}(\sigma).state, \sigma(\tau).comm = \emptyset, \text{and} \sigma(\text{end}(\sigma)).state = \sigma(\text{begin}(\sigma)).state \}$$

**Input and Output**

Observe that in the execution of an io-statement there are, in general, two time-periods: first there is a waiting period during which no communication partner is available (recall that communication is synchronous) and, secondly, when such a partner is ready to communicate, there is a period (of $K_c$ time units) during which the actual communication takes place. For an output command $c!e$ these two periods are represented by two sets of models $\text{Wait}(c!)$ and $\text{Send}(c,e)$ defined below. Then the semantics of $c!e$ is defined as follows.

$$\mathcal{M}(c!e) = SEQ(\text{Wait}(c!), \text{Send}(c, e))$$

with

$$\text{Wait}(c!) = \{ \sigma | \text{there exists } \tau \in TIME \cup \{ \infty \} \text{ such that } \text{end}(\sigma) = \tau, \text{for all } \tau_1, \text{begin}(\sigma) \leq \tau_1 < \text{end}(\sigma), \sigma(\tau_1).state = \text{end}(\sigma).state, \sigma(\tau_1).comm = \{ c! \},$$
$\text{Send}(c, e) = \{ \sigma \mid \text{end}(\sigma) = \text{begin}(\sigma) + K_c, \text{ for all } \tau_1, \text{begin}(\sigma) \leq \tau_1 < \text{end}(\sigma), \text{ and } \sigma(\tau_1).\text{state} = \sigma(\text{begin}(\sigma)).\text{state} \}$

$\text{comm} = \{(c, e) (\sigma(\text{begin}(\sigma)).\text{state})\}$

$\text{and } \sigma(\text{end}(\sigma)).\text{state} = \sigma(\text{begin}(\sigma)).\text{state}$

To represent all potential computations of an input statement $c?x$, the semantics contains a model for every possible value that can be received. The value of $x$ in the final state equals the value in the communication record.

$\mathcal{M}(c?x) = \text{SEQ}(\text{Wait}(c?), \text{Receive}(c, x))$ where $\text{Wait}(c?)$ is defined similar to $\text{Wait}(c!)$, and

$\text{Receive}(c, x) = \{ \sigma \mid \text{end}(\sigma) = \text{begin}(\sigma) + K_c, \text{there exists a value } \vartheta \in \text{VAL} \text{ such that, } \text{for all } \tau_1, \text{begin}(\sigma) \leq \tau_1 < \text{end}(\sigma), \sigma(\tau_1).\text{state} = \sigma(\text{begin}(\sigma)).\text{state}, \text{ and } \sigma(\text{end}(\sigma)).\text{state} = (\sigma(\text{begin}(\sigma)).\text{state} : x \mapsto \vartheta) \}$

**Sequential Composition**

Using the $\text{SEQ}$ operator defined before, sequential composition is straightforward:

$\mathcal{M}(S_1; S_2) = \text{SEQ}(\mathcal{M}(S_1), \mathcal{M}(S_2))$

Since $\text{SEQ}$ is associative, the sequential composition is also associative.

**Guarded Command**

For a guarded command $G$, first define

$b_G = \begin{cases} 
\bigvee_{i=1}^n b_i & \text{if } G = [\bigvee_{i=1}^n b_i \rightarrow S_i] \\
\bigvee_{i=1}^n b_i \lor b & \text{if } G = [\bigvee_{i=1}^n (b_i; c_i?x_i, \rightarrow S_i; \text{delay } e \rightarrow S)] 
\end{cases}$

Consider $G = [\bigvee_{i=1}^n b_i \rightarrow S_i]$. Then there are two possibilities: either none of the booleans evaluates to true and the command terminates after $K_g$ time units, or at least one of the booleans yields true and then the corresponding statement $S_i$ is executed. Recall that the evaluation of the guards takes $K_g$ time units. In the semantics below this is represented by statement $\text{delay } K_g$.

$\mathcal{M}([\bigvee_{i=1}^n b_i \rightarrow S_i]) = \{ \sigma \mid B(-b_G)(\sigma(\text{begin}(\sigma)).\text{state}), \text{ and } \sigma \in \mathcal{M}(\text{delay } K_g) \} \cup \{ \sigma \mid \text{there exists a } k, 1 \leq k \leq n, \text{ such that } B(b_k)(\sigma(\text{begin}(\sigma)).\text{state}), \text{ and } \sigma \in \mathcal{M}(\text{delay } K_g; S_k) \}$

Next, consider $G = [\bigvee_{i=1}^n (b_i; c_i?x_i, \rightarrow S_i; \text{delay } e \rightarrow S)]$.

We should consider the following four possibilities:

- None of the booleans evaluates to true and then the guarded command terminates after $K_g$ time units.
- $b$ evaluates to true and at least one of the $c_i?x_i$ for which $b_i$ evaluates to true can perform the communication within $e$ time units after the evaluation of the booleans, then the corresponding $S_i$ is executed.

10
• \(b\) evaluates to true and none of the open io-guards can perform a communication within \(e\) time units after the evaluation of the booleans, then \(S\) is executed.

• \(b\) evaluates to false, then the command waits to communicate on those \(c_i\) whose guards are open.

This leads to the following definition:

\[
\mathcal{M}([\prod_{i=1}^{n} b_i; c_i?x_i \rightarrow S_i; \neg b; \text{delay } e \rightarrow S]) = \\
\{ \sigma \mid B(\neg b_G)\sigma(\text{begin}(\sigma)).\text{state}, \text{ and } \sigma \in \mathcal{M}\text{(delay } K_g)\}\ \cup \\
\text{SEQ}(\mathcal{M}(\text{delay } K_g), \text{BoundWait}(G), \text{Comm}(G)) \cup \\
\text{SEQ}(\mathcal{M}(\text{delay } K_g), \text{TimeOut}(G), \mathcal{M}(S)) \cup \\
\text{SEQ}(\mathcal{M}(\text{delay } K_g), \text{AnyWait}(G), \text{Comm}(G))
\]

where

\[\text{Wait}(G) = \{ \sigma \mid B(b_G)\sigma(\text{begin}(\sigma)).\text{state}, \text{ and exists a } \tau \in \text{TIME} \cup \{\infty\} \text{ such that}
\]

\[\text{for all } \tau_i, \text{begin}(\sigma) \leq \tau_i < \text{end}(\sigma), \sigma(\tau_i).\text{state} = \sigma(\text{begin}(\sigma)).\text{state},
\]

\[\sigma(\tau_i).\text{comm} = \{c_i? \mid B(b_i)\sigma(\text{begin}(\sigma)).\text{state}, 1 \leq i \leq n\}, \text{ and } \sigma(\text{end}(\sigma)).\text{state} = \sigma(\text{begin}(\sigma)).\text{state}, \text{ and end}(\sigma) = \tau\}
\]

\[\text{Comm}(G) = \{ \sigma \mid \text{there exists a } k, 1 \leq k \leq n, \text{ such that } B(b_k)\sigma(\text{begin}(\sigma)).\text{state}, \text{ and } \sigma \in \text{SEQ}(\text{Receive}(c_k, z_k), \mathcal{M}(S_k))\}\}
\]

\[\text{BoundWait}(G) = \{ \sigma \mid \sigma \in \text{Wait}(G), B(b_G)\sigma(\text{begin}(\sigma)).\text{state}, \text{ and } \text{end}(\sigma) < \text{begin}(\sigma) + \mathcal{E}(e)(\sigma(\text{begin}(\sigma)).\text{state})\}\}
\]

\[\text{TimeOut}(G) = \{ \sigma \mid \sigma \in \text{Wait}(G), B(b_G)\sigma(\text{begin}(\sigma)).\text{state}, \text{ and } \text{end}(\sigma) = \text{begin}(\sigma) + \mathcal{E}(e)(\sigma(\text{begin}(\sigma)).\text{state})\}\}
\]

\[\text{AnyWait}(G) = \{ \sigma \mid \sigma \in \text{Wait}(G) \text{ and } B(\neg b_G)\sigma(\text{begin}(\sigma)).\text{state}\}\}
\]

Iteration

For a model in the semantics of the iteration construct \(\ast G\) we have the following possibilities:

• Either it is the concatenation of a finite sequence of models from \(\mathcal{M}(G)\) such that the last model corresponds to an execution where all boolean guards evaluate to false or it represents a non-terminating computation of \(G\).

• Or it is the concatenation of an infinite sequence of models from \(\mathcal{M}(G)\) that all represent terminating computations in which not all booleans yield the value false.

This leads to the following definition:

\[\mathcal{M}(\ast G) = \{ \sigma \mid \text{there exist a } k \in \mathbb{N}, k \geq 1, \text{ and } \sigma_1, \ldots, \sigma_k \text{ such that } \sigma = \sigma_1 \cdots \sigma_k,
\]

\[\text{for all } i, 1 \leq i \leq k, \sigma_i \in \mathcal{M}(G), \text{ for all } j, 1 \leq j \leq k - 1, \text{ end}(\sigma_j) < \infty,
\]

\[B(b_G)\sigma_j(\text{begin}(\sigma_j)).\text{state}, \text{ and either end}(\sigma_k) = \infty \text{ or } B(\neg b_G)\sigma_k(\text{begin}(\sigma_k)).\text{state}\}\}

\[\cup \{ \sigma \mid \text{there exists an infinite sequence of models } \sigma_1, \sigma_2, \ldots \text{ such that } \sigma = \sigma_1 \sigma_2 \cdots,
\]

\[\text{for all } i \geq 1, \sigma_i \in \mathcal{M}(G), \text{ end}(\sigma_i) < \infty, \text{ and } B(b_G)\sigma_i(\text{begin}(\sigma_i)).\text{state}\}\}
Parallel Composition

The semantics of $S_1\|S_2$ consists of all models $\sigma$ such that there exist models $\sigma_1 \in \mathcal{M}(S_1)$ and $\sigma_2 \in \mathcal{M}(S_2)$ such that the $comm$-field of $\sigma$ is the point-wise union of the $comm$-fields of $\sigma_1$ and $\sigma_2$, provided the following requirements are fulfilled:

1. $\text{end}(\sigma) = \max(\text{end}(\sigma_1), \text{end}(\sigma_2))$, to express that $S_1\|S_2$ terminates when both processes have terminated.
2. Since communication is synchronous, $S_1$ and $S_2$ should communicate simultaneously on shared channels which connect the two processes.
3. In our execution model we assume minimal waiting for communications, i.e. two processes should not be simultaneously waiting to send and waiting to receive on a shared channel. Formally, for all $\tau$, $\text{begin}(\sigma) \leq \tau < \text{end}(\sigma)$ and for all $c \in dch(S_1) \cap dch(S_2)$:
   $$-(c! \in \sigma(\tau).comm \land c? \in \sigma(\tau).comm).$$

For the $state$-field of $\sigma$, recall that there are no shared variables, i.e. $\text{var}(S_1) \cap \text{var}(S_2) = \emptyset$. Hence the value of a variable $x$ during the execution of $S_1\|S_2$ can be obtained from the state of $S_i$ if $x \in \text{var}(S_i)$, and from the initial state otherwise. This leads to the following definition for the semantics of parallel composition.

$$\mathcal{M}(S_1\|S_2) = \{ \sigma \mid dch(\sigma) \subseteq dch(S_1) \cup dch(S_2), \text{ for } i = 1, 2, \text{ there exist } \sigma_i \in \mathcal{M}(S_i) \text{ such that } \text{begin}(\sigma) = \text{begin}(\sigma_1) = \text{begin}(\sigma_2), \text{ end}(\sigma) = \max(\text{end}(\sigma_1), \text{end}(\sigma_2)), \}

[\sigma]_{dch(S_i)}(\tau).comm = \begin{cases} \sigma_i(\tau).comm & \text{begin}(\sigma_i) \leq \tau < \text{end}(\sigma_i) \\ \emptyset & \text{end}(\sigma_i) \leq \tau < \text{end}(\sigma) \end{cases}

(\sigma \uparrow \text{var}(S_i))(\tau).state = \begin{cases} \sigma_i(\tau).state & \text{begin}(\sigma_i) \leq \tau \leq \text{end}(\sigma_i) \\ \sigma_i(\text{end}(\sigma_i)).state & \text{end}(\sigma_i) < \tau \leq \text{end}(\sigma) \end{cases}

\text{and for all } c \in dch(S_1) \cap dch(S_2), \text{ for all } \tau, \text{ begin}(\sigma) \leq \tau < \text{end}(\sigma),

$$-(c! \in \sigma(\tau).comm \land c? \in \sigma(\tau).comm).$$

We can prove that Parallel composition is commutative and associative.

3.3 Properties of the Semantics

First we define three well-formedness properties.

Definition 3.11 (Well-Formedness) A model $\sigma$ is well-formed iff for all $c \in \text{CHAN}$, for all $\vartheta, \vartheta_1, \vartheta_2 \in \text{VAL}$, and for all $\tau$, $\text{begin}(\sigma) \leq \tau < \text{end}(\sigma)$:

1. $-(c! \in \sigma(\tau).comm \land c? \in \sigma(\tau).comm).$
   (Minimal waiting: It is not allowed to be simultaneously waiting to send and waiting to receive on a particular channel.)
2. $-((c, \vartheta) \in \sigma(\tau).comm \land c! \in \sigma(\tau).comm)$ and $-((c, \vartheta) \in \sigma(\tau).comm \land c? \in \sigma(\tau).comm).$
   (Exclusion: It is not allowed to be simultaneously communicating and waiting to communicate on a given channel.)
3. \((c, \vartheta_1) \in \sigma(\tau).comm \land (c, \vartheta_2) \in \sigma(\tau).comm \Rightarrow \vartheta_1 = \vartheta_2.\)

(Uniqueness: At most one value is transmitted on a particular channel at any point of time.)

Then we have the following theorem.

**Theorem 3.12** For any program \(S\), if \(\sigma \in \mathcal{M}(S)\) then

1. \(dch(\sigma) \subseteq dch(S)\), and
2. \(\sigma\) is well-formed.

By induction on the structure of \(S\) and the definition of well-formedness, we can easily prove this theorem.

### 4 Assertion Language

We define an assertion language which is based on ordinary linear temporal logic augmented with a global clock variable denoted by \(T\). Intuitively, \(T\) refers to the current point of time during execution. We use \textit{start} to express the starting time of a computation and \textit{term} to denote the termination time of a computation (\(\text{term} = \infty\) for a non-terminating computation). We also use \textit{last}(\(x\)) to refer to the value of program variable \(x\) at the last state of a computation model. To specify the communication behaviour of programs, we use a primitive \textit{comm}(\(c, \textit{exp}\)) to express a communication along channel \(c\) with value \(\textit{exp}\). We also use \textit{comm}(\(c\)) to abstract from the value communicated. Furthermore, the assertion language includes primitives \textit{wait}(\(c!\)) and \textit{wait}(\(c?\)) to denote that processes are waiting to communicate. Similar to the semantics, this is required to express maximal parallelism. By including the strong until operator, \(U\), from classical temporal logic we obtain the standard modal operators. In order to give compositional proof rules for sequential composition and iteration, we add the "combine" operator \(C\) and the "iterated combine" operator \(C^*\) from [BKP84]. To express properties of program variables we have expressions of type \(\text{VAL}\). We also have expressions of type \(\text{TIME}\) to allow references to points of time.

In addition to program variables and time variables, specifications may also use so-called logical variables which are not affected by program execution. These logical variables can be used to record the value of a program variable or time variable. Let \(\text{VVAR}\) be a set of logical variables ranging over \(\text{VAL}\), and \(\text{TVAR}\) be a set of logical variables ranging over \(\text{TIME} \cup \{\infty\}\). The syntax of this specification language is given in Table 2, with \(\vartheta \in \text{VAL}\), \(v \in \text{VVAR}\), \(x \in \text{VAR}\), \(\tau \in \text{TIME} \cup \{\infty\}\), \(t \in \text{TVAR}\), and \(c \in \text{CHAN}\).

Let \(dch(\varphi)\) denote the set of all \(c, c!\), or \(c?\) occurring in \(\varphi\), and \(\text{var}(\varphi)\) denote the set of all program variables in \(\varphi\).

To interpret logical variables we use a logical variable environment \(\gamma\), which is a mapping which assigns a value from \(\text{VAL}\) to each logical variable \(v \in \text{VVAR}\) and a value from \(\text{TIME} \cup \{\infty\}\) to each logical variable \(t \in \text{TVAR}\). The values of \(v\) and \(t\) in an environment \(\gamma\) are denoted by \(\gamma(v)\) and \(\gamma(t)\), respectively.
Table 2: Syntax of Assertion Language

| Val Expression | vexp ::= | θ | v | x | last(x) | vexp1 + vexp2 | vexp1 × vexp2 |
|----------------|---------|---|---|------|----------------|----------------|
| Time Expression | texp ::= | † | t | T | start | term | vexp |
| | | texp1 + texp2 | texp1 - texp2 | texp1 × texp2 |
| Assertion | φ ::= | texp1 = texp2 | texp1 < texp2 |
| | | comm(c,vexp) | comm(c) | wait(c!) | wait(c?) | φ1 ∨ φ2 | ¬φ | φ1 U φ2 | φ1 C φ2 | φ1 C* φ2 |

The interpretation of assertions is defined over the computational model of section 3.1. First we define the value of expression \( vexp \) in a model \( a \) at time \( T \) as:

- \( V(\emptyset)\gamma(\sigma, T) = \emptyset \)
- \( V(\psi)\gamma(\sigma, T) = \gamma(\psi) \)
- \( V(x)\gamma(\sigma, T) = \begin{cases} \sigma(\tau).state(x) & \text{if } \tau \leq end(\sigma) \\ \sigma(end(\sigma)).state(x) & \text{if } \tau > end(\sigma) \end{cases} \)
- \( V(last(x))\gamma(\sigma, T) = \sigma(end(\sigma)).state(x) \)
- \( V(vexp1 + vexp2)\gamma(\sigma, T) = V(vexp1)\gamma(\sigma, T) + V(vexp2)\gamma(\sigma, T) \)
- \( V(vexp1 × vexp2)\gamma(\sigma, T) = V(vexp1)\gamma(\sigma, T) × V(vexp2)\gamma(\sigma, T) \)

Next we define the value of time-expression \( texp \) in a model \( a \) at time \( T \) as:

- \( T(\hat{t})\gamma(\sigma, T) = \hat{t} \)
- \( T(t)\gamma(\sigma, T) = \gamma(t) \)
- \( T(T)\gamma(\sigma, T) = \tau \)
- \( T(start)\gamma(\sigma, T) = begin(\sigma) \)
- \( T(term)\gamma(\sigma, T) = end(\sigma) \)
- \( T(vexp)\gamma(\sigma, T) = V(vexp)\gamma(\sigma, T) \)
- \( T(texp1 + texp2)\gamma(\sigma, T) = T(texp1)\gamma(\sigma, T) + T(texp2)\gamma(\sigma, T) \)
- \( T(texp1 - texp2)\gamma(\sigma, T) = T(texp1)\gamma(\sigma, T) - T(texp2)\gamma(\sigma, T) \)
- \( T(texp1 × texp2)\gamma(\sigma, T) = T(texp1)\gamma(\sigma, T) × T(texp2)\gamma(\sigma, T) \)

The interpretation of an assertion \( \phi \) at time \( T \geq begin(\sigma) \) in a model \( \sigma \) and an environment \( \gamma \) is denoted by \( \langle \sigma, \gamma \rangle \models \phi \) and defined by induction on the structure of \( \phi \):

- \( \langle \sigma, \gamma \rangle \models texp1 = texp2 \text{ iff } T(texp1)\gamma(\sigma, T) = T(texp2)\gamma(\sigma, T) \).
- \( \langle \sigma, \gamma \rangle \models texp1 < texp2 \text{ iff } T(texp1)\gamma(\sigma, T) < T(texp2)\gamma(\sigma, T) \).
- \( \langle \sigma, \gamma \rangle \models comm(c, vexp) \text{ iff } \tau < end(\sigma) \text{ and } (c, V(vexp)\gamma(\sigma, T)) \in \sigma(\tau).comm. \)
- \( \langle \sigma, \gamma \rangle \models comm(c) \text{ iff } \tau < end(\sigma) \text{ and there exists a value } \emptyset \in VAL \text{ such that } \emptyset \in c(\tau).comm. \)
- \( \langle \sigma, \gamma \rangle \models wait(c!) \text{ iff } \tau < end(\sigma) \text{ and } c! \in \sigma(\tau).comm. \)
- \( \langle \sigma, \gamma \rangle \models wait(c?) \text{ iff } \tau < end(\sigma) \text{ and } c? \in \sigma(\tau).comm. \)
• \((\sigma, \tau) \gamma \models \varphi_1 \lor \varphi_2\) iff \((\sigma, \tau) \gamma \models \varphi_1\) or \((\sigma, \tau) \gamma \models \varphi_2\).

• \((\sigma, \tau) \gamma \models \neg \varphi\) iff not \((\sigma, \tau) \gamma \models \varphi\).

• \((\sigma, \tau) \gamma \models \varphi_1 \lor \varphi_2\) iff there exists a \(\tau_2 \geq \tau\), such that \((\sigma, \tau_2) \gamma \models \varphi_2\), and for all \(\tau_1, \tau \leq \tau_1 < \tau_2\): \((\sigma, \tau_1) \gamma \models \varphi_1\).

• \((\sigma, \tau) \gamma \models \varphi_1 \land \varphi_2\) iff there exist models \(\sigma_1\) and \(\sigma_2\) such that \(\sigma = \sigma_1 \sigma_2\), \(\text{end}(\sigma_1) \geq \tau\), \((\sigma_1, \tau) \gamma \models \varphi_1\), and \((\sigma_2, \text{begin}(\sigma_2)) \gamma \models \varphi_2\).

• \((\sigma, \tau) \gamma \models \neg \varphi_1 \lor \varphi_2\) iff
  - either there exist a \(k \geq 1\) and models \(\sigma_1, \ldots, \sigma_k\) such that \(\sigma = \sigma_1 \cdots \sigma_k\),
    \((\sigma_1, \tau) \gamma \models \varphi_1\), \(\tau \leq \text{end}(\sigma_j) < \infty\), for all \(j, 2 \leq j \leq k - 1\), \((\sigma_j, \text{begin}(\sigma_j)) \gamma \models \varphi_1\), \(\text{end}(\sigma_j) < \infty\), and if \(\text{end}(\sigma_k) < \infty\) then \((\sigma_k, \text{begin}(\sigma_k)) \gamma \models \varphi_2\), otherwise
    \((\sigma_k, \text{begin}(\sigma_k)) \gamma \models \varphi_1\);
  - or there exist infinite models \(\sigma_1, \sigma_2, \sigma_3, \ldots\) such that \(\sigma = \sigma_1 \sigma_2 \sigma_3 \ldots\), \(\text{end}(\sigma_i) \geq \tau\), for all \(i \geq 1\), \(\text{end}(\sigma_i) < \infty\), \((\sigma_1, \tau) \gamma \models \varphi_1\), and for all \(j \geq 2\), \((\sigma_j, \text{begin}(\sigma_j)) \gamma \models \varphi_1\).

We use the standard abbreviations such as \text{true} \equiv 0 = 0, \varphi_1 \land \varphi_2 \equiv \neg(\neg \varphi_1 \lor \neg \varphi_2), \varphi_1 \lor \varphi_2 \equiv \neg \varphi_1 \lor \varphi_2, \text{exp}_1 \leq \text{exp}_2 \equiv \text{exp}_1 = \text{exp}_2 \lor \text{exp}_1 < \text{exp}_2\), etc.

In the proof system we will frequently use the following abbreviations:

• For a finite set \(\text{cset}\) of channels and directional channels, define
  \(\text{empty}(\text{cset}) \equiv \bigwedge_{c \in \text{cset}} \neg \text{wait}(c) \land \bigwedge_{c \in \text{cset}} \neg \text{wait}(c\uparrow) \land \bigwedge_{c \in \text{cset}} \neg \text{comm}(c)\).

• For a finite set \(\text{vset}\) of program variables, define
  \(\text{inv}(\text{vset}) \equiv \bigwedge_{x \in \text{vset}} x = v_x \rightarrow \square (x = v_x)\) where \(v_x\) is a logical variable recording the initial value of \(x\).

Furthermore we have the usual abbreviations from temporal logic:

• \(\Diamond \varphi \equiv \text{true} \lor \varphi\) (eventually \(\varphi\) will be true)

• \(\Box \varphi \equiv \neg \Diamond \neg \varphi\) (henceforth \(\varphi\) will be true)

• \(\varphi_1 \lor \varphi_2 \equiv (\varphi_1 \lor \varphi_2) \lor \Box \varphi_1\) (weak until: either eventually \(\varphi_2\) will hold and until that point \(\varphi_1\) holds continuously, or \(\varphi_1\) holds henceforth)

Next we define validity of assertions and correctness formulas of the form \(S \models \varphi\).

**Definition 4.1 (Valid Assertion)** An assertion \(\varphi\) is valid, denoted by \(\models \varphi\), iff
\((\sigma, \text{begin}(\sigma)) \gamma \models \varphi\) for any model \(\sigma\) and any environment \(\gamma\).

For instance, \(\models T = \text{start}\), \(\models T = t \rightarrow \square (T \geq t)\), \(\models \square (T = \text{term} \land x = v \rightarrow \square x = v)\), and \(\models \square (T = \text{term} \rightarrow \square \text{empty}(\{c\}))\).

**Definition 4.2 (Satisfaction)** A program \(S\) satisfies an assertion \(\varphi\), denoted by
\(\models S \models \varphi\), iff \((\sigma, \text{begin}(\sigma)) \gamma \models \varphi\) for any \(\sigma \in \mathcal{M}(S)\) and any environment \(\gamma\).

We also say that \(S \models \varphi\) is valid if \(\models S \models \varphi\).

Finally we give a few simple examples to illustrate our assertion language. General safety properties can be specified, e.g.,
• Program $S$ does not terminate: $S \text{ sat } \text{term} = \infty$.
  Note that we could also use $S \text{ sat } \Box \neg (T = \text{term})$.
• $S$ does not perform any communication along channel $c$: $S \text{ sat } \Box \neg \text{comm}(c)$.

Some examples of real-time safety properties:
• If $S$ starts its execution with $x = 0$, then $S$ will terminate in less than 5 time units with $x = 8$: $S \text{ sat } x = 0 \rightarrow \text{term} \leq \text{start} + 5 \land \text{last}(x) = 8$.
• If $S$ communicates on channel $c$ then $S$ is waiting to send or communicating on channel $d$ within 25 time units:
  $S \text{ sat } \Box (T = t \land \text{comm}(c) \rightarrow \Diamond (T \leq t + 25 \land (\text{wait}(d!) \lor \text{comm}(d))))$.
  Note that logical variable $t$ is implicitly universally quantified.
• During the execution of $S$, the program variable $x$ has value 5 at 3 time units after the start of the execution, after 5 time units $x$ has value 8 and $y$ has value 9, and finally after 7 time units program $S$ terminates with $x = 10$ and $y = 12$:
  $S \text{ sat } \Box ((T = \text{start} + 3 \rightarrow x = 5) \land (T = \text{start} + 5 \rightarrow x = 8 \land y = 9) \land (T = \text{start} + 7 \rightarrow x = 10 \land y = 12)) \land \text{term} = \text{start} + 7$.

Liveness properties can also be expressed:
• $S$ terminates: $S \text{ sat } \text{term} < \infty$. (Or, equivalently, $S \text{ sat } \Diamond T = \text{term}$.)
• $S$ either communicates along channel $c$ infinitely often or eventually it waits forever to send on $c$: $S \text{ sat } (\Box \Diamond \text{comm}(c)) \lor (\Diamond \Box \text{wait}(c!))$.

5 Proof System

In this section, we give a compositional proof system for our correctness formulae. First we formulate axioms and rules which are generally applicable to any statement. Next we axiomatize the programming language by formulating axioms and rules for all atomic statements and compound programming constructs.

Let $vexp_1$, $vexp_2$ be expressions of type $\text{VAL}$. The well-formedness properties of the semantics models are axiomatized by the following axiom, for any finite $cset \subseteq \text{DCHAN}$:

**Axiom 5.1 (Well-Formedness)** $S \text{ sat } WF_{cset}$

where

$WF_{cset} \equiv \Box (\text{MinWait}_{cset} \land \text{Exclusion}_{cset} \land \text{Unique}_{cset})$

$\text{MinWait}_{cset} \equiv \land_{\langle c!, c? \rangle \subseteq cset} \neg (\text{wait}(c!) \land \text{wait}(c?))$

$\text{Exclusion}_{cset} \equiv \land_{\langle c, c! \rangle \subseteq cset} \neg (\text{comm}(c) \land \text{wait}(c!)) \land \land_{\langle c, c? \rangle \subseteq cset} \neg (\text{comm}(c) \land \text{wait}(c?))$

$\text{Unique}_{cset} \equiv \land_{e \in cset} \text{comm}(c, vexp_1) \land \text{comm}(c, vexp_2) \rightarrow vexp_1 = vexp_2$

The next general axiom expresses that a program does not (try to) communicate on channels that do not syntactically occur in the program. For each finite set $cset \subseteq \text{DCHAN}$, we have the following axiom:

**Axiom 5.2 (Communication Invariance)** $S \text{ sat } \Box \text{empty}(cset)$

provided $cset \cap dch(S) = \emptyset$.  

16
Similarly, the proof system has an axiom to express that certain program variables are not changed by a program. Then the following axiom expresses that if \( x \in \text{wvar}(S) \) then \( x \) will not be changed by \( S \).

**Axiom 5.3 (Variable Invariance)**  
\[ S \text{ sat } \text{inv}(\{x\}) \]
provided \( x \notin \text{wvar}(S) \).

Furthermore, we have the usual conjunction rule and consequence rule.

**Axiom 5.4 (Conjunction)**  
\[ S \text{ sat } \varphi_1, S \text{ sat } \varphi_2 \quad \rightarrow \quad S \text{ sat } \varphi_1 \land \varphi_2 \]

**Axiom 5.5 (Consequence)**  
\[ S \text{ sat } \varphi_1, \varphi_1 \rightarrow \varphi_2 \quad \rightarrow \quad S \text{ sat } \varphi_2 \]

Next we give axioms for five atomic statements. Statement \text{skip} terminates immediately.

**Axiom 5.6 (Skip)**  
\[ \text{skip sat } \text{term} = \text{start} \]

The assignment axiom expresses that \( x := e \) terminates after \( K_a \) time units and that the final value of \( x \) equals the value of \( e \) in the initial state. If \( x \) occurs in the expression \( e \), the initial value of \( x \) is needed to evaluate the value of \( e \). Therefore we use a logical variable \( v \) to record the initial value of \( x \).

**Axiom 5.7 (Assignment)**  
\[ x := e \text{ sat } x = v \rightarrow (x = v \lor \text{term} = \text{start} + K_a \land x = e[v/x]) \]

**Example 5.1** With this axiom and the Consequence Rule we can derive, for instance,  
\[ x := x + 1 \text{ sat } x = v \rightarrow (x = v + 1 \land \text{term} = \text{start} + K_a). \]

**Example 5.2** We show that we can derive  
\[ x := y + 4 \text{ sat } y = 5 \rightarrow (x = y + 4 \land \text{term} = \text{start} + K_a). \]

By the Assignment Axiom and the Consequence Rule we obtain  
\[ x := y + 4 \text{ sat } x = v \rightarrow (x = y + 4 \land \text{term} = \text{start} + K_a). \]

Since \( v \) does not occur in the consequence of this implication, we can substitute \( x \) for \( v \).

By the Consequence Rule, this leads to  
\[ x := y + 4 \text{ sat } (x = y + 4 \land \text{term} = \text{start} + K_a). \]

Since \( y \notin \text{wvar}(x := y + 4) \), we can derive from the Variable Invariance Axiom  
\[ x := y + 4 \text{ sat } y = v \rightarrow \square y = v. \]

Hence, by the Conjunction Rule and the Consequence Rule,  
\[ x := y + 4 \text{ sat } y = v \rightarrow (x = v + 4 \land \text{term} = \text{start} + K_a). \]

Since \( y = v \rightarrow (x = v + 4 \land \text{term} = \text{start} + K_a) \) implies \( y = 5 \rightarrow (x = 9 \land \text{term} = \text{start} + K_a) \), the Consequence Rule leads to  
\[ x := y + 4 \text{ sat } y = 5 \rightarrow (x = 9 \land \text{term} = \text{start} + K_a). \]

Statement \text{delay } e \text{ terminates after exactly (the value of) } e \text{ time units.}
Axiom 5.8 (Delay) \( \text{delay } e \text{ sat } \text{term} = \text{start} + e \)

An output statement starts waiting to send a message, and as soon as a communication partner is available the communication takes place during \( K_c \) time units. Note that we use a weak until operator in the axiom below to allow an infinite waiting period (i.e., deadlock) when no partner becomes available.

Axiom 5.9 (Output) \( c!e \text{ sat } \text{wait}(c!) U (T = \text{term} - K_c \land (\text{comm}(c, e) U T = \text{term})) \)

Similarly, an input statement \( c?x \) waits to receive a value on the channel \( c \). When the communication finishes the value received is assigned to variable \( x \). Thus at the last state of the execution model \( x \) possesses that value.

Axiom 5.10 (Input) \( c?x \text{ sat } x = v \rightarrow [(x = v \land \text{wait}(c?)) U (T = \text{term} - K_c \land ((x = v \land \text{comm}(c, \text{last}(x))) U T = \text{term})]] \)

Using the \( C \) operator we can easily formulate an inference rule for sequential composition.

Rule 5.11 (Sequential Composition Rule) \( \frac{S_1 \text{ sat } \varphi_1, S_2 \text{ sat } \varphi_2}{S_1; S_2 \text{ sat } \varphi_1 C \varphi_2} \)

Example 5.3 Consider the program \( x := x + 1; \ x := x + 2 \). By the Assignment Axiom and the Consequence Rule we can derive:

\[
\begin{align*}
\text{x := x + 1 sat} & \quad x = v_1 \to \Box (T = \text{start} + K_a \to x = v_1 + 1) \land \text{term} = \text{start} + K_a, \\
\text{x := x + 2 sat} & \quad x = v_2 \to \Box (T = \text{start} + K_a \to x = v_2 + 2) \land \text{term} = \text{start} + K_a.
\end{align*}
\]

Then the Sequential Composition Rule and the Consequence Rule lead to

\[
\begin{align*}
\text{x := x + 1; x := x + 2 sat} \\
& \quad x = v \to \Box (T = \text{start} + K_a \to x = v + 1) \land \Box (T = \text{start} + 2 \times K_a \to x = v + 3)
\end{align*}
\]

Note that instead of \( \Box (T = \text{start} + K_a \to x = v + 1) \) we could have used \( \bigodot (T = \text{start} + K_a \land x = v + 1) \).

Now consider a guarded command \( G \). Recall that \( b_G \) is defined as (see section 3.2)

\[
b_G \equiv \left\{ \begin{array}{ll}
\forall_{i=1}^n b_i & \text{ if } G \equiv [\forall_{i=1}^n b_i \to S_i] \\
\forall_{i=1}^n b_i \lor b & \text{ if } G \equiv [\forall_{i=1}^n b_i; c?xi \to S_i \parallel b; \text{delay e} \to S]
\end{array} \right.
\]

We use a logical variable \( v_y \) to refer to the initial value of program variable \( y \) where \( y \in \text{var}(G) \).

Define \( \text{Quiet} \equiv \Lambda_{y \in \text{var}(G)} v_y = v_y \land \text{empty(dch}(G)) \).

First we give an axiom which expresses that if none of the booleans evaluates to true then the guarded command terminates after \( K_g \) time units (during which the booleans are evaluated). Furthermore we express that there is no activity on the channels of \( G \) and no variable of \( G \) is changed during the evaluation of the guards.

Axiom 5.12 (Guarded Command Evaluation)

\[
G \text{ sat } \Lambda_{y \in \text{var}(G)} y = v_y \to (\text{Quiet} U (T = \text{start} + K_g \land \Lambda_{y \in \text{var}(G)} y = v_y)) \land \\
(\neg b_G \to \text{term} = \text{start} + K_g)
\]

18
Now consider a guarded command with purely boolean guards $G ≡ \prod^n_{i=1} b_i → S_i$. If at least one of the booleans yields true then after the evaluation of the booleans (the evaluation period has length $K_g$) one of the statements $S_i$ for which $b_i$ evaluates to true is executed. This leads to the following rule:

**Rule 5.13 (Guarded Command with Purely Boolean Guards)**

$$
\begin{align*}
S_i \text{ sat } \varphi_i, & \text{ for } i = 1, \ldots, n \\
\prod^n_{i=1} b_i → S_i \text{ sat } b_G → (\text{term} = \text{start} + K_g) & \text{ sat } V^n_{i=1} (b_i \land \varphi_i)
\end{align*}
$$

Next we formulate a rule for $G ≡ \prod^n_{i=1} b_i; c_i?x_i → S_i; \prod b_i; \text{delay } e → S'$, using

$$
\begin{align*}
\text{Wait} & ≡ \Lambda y \in \text{var}(G) y = v_y \land \text{empty}(dch(G) - \{c_1?, \ldots, c_n?\}) \land \\
& \quad (b \rightarrow T < \text{start} + e) \land \Lambda^n_{i=1} (b_i \rightarrow \text{wait}(c_i))
\end{align*}
$$

$$
\begin{align*}
\text{InTime} & ≡ \Lambda y \in \text{var}(G) y = v_y \land T = \text{term} \land (b \rightarrow T < \text{start} + e), \\
\text{EndTime} & ≡ \Lambda y \in \text{var}(G) y = v_y \land b \land T = \text{term} = \text{start} + e,
\end{align*}
$$

$$
\begin{align*}
\text{Eval} & ≡ \text{term} = \text{start} + K_g,
\end{align*}
$$

$$
\begin{align*}
\text{Comm} & ≡ (\text{Wait} \cup \text{InTime}) \land V^n_{i=1} (b_i \land \varphi_i \land \text{comm}(c_i)) \land \text{and} \\
\text{TimeOut} & ≡ (\text{Wait} \cup \text{EndTime}) \land \varphi
\end{align*}
$$

**Rule 5.14 (Guarded Command with IO-guards)**

$$
\begin{align*}
c_i?x_i; S_i \text{ sat } \varphi_i, & \text{ for } i = 1, \ldots, n, \quad S \text{ sat } \varphi \\
\prod^n_{i=1} b_i; c_i?x_i → S_i; \prod b_i; \text{delay } e → S\text{ sat } & \text{ Eval } \land (\text{Comm} \lor \text{TimeOut})
\end{align*}
$$

The inference rule for an iterated guarded command is:

**Rule 5.15 (Iteration)**

$$
G \text{ sat } \varphi \quad \frac{}{\ast G \text{ sat } (b_G \land \varphi) \land \text{Eval } \land \text{TimeOut}}
$$

Next consider parallel composition of $S_1$ and $S_2$. Suppose we have deduced specifications $\varphi_1$ and $\varphi_2$ for, respectively, $S_1$ and $S_2$. If $\varphi_1$ and $\varphi_2$ do not contain term then we have the following simple rule:

**Rule 5.16 (Simple Parallel Composition)**

$$
\begin{align*}
S_1 \text{ sat } \varphi_1, & \quad S_2 \text{ sat } \varphi_2, \quad \text{neither } \varphi_1 \text{ nor } \varphi_2 \text{ contain term} \\
S_1 \parallel S_2 \text{ sat } & \varphi_1 \land \varphi_2
\end{align*}
$$

provided $\text{dch}(\varphi_1) \subseteq \text{dch}(S_i)$ and $\text{var}(\varphi_1) \subseteq \text{var}(S_i)$, for $i = 1, 2$.

If one of $\varphi_1$ and $\varphi_2$ contains term, we have to take into account that the termination times of $S_1$ and $S_2$ are, in general, different. Observe that if $S_1$ terminates after (or at the same time as) $S_2$ then the model representing this computation satisfies $\varphi_1 \land (\varphi_2 \land \text{true})$. Furthermore we have to express that the variables of $S_2$ are not changed and there is no activity on the channels of $S_2$ after the termination of $S_2$. Similarly, for $S_1$ and $S_2$ interchanged. Then it leads to the following general rule for parallel composition:

19
Rule 5.17 (Parallel Composition) Let \( \psi_1 \equiv \text{inv}(\text{var}(S_2)) \land \square \text{empty}(\text{dch}(S_2)) \) and \( \psi_2 \equiv \text{inv}(\text{var}(S_1)) \land \square \text{empty}(\text{dch}(S_1)) \).

\[
\frac{S_1 \text{ sat } \psi_1, S_2 \text{ sat } \psi_2}{S_1 || S_2 \text{ sat } (\psi_1 \land (\psi_2 \land \psi_1)) \lor (\psi_2 \land (\psi_1 \land \psi_2))}
\]

provided \( \text{dch}(\varphi_i) \subseteq \text{dch}(S_i) \) and \( \text{var}(\varphi_i) \subseteq \text{var}(S_i) \), for \( i = 1, 2 \).

Example 5.4 Consider a program \( c!5 || c?x \). Since we have assumed maximal parallelism, the communication takes place immediately, and hence this program should satisfy \( \text{comm}(c, 5) \lor (T = \text{term} = \text{start} + K_c \land x = 5) \). By the Input and Output Axioms and the Consequence Rule, we obtain \( c!5 \text{ sat } \varphi_1 \) and \( c?x \text{ sat } \varphi_2 \) with

\[
\varphi_1 \equiv \text{wait}(c!) \text{ U } (T = \text{term} - K_c \land (\text{comm}(c, 5) \text{ U } T = \text{term})) \quad \text{and} \quad \varphi_2 \equiv \text{wait}(c?) \text{ U } (T = \text{term} - K_c \land (\text{comm}(c, x) \text{ U } T = \text{term}) \land (x = v_1 \rightarrow \square (x = v_1))).
\]

Suppose \( \psi_1 \equiv \text{inv}([x]) \land \square \text{empty}([c, e_c]) \) and \( \psi_2 \equiv \square \text{empty}([c, c_c]) \). Then the General Parallel Composition Rule leads to

\[
c!5 || c?x \text{ sat } (\psi_1 \land (\psi_2 \land \psi_1)) \lor (\psi_2 \land (\psi_1 \land \psi_2)).
\]

The Well-Formedness Axiom and the Conjunction Rule allow us to add \( \text{MinWait}_{\{c, e_c\}}, \text{Exclusion}_{\{c, e_c\}} \), and \( \text{Unique}_{\{c\}} \) to \( \psi_1 \land (\psi_2 \land \psi_1) \lor (\psi_2 \land (\psi_1 \land \psi_2)) \). Consider \( \varphi_1 \land (\psi_2 \land \psi_1) \land \text{MinWait}_{\{c, e_c\}} \land \text{Exclusion}_{\{c, e_c\}} \land \text{Unique}_{\{c\}} \). It implies

\[
[\text{wait}(c!) \text{ U } (T = \text{term} - K_c \land (\text{comm}(c, 5) \text{ U } T = \text{term}))] \land 
[[\text{wait}(c?) \land \lnot \text{wait}(c!) \land \lnot \text{comm}(c)] \text{ U } ((\text{comm}(c, x) \land \lnot \text{wait}(c!)) \land \text{inv}([x]))] \land \text{Unique}_{\{c\}},
\]

which implies

\[
T = \text{term} - K_c \land (\text{comm}(c, 5) \text{ U } T = \text{term}) \land \square (x = 5).
\]

Since \( T = \text{start} \), we can then derive \( \text{comm}(c, 5) \text{ U } (T = \text{term} = \text{start} + K_c \land x = 5) \).

Similarly, we can also obtain that

\[
\varphi_2 \land (\psi_1 \land \psi_2) \land \text{MinWait}_{\{c, e_c\}} \land \text{Exclusion}_{\{c, e_c\}} \land \text{Unique}_{\{c\}} \text{ implies}
\]

\[
\text{comm}(c, 5) \text{ U } (T = \text{term} = \text{start} + K_c \land x = 5).
\]

Then, using the Consequence Rule again, we obtain

\[
c!5 || c?x \text{ sat } \text{comm}(c, 5) \text{ U } (T = \text{term} = \text{start} + K_c \land x = 5). \quad \blacksquare
\]

Example 5.5 Consider program \( c!0; d!1 || d?x; c?y \). Since this program leads to deadlock, we should be able to prove \( c!0; d!1 || d?x; c?y \text{ sat } \square (\text{wait}(c!) \land \text{wait}(d!)) \).

By the Output Axiom, the Communication Invariance Axiom, and the Consequence Rule, we can derive \( c!0 \text{ sat } \text{wait}(c!) \text{ U } \text{comm}(c) \text{ and } c!0 \text{ sat } \lnot \text{comm}(d) \).

Using the Conjunction Rule and the Consequence Rule, we can combine these two formulae into \( c!0 \text{ sat } (\text{wait}(c!) \land \lnot \text{comm}(d)) \text{ U } (\text{comm}(c) \land \lnot \text{comm}(d)). \)

Since \( (\text{wait}(c!) \land \lnot \text{comm}(d)) \text{ U } (\text{comm}(c) \land \lnot \text{comm}(d)) \text{ true implies } (\text{wait}(c!) \land \lnot \text{comm}(d)) \text{ U } (\text{comm}(c) \land \lnot \text{comm}(d)) \text{, the Sequential Composition Rule and the Consequence Rule lead to } c!0; d!1 \text{ sat } (\text{wait}(c!) \land \lnot \text{comm}(d)) \text{ U } (\text{comm}(d) \land \lnot \text{comm}(d)). \)

Similarly, we have \( d?x; c?y \text{ sat } (\text{wait}(d?) \land \lnot \text{comm}(c)) \text{ U } (\text{comm}(d) \land \lnot \text{comm}(c)). \)

From the Simple Parallel Composition Rule, we obtain

\[
c!0; d!1 || d?x; c?y \text{ sat } ((\text{wait}(c!) \land \lnot \text{comm}(d)) \text{ U } (\text{comm}(c) \land \lnot \text{comm}(d))) \land
\]
Clearly this assertion implies \( \square (\text{wait}(c!) \land \text{wait}(d?)) \) and hence, by the Consequence Rule, \( c!0; d!1 \parallel d?x; c?y \text{ sat } \square (\text{wait}(c!) \land \text{wait}(d?)). \)

6 Soundness and Completeness

In order to prove the soundness of our proof system, we need the following lemmas.

**Lemma 6.1** For any expression \( e \) from the programming language, any model \( \sigma \), any environment \( \gamma \), and any \( \tau \in \text{TIME}, \tau \geq \text{begin}(\sigma), \mathcal{E}(e)(\sigma(\tau).\text{state}) = \mathcal{V}(e)(\gamma(\sigma, \tau)). \)

The proofs of all lemmas can be found in Appendix A.

**Lemma 6.2** For any boolean expression \( b \) from the programming language, any model \( \sigma \), any environment \( \gamma \), and any \( \tau \in \text{TIME}, \tau \geq \text{begin}(\sigma), \mathcal{B}(b)(\sigma(\tau).\text{state}) \iff (\sigma, \tau) \models b. \)

Regarding the soundness of our real-time proof system, we must show that every formula \( S \text{ sat } \varphi \) derivable in the proof system is indeed valid.

**Theorem 6.3 (Soundness)** The proof system is sound.

**Proof:** We have to show that all axioms are valid and all inference rules preserve the validity, i.e. if the hypotheses of any rule are valid, so is the conclusion. For most axioms and inference rules, soundness follows directly from the definition of the semantics, by using Lemma 6.1 and Lemma 6.2. The details of the proof appear in Appendix B.

We would also like the proof system to be complete, i.e. if \( S \text{ sat } \varphi \) is valid then it is derivable from our proof system. Observe that the Consequence Rule relies on implications that are formulae in ECTL (Explicit Clock Temporal Logic), and hence the completeness of our proof system also requires that every valid ECTL formula is provable. Since proof systems for ECTL are beyond the scope of this paper, we prove relative completeness: Every valid specification is derivable in our proof system, assuming that any valid ECTL formula can be proved.

In order to prove the relative completeness of our proof system, we need the following lemmas. The first nine lemmas express some properties of the projection operations.

**Lemma 6.4** For any model \( \sigma \) and any cset \( \subseteq \text{DCHAN}, \text{dch}(\sigma) \subseteq \text{cset} \iff \sigma = [\sigma]_{\text{cset}}. \)

**Lemma 6.5** For any expression \( vexp \) of type VAL, any model \( \sigma \), any environment \( \gamma \), and any cset \( \subseteq \text{DCHAN}, \mathcal{V}(vexp)(\gamma(\sigma, \text{begin}(\sigma)) = \mathcal{V}(vexp)(\gamma([\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}}))). \)

**Lemma 6.6** For any expression \( vexp \) of type VAL, any model \( \sigma \), any environment \( \gamma \), and any vset \( \subseteq \text{VAR}, \mathcal{V}(vexp)(\gamma(\sigma, \text{begin}(\sigma)) = \mathcal{V}(vexp)(\gamma(\sigma \uparrow \text{vset}, \text{begin}(\sigma \uparrow \text{vset})), \text{if var}(vexp) \subseteq \text{vset}. \)
Lemma 6.7 For any expression $texp$ of type $TIME$, any model $\sigma$, any environment $\gamma$, and any $cset \subseteq DCHAN$, $T(texp)\gamma(\sigma, \text{begin}(\sigma)) = T(texp)\gamma([\sigma]_{cset}, \text{begin}([\sigma]_{cset}))$.

Lemma 6.8 For any expression $texp$ of type $TIME$, any model $\sigma$, any environment $\gamma$, and any $vset \subseteq VAR$, $T(texp)\gamma(\sigma, \text{begin}(\sigma)) = T(texp)\gamma(\sigma \uparrow vset, \text{begin}(\sigma \uparrow vset))$, if $\text{var}(texp) \subseteq vset$.

Lemma 6.9 For any $cset \subseteq DCHAN$ and any assertion $\varphi$, if $dch(\varphi) \subseteq cset$, then for any model $\sigma$ and any environment $\gamma$, $(\sigma, \text{begin}(\sigma))\gamma \models \varphi$ iff $(\sigma[cset, \text{begin}([\sigma]_{cset})])\gamma \models \varphi$.

Lemma 6.10 For any $vset \subseteq VAR$ and any assertion $\varphi$, if $\text{var}(\varphi) \subseteq vset$, then for any model $\sigma$ and any environment $\gamma$, $(\sigma, \text{begin}(\sigma))\gamma \models \varphi$ iff $(\sigma \uparrow vset, \text{begin}(\sigma \uparrow vset))\gamma \models \varphi$.

Lemma 6.11 For any model $\sigma$, any environment $\gamma$, and any $cset_1, cset_2 \subseteq DCHAN$, if $(\sigma, \text{begin}(\sigma))\gamma \models \square \text{empty}(\overline{cset_2 - cset_1})$ then $[\sigma]_{cset_1 \cup cset_2} = [\sigma]_{cset_1}$.

Lemma 6.12 For any model $\sigma$, any environment $\gamma$, and any $vset_1, vset_2 \subseteq VAR$, if $(\sigma, \text{begin}(\sigma))\gamma \models \text{inv}(vset_2 - vset_1)$, then $\sigma \uparrow (vset_1 \cup vset_2) = \sigma \uparrow vset_1$.

The next lemma expresses the relation between a well-formed model and the assertion $WF_{cset}$.

Lemma 6.13 For any model $\sigma$ and any environment $\gamma$, if $dch(\sigma) \subseteq cset$ and $(\sigma, \text{begin}(\sigma))\gamma \models WF_{cset}$, then $\sigma$ is well-formed.

In order to prove the relative completeness of our system, we need to define a property of assertions called $\text{preciseness}$. Before giving the definition, we first define the following concept.

Definition 6.14 (Invariant) A program variable $x$ is invariant with respect to a model $\sigma$ iff for all $\tau$, $\text{begin}(\sigma) \leq \tau \leq \text{end}(\sigma)$, $\sigma(\tau).\text{state}(x) = \sigma(\text{begin}(\sigma)).\text{state}(x)$.

Next we give the definition of $\text{preciseness}$:

Definition 6.15 (Preciseness) An assertion $\varphi$ is precise for a statement $S$ iff

1. $S$ sat $\varphi$, i.e. $(\sigma, \text{begin}(\sigma))\gamma \models \varphi$, for any $\sigma \in M(S)$ and any environment $\gamma$;

2. If $\sigma$ is a well-formed model, $dch(\sigma) \subseteq dch(S)$, for any program variable $x \notin \text{wvar}(S)$, $x$ is invariant with respect to $\sigma$, and $(\sigma, \text{begin}(\sigma))\gamma \models \varphi$ for any environment $\gamma$, then $\sigma \in M(S)$; and

3. $dch(\varphi) = dch(S)$ and $\text{var}(\varphi) = \text{var}(S)$.

A precise assertion $\varphi$ for $S$ thus characterizes all possible computations of $S$: $\varphi$ is valid for $S$, and any "reasonable" computation satisfying $\varphi$ is a possible computation of $S$.

We first prove that for any program $S$ a precise specification can be derived from the axioms and inference rules (Theorem 6.16). We then show (in Theorem 6.17) that any assertion $\varphi_2$ which is valid for $S$ can be derived from a precise assertion $\varphi_1$ for $S$ and three predicates. Hence, relative completeness follows directly (Theorem 6.18).
Theorem 6.16 If \( S \) is a statement then a precise specification for \( S \) can be derived by using the proof system.

The proof of this theorem can be found in Appendix C.

Theorem 6.17 If \( \varphi_1 \) is precise for \( S \) and \( \varphi_2 \) is valid for \( S \), then
\[
\models [\varphi_1 \land WF_{dch(\varphi_1)} \land \square \text{empty}(dch(\varphi_2) - dch(\varphi_1)) \land \text{inv}(\text{var}(\varphi_2) - \text{var}(\varphi_1))] \rightarrow \varphi_2.
\]

Proof: Let \( \varphi_1 \) is precise for \( S \) and \( \varphi_2 \) is valid for \( S \). Consider a model \( \sigma \) and an environment \( \gamma \). Assume \( (\sigma, \text{begin}(\sigma)) \models \varphi_1 \land WF_{dch(\varphi_1)} \land \square \text{empty}(dch(\varphi_2) - dch(\varphi_1))\land \text{inv}(\text{var}(\varphi_2) - \text{var}(\varphi_1)) \). We show \( (\sigma, \text{begin}(\sigma)) \models \varphi_2 \). By \( (\sigma, \text{begin}(\sigma)) \models \varphi_1 \), Lemma 6.9 leads to \( ([\sigma]_{dch(\varphi_1)}, \text{begin}([\sigma]_{dch(\varphi_1)})) \models \varphi_1 \). By Lemma 6.10, \( ([\sigma]_{dch(\varphi_1)} \uparrow \text{var}(\varphi_1), \text{begin}([\sigma]_{dch(\varphi_1)} \uparrow \text{var}(\varphi_1))) \models \varphi_1 \). Since \( dch(WF_{dch(\varphi_1)}) = dch(\varphi_1) \), we obtain \( ([\sigma]_{dch(\varphi_1)}, \text{begin}([\sigma]_{dch(\varphi_1)})) \models WF_{dch(\varphi_1)} \). Then by Lemma 6.13, \( [\sigma]_{dch(\varphi_1)} \) is well-formed. By definition, \( [\sigma]_{dch(\varphi_1)} \uparrow \text{var}(\varphi_1) \) is also well-formed. Since \( \varphi_1 \) is precise for \( S \), we have \( dch(\varphi_1) = dch(S) \) and \( \text{var}(\varphi_1) = \text{var}(S) \). By the definition of projection onto variables, every variable \( x \notin \text{wvar}(S) \) is invariant with respect to \( [\sigma]_{dch(\varphi_1)} \uparrow \text{var}(\varphi_1) \). Hence by the definition of preciseness, \( [\sigma]_{dch(\varphi_1)} \uparrow \text{var}(\varphi_1) \in M(S) \). From \( (\sigma, \text{begin}(\sigma)) \models \square \text{empty}(dch(\varphi_2) - dch(\varphi_1)) \), Lemma 6.11 leads to \( (\sigma)_{dch(\varphi_1)} \uparrow \text{var}(\varphi_2) = \sigma \uparrow \text{var}(\varphi_1) \). Therefore \( (\sigma)_{dch(\varphi_1)} \uparrow \text{var}(\varphi_1) \uparrow \text{var}(\varphi_2) \in M(S) \). Since \( \varphi_2 \) is valid for \( S \), for any environment \( \gamma \), \( (\sigma)_{dch(\varphi_1)} \uparrow \text{var}(\varphi_1) \uparrow \text{var}(\varphi_2), \text{begin}(\sigma) \models \varphi_2 \). From \( \text{var}(\varphi_2) \subseteq \text{var}(\varphi_1) \cup \text{var}(\varphi_2) \), Lemma 6.10 leads to \( ([\sigma]_{dch(\varphi_1)} \uparrow \text{dch}(\varphi_2), \text{begin}(\sigma)) \models \varphi_2 \). By \( \text{dch}(\varphi_2) \subseteq (dch(\varphi_1) \cup dch(\varphi_2)) \), Lemma 6.9 leads to \( (\sigma, \text{begin}(\sigma)) \models \varphi_2 \). Then we have proved our theorem.

Theorem 6.18 (Relative Completeness) If assertion \( \varphi \) is valid for program \( S \), then \( S \) sat \( \varphi \) is derivable in the proof system.

Proof: By Theorem 6.16, we can derive \( S \) sat \( \varphi_1 \) where \( \varphi_1 \) is a precise assertion for \( S \). By the Well-Formedness Axiom we can derive \( S \) sat \( WF_{dch(\varphi_1)} \). Since \( dch(\varphi_1) = dch(S) \), we obtain that \( [dch(\varphi) - dch(\varphi_1)] \cap dch(S) = \emptyset \). Then by the Communication Invariance Axiom, \( S \) sat \( \square \text{empty}(dch(\varphi) - dch(\varphi_1)) \). From the Variable Invariance Axiom, we can derive \( S \) sat \( \text{inv}(\text{var}(\varphi) - \text{var}(\varphi_1)) \) since \( [\text{var}(\varphi) - \text{var}(\varphi_1)] \cap \text{var}(S) = \emptyset \). Then by the Conjunction Rule leads to \( S \) sat \( \varphi_1 \land WF_{dch(\varphi_1)} \land \square \text{empty}(dch(\varphi) - dch(\varphi_1)) \land \text{inv}(\text{var}(\varphi) - \text{var}(\varphi_1)) \). By Theorem 6.17, \( [\varphi_1 \land WF_{dch(\varphi_1)} \land \square \text{empty}(dch(\varphi) - dch(\varphi_1)) \land \text{inv}(\text{var}(\varphi) - \text{var}(\varphi_1))] \rightarrow \varphi \) is valid and, by our relative completeness assumption, provable. Hence, by the Consequence Rule, \( S \) sat \( \varphi \) is derivable in the proof system.

7 Conclusion

We have formulated a proof system to verify that a program satisfies a specification written in a version of explicit clock temporal logic. We also have shown that the proof system is
sound, that is, every formula $S \text{ sat } \varphi$ which can be proven in our proof system is indeed valid. Furthermore we have established relative completeness, that is, if valid formulas from our assertion language can be proven, then any valid formula $S \text{ sat } \varphi$ can also be derived in our axiomatic system.

Important is that our axiomatization is compositional, and hence allows us to reduce the complexity of the verification problem. In contrast with for instance [Ost89], we have not required a discrete time domain. Often a dense time domain is convenient, since it allows events to be arbitrary close to each other in time. This is important when reactive systems are modeled. Such systems have an intensive interaction with the environment and external events may occur at any moment of time. Moreover, by having dense time we can easily reason about the timing of atomic actions on one level of abstraction and their refinement into several actions on a lower, more concrete, level. A close look at the development of the theory reveals that this choice is orthogonal to the rest of the theory. Therefore, should for other reasons, e.g., efficient automatic verification, a discrete domain appear more attractive, then this can be substituted without causing changes to the theory elsewhere.

As mentioned in the introduction, the proof system from this paper is closely related to the compositional axiomatization given in [HW89] where MTL is used instead of ECTL. In MTL, explicit bounds are added to the modal operators to obtain quantitative timing. For instance, there is a real-time until operator $\varphi_1 \mathcal{U}_{<d} \varphi_2$ which expresses that $\varphi_2$ will be true within $d$ time units and until that point $\varphi_1$ holds. In our framework this operator could be defined as follows: $(\sigma, \tau) \gamma \models \psi_1 \mathcal{U}_{<d} \psi_2$ iff there exists a $\tau_1, \tau \leq \tau_1 < \tau + d$, such that $(\sigma, \tau_1) \gamma \models \psi_2$, and for all $\tau_2, \tau \leq \tau_2 < \tau_1, (\sigma, \tau_2) \gamma \models \psi_1$. Observe that $\varphi_1 \mathcal{U}_{<d} \varphi_2$ in MTL is equivalent to $T = t - (\varphi_1 \mathcal{U} (T < t + d \land \varphi_2))$ in ECTL, where $\varphi_1$ and $\varphi_2$ are formulae in ECTL which are equivalent to $\varphi_1$ and $\varphi_2$, respectively. From this transformation, it is clear that in MTL all timing properties are specified relative to a starting point, whereas ECTL allows direct references to the value of the clock. In future work we will address the precise relation between these two approaches and investigate, by means of examples, which version is most applicable for specific applications.

The present paper is part of a larger, more ambitious, endeavour to describe scheduling with respect to limited resources in an abstract, compositional, manner. The desired abstraction would be to describe how, for a given scheduler and resources, combination of two components influences their execution time without modeling the effect of the scheduler at the computation step level. In this paper we believe to have obtained some more insight in the description of time properties, be it in the limited context of maximal parallelism.

References


Appendix A

Proofs of Lemmas

Proof of Lemma 6.1

Consider any expression \( e \) from the programming language, any model \( \sigma \), any environment \( \gamma \), and any \( \tau \in \text{TIME}, \tau \geq \begin{ \text{begin}(\sigma) \). \) Prove \( \mathcal{E}(e)(\sigma(\tau).\text{state}) = \mathcal{V}(e)(\gamma(\sigma, \tau)) \).

Proof: We give the proof by induction on the structure of \( e \).

- \( e \equiv \emptyset \). Then \( \mathcal{E}(\emptyset)(\sigma(\tau).\text{state}) = \emptyset = \mathcal{V}(\emptyset)(\gamma(\sigma, \tau)) \).
- \( e \equiv \text{a} \). Then \( \mathcal{E}(\text{a})(\sigma(\tau).\text{state}) = (\gamma(\sigma, \tau) \).
- \( e \equiv e_1 + e_2 \). By the induction hypothesis, \( \mathcal{E}(e_1)(\sigma(\tau).\text{state}) = \mathcal{V}(e_1)(\gamma(\sigma, \tau)) \) and \( \mathcal{E}(e_2)(\sigma(\tau).\text{state}) = \mathcal{V}(e_2)(\gamma(\sigma, \tau)) \). Then, \( \mathcal{E}(e_1 + e_2)(\sigma(\tau).\text{state}) = \mathcal{E}(e_1)(\sigma(\tau).\text{state}) + \mathcal{E}(e_2)(\sigma(\tau).\text{state}) = \mathcal{V}(e_1)(\gamma(\sigma, \tau)) + \mathcal{V}(e_2)(\gamma(\sigma, \tau)) = \mathcal{V}(e_1 + e_2)(\gamma(\sigma, \tau)) \).
- \( e \equiv e_1 \times e_2 \). By the induction hypothesis, \( \mathcal{E}(e_1)(\sigma(\tau).\text{state}) = \mathcal{V}(e_1)(\gamma(\sigma, \tau)) \) and \( \mathcal{E}(e_2)(\sigma(\tau).\text{state}) = \mathcal{V}(e_2)(\gamma(\sigma, \tau)) \). Then, \( \mathcal{E}(e_1 \times e_2)(\sigma(\tau).\text{state}) = \mathcal{E}(e_1)(\sigma(\tau).\text{state}) \times \mathcal{E}(e_2)(\sigma(\tau).\text{state}) = \mathcal{V}(e_1)(\gamma(\sigma, \tau)) \times \mathcal{V}(e_2)(\gamma(\sigma, \tau)) = \mathcal{V}(e_1 \times e_2)(\gamma(\sigma, \tau)) \).

Proof of Lemma 6.2

Consider any boolean expression \( b \) from the programming language, any model \( \sigma \), any environment \( \gamma \), and any \( \tau \in \text{TIME}, \tau \geq \begin{ \text{begin}(\sigma) \). \) Prove that \( B(b)(\sigma(\tau).\text{state}) \iff (\sigma, \tau) |\!| b \). \)

Proof: The proof is given by induction on the structure of \( b \).

- \( b \equiv \text{e}_1 = \text{e}_2 \). Thus, \( B(\text{e}_1 = \text{e}_2)(\sigma(\tau).\text{state}) \iff \mathcal{E}(\text{e}_1)(\sigma(\tau).\text{state}) = \mathcal{E}(\text{e}_2)(\sigma(\tau).\text{state}) \iff \mathcal{V}(\text{e}_1)(\gamma(\sigma, \tau)) = \mathcal{V}(\text{e}_2)(\gamma(\sigma, \tau)) \iff (\sigma, \tau) |\!| \text{e}_1 = \text{e}_2 \).
- \( b \equiv \text{e}_1 < \text{e}_2 \). Thus, \( B(\text{e}_1 < \text{e}_2)(\sigma(\tau).\text{state}) \iff \mathcal{E}(\text{e}_1)(\sigma(\tau).\text{state}) < \mathcal{E}(\text{e}_2)(\sigma(\tau).\text{state}) \iff \mathcal{V}(\text{e}_1)(\gamma(\sigma, \tau)) < \mathcal{V}(\text{e}_2)(\gamma(\sigma, \tau)) \iff (\sigma, \tau) |\!| \text{e}_1 < \text{e}_2 \).
- \( b \equiv \neg b_1 \). Thus, \( B(\neg b_1)(\sigma(\tau).\text{state}) \iff \neg B(b_1)(\sigma(\tau).\text{state}) \iff \neg (\sigma, \tau) |\!| b_1 \iff (\sigma, \tau) |\!| \neg b_1 \).
- \( b \equiv b_1 \lor b_2 \). Thus, \( B(b_1 \lor b_2)(\sigma(\tau).\text{state}) \iff B(b_1)(\sigma(\tau).\text{state}) \lor B(b_2)(\sigma(\tau).\text{state}) \iff \neg (\sigma, \tau) |\!| b_1 \lor b_2 \iff (\sigma, \tau) |\!| \neg b_1 \lor b_2 \).

Proof of Lemma 6.4

Consider a model \( \sigma \) and \( \text{cset} \subseteq \text{DCHAN} \). Prove that \( \text{dch}(\sigma) \subseteq \text{cset} \iff \sigma = [\sigma]_{\text{cset}} \).

Proof: By definition of projection, \( \begin{ \text{begin}(\sigma) = \begin{ \text{begin}(\sigma)_{\text{cset}}, \end{end}(\sigma) = \end{end}(\sigma)_{\text{cset}} \), \) and for all \( \tau_1, \begin{ \text{begin}(\sigma) \leq \tau_1 \leq \begin{ \text{end}(\sigma) \), \( \sigma(\tau_1).\text{state} = [\sigma]_{\text{cset}}(\tau_1).\text{state}. \) Then we only have to prove that, for all \( \tau, \begin{ \text{begin}(\sigma) \leq \tau < \begin{ \text{end}(\sigma) \), \( \sigma(\tau).\text{comm} = [\sigma]_{\text{cset}}(\tau).\text{comm} \iff \text{dch}(\sigma) \subseteq \text{cset}. \) \)

Let \( c \in \text{CHAN} \) and \( \emptyset \in \text{VAL} \). By definition, for all \( \tau \), \( \begin{ \text{begin}(\sigma) \leq \tau < \begin{ \text{end}(\sigma) \), \( [\sigma]_{\text{cset}}(\tau).\text{comm} = \{ c! \mid c! \in \sigma(\tau).\text{comm} \land c! \in \text{cset} \} \cup \{ c? \mid c? \in \sigma(\tau).\text{comm} \land c? \in \text{cset} \} \cup \{ (c, \emptyset) \mid (c, \emptyset) \in \sigma(\tau).\text{comm} \land c \in \text{cset} \} \)
and
\[
dch(\sigma) = \bigcup_{\begin{subarray}{l}
\text{begin}(\sigma) \leq \tau < \text{end}(\sigma)
\end{subarray}} \{c! | c! \in \sigma(\tau)\text{.comm}\} \cup \\
\{c? | c? \in \sigma(\tau)\text{.comm}\} \cup \\
\{c | \text{there exists a } \vartheta \text{ such that } (c, \vartheta) \in \sigma(\tau)\text{.comm}\}
\]

Assume \(dch(\sigma) \subseteq cset\). We show \(\sigma(\tau)\text{.comm} = [\sigma]_{cset}(\tau)\text{.comm}\), for all \(\tau\), \(\begin{subarray}{l}
\text{begin}(\sigma) \leq \tau < \text{end}(\sigma)
\end{subarray}\). If \(c! \in \sigma(\tau)\text{.comm}\), then \(c! \in dch(\sigma)\). By the assumption, \(c! \in cset\) and thus \(c! \in [\sigma]_{cset}(\tau)\text{.comm}\). Similarly, if \(c? \in \sigma(\tau)\text{.comm}\), then \(c? \in [\sigma]_{cset}(\tau)\text{.comm}\). If \((c, \vartheta) \in \sigma(\tau)\text{.comm}\), then \((c, \vartheta) \in [\sigma]_{cset}(\tau)\text{.comm}\).

Thus \(\sigma(\tau)\text{.comm} \subseteq [\sigma]_{cset}(\tau)\text{.comm}\). On the other hand, if \(c! \in [\sigma]_{cset}(\tau)\text{.comm}\), then \(c! \in \sigma(\tau)\text{.comm}\). If \((c, \vartheta) \in [\sigma]_{cset}(\tau)\text{.comm}\), then \((c, \vartheta) \in \sigma(\tau)\text{.comm}\). Therefore \([\sigma]_{cset}(\tau)\text{.comm} \subseteq \sigma(\tau)\text{.comm}\). Hence
\[
\sigma(\tau)\text{.comm} = [\sigma]_{cset}(\tau)\text{.comm}
\]

On the other hand, if \(c! \in [\sigma]_{cset}(\tau)\text{.comm}\), then \(c! \in \sigma(\tau)\text{.comm}\). If \((c, \vartheta) \in [\sigma]_{cset}(\tau)\text{.comm}\), then \((c, \vartheta) \in \sigma(\tau)\text{.comm}\). Therefore \(\sigma(\tau)\text{.comm} = [\sigma]_{cset}(\tau)\text{.comm}\) iff \(dch(\sigma) \subseteq cset\).

Proof of Lemma 6.5

Consider any expression \(vexp\) of type \(VAL\), any model \(\sigma\), any environment \(\gamma\), and any \(cset \subseteq DCHAN\). Prove that \(\mathcal{V}(vexp)\gamma(\sigma, \text{begin}(\sigma)) = \mathcal{V}(vexp)\gamma([\sigma]_{cset}, \text{begin}([\sigma]_{cset}))\).

Proof: We give the proof by induction on the structure of \(vexp\). Observe that, by the definition of projection onto channels, \(\text{begin}(\sigma) = \text{begin}([\sigma]_{cset})\), for any \(cset \subseteq DCHAN\).

- \(vexp \equiv \vartheta\). Then \(\mathcal{V}(\vartheta)\gamma(\sigma, \text{begin}(\sigma)) = \vartheta = \mathcal{V}(\vartheta)\gamma([\sigma]_{cset}, \text{begin}([\sigma]_{cset})).\)
- \(vexp \equiv v\). Then \(\mathcal{V}(v)\gamma(\sigma, \text{begin}(\sigma)) = \gamma(v) = \mathcal{V}(v)\gamma([\sigma]_{cset}, \text{begin}([\sigma]_{cset})).\)
- \(vexp \equiv x\). Then \(\mathcal{V}(x)\gamma(\sigma, \text{begin}(\sigma)) = \sigma(\text{begin}(\sigma)).\text{state}(x) = [\sigma]_{cset}(\text{begin}([\sigma]_{cset})).\text{state}(x) = \mathcal{V}(x)\gamma([\sigma]_{cset}, \text{begin}([\sigma]_{cset})).\)
- \(vexp \equiv \text{last}(x)\). Then \(\mathcal{V}(\text{last}(x))\gamma(\sigma, \text{begin}(\sigma)) = \sigma(\text{end}(\sigma)).\text{state}(x) = [\sigma]_{cset}(\text{end}([\sigma]_{cset})).\text{state}(x) = \mathcal{V}(\text{last}(x))\gamma([\sigma]_{cset}, \text{begin}([\sigma]_{cset})).\)
- \(vexp \equiv vexp_1 + vexp_2\). By the induction hypothesis,
\[
\mathcal{V}(vexp_1)\gamma(\sigma, \text{begin}(\sigma)) = \mathcal{V}(vexp_1)\gamma([\sigma]_{cset}, \text{begin}([\sigma]_{cset}))\text{ and } \\
\mathcal{V}(vexp_2)\gamma(\sigma, \text{begin}(\sigma)) = \mathcal{V}(vexp_2)\gamma([\sigma]_{cset}, \text{begin}([\sigma]_{cset})).
\]
Thus \(\mathcal{V}(vexp_1 + vexp_2)\gamma(\sigma, \text{begin}(\sigma)) = \mathcal{V}(vexp_1 + vexp_2)\gamma([\sigma]_{cset}, \text{begin}([\sigma]_{cset})).\)
- \(vexp \equiv vexp_1 \times vexp_2\). The proof is similar as for \(vexp \equiv vexp_1 + vexp_2\).
Proof of Lemma 6.6

Consider any expression $vexp$ of type $VAL$, any model $\sigma$, any environment $\gamma$, and any $vset \subseteq VAR$. Prove that $V(vexp)\gamma(\sigma, begin(\sigma)) = V(vexp)\gamma(\sigma \uparrow vset, begin(\sigma \uparrow vset))$, if $var(vexp) \subseteq vset$.

Proof: By the definition of projection onto variables, we obtain $begin(\sigma) = begin(\sigma \uparrow vset)$, for any $vset \subseteq VAR$. Follow the definition and the induction proof for Lemma 6.5, we can easily prove this lemma.

Proof of Lemma 6.7

Consider any expression $texp$ of type $TIME$, any model $\sigma$, any environment $\gamma$, and any $cset \subseteq DCHAN$. Prove that $T(texp)\gamma(\sigma, begin(\sigma)) = T(texp)\gamma([\sigma]_{cset}, begin([\sigma]_{cset}))$.

Proof: By definition, $begin(\sigma) = begin([\sigma]_{cset})$, for any $cset \subseteq DCHAN$. The proof is given by induction on the structure of $texp$.

- $texp \equiv \tau$. Then $T(\tau)\gamma(\sigma, begin(\sigma)) = T(\tau)\gamma([\sigma]_{cset}, begin([\sigma]_{cset}))$.
- $texp \equiv t$. Then $T(t)\gamma(\sigma, begin(\sigma)) = T(t)\gamma([\sigma]_{cset}, begin([\sigma]_{cset}))$.
- $texp \equiv T$. Then $T(T)\gamma(\sigma, begin(\sigma)) = begin(\sigma) = begin([\sigma]_{cset}) = T(T)\gamma([\sigma]_{cset}, begin([\sigma]_{cset}))$.
- $texp \equiv start$. Then $T(start)\gamma(\sigma, begin(\sigma)) = begin(\sigma) = begin([\sigma]_{cset}) = T(start)\gamma([\sigma]_{cset}, begin([\sigma]_{cset}))$.
- $texp \equiv term$. Then $T(term)\gamma(\sigma, begin(\sigma)) = end(\sigma) = end([\sigma]_{cset}) = T(term)\gamma([\sigma]_{cset}, begin([\sigma]_{cset}))$.
- $texp \equiv vexp$. By definition, $T(vexp)\gamma(\sigma, begin(\sigma)) = V(vexp)\gamma(\sigma, begin(\sigma))$ and $T(vexp)\gamma([\sigma]_{cset}, begin([\sigma]_{cset})) = V(vexp)\gamma([\sigma]_{cset}, begin([\sigma]_{cset}))$. By Lemma 6.5, $V(vexp)\gamma(\sigma, begin(\sigma)) = V(vexp)\gamma([\sigma]_{cset}, begin([\sigma]_{cset}))$. Then it leads to $T(vexp)\gamma(\sigma, begin(\sigma)) = T(vexp)\gamma([\sigma]_{cset}, begin([\sigma]_{cset}))$.
- $texp \equiv texp_1 + texp_2$. By the induction hypothesis, $T(texp_1)\gamma(\sigma, begin(\sigma)) = T(texp_1)\gamma([\sigma]_{cset}, begin([\sigma]_{cset}))$ and $T(texp_2)\gamma(\sigma, begin(\sigma)) = T(texp_2)\gamma([\sigma]_{cset}, begin([\sigma]_{cset}))$. Thus, by definition, $T(texp_1 + texp_2)\gamma(\sigma, begin(\sigma)) = T(texp_1 + texp_2)\gamma([\sigma]_{cset}, begin([\sigma]_{cset}))$.
- $texp \equiv texp_1 - texp_2$ and $texp \equiv texp_1 \times texp_2$. The proofs are similar as for $texp \equiv texp_1 + texp_2$.

Proof for Lemma 6.8

Consider any expression $texp$ of type $TIME$, any model $\sigma$, any environment $\gamma$, and any $vset \subseteq VAR$. Prove that $T(texp)\gamma(\sigma, begin(\sigma)) = T(texp)\gamma(\sigma \uparrow vset, begin(\sigma \uparrow vset))$, if $\var(texp) \subseteq vset$.

Proof: Similar to the proof for Lemma 6.7.
Proof for Lemma 6.9

Consider any cset \( DCHAN \) and any assertion \( \varphi \). Prove that if \( dch(\varphi) \subseteq cset \) then, for any model \( \sigma \) and any environment \( \gamma \), \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi \) if \( \langle [\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}}) \rangle \gamma \models \varphi \).

**Proof:** We give the proof by induction on the structure of \( \varphi \).

- \( \varphi \equiv \text{texp}_1 = \text{texp}_2 \). Then \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \text{texp}_1 = \text{texp}_2 \) if
  \[ T(\text{texp}_1) \gamma(\sigma, \text{begin}(\sigma)) = T(\text{texp}_2) \gamma(\sigma, \text{begin}(\sigma)) \text{ if, by Lemma 6.7,} \]
  \[ T(\text{texp}_1) \gamma([\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}})) = T(\text{texp}_2) \gamma([\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}})) \text{ iff, by definition,} \]
  \[ \langle [\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}}) \rangle \gamma \models \text{texp}_1 = \text{texp}_2. \]

- \( \varphi \equiv \text{comm}(c, \text{vexp}) \). Then \( \{c\} = \text{dch}(\varphi) \subseteq cset \), and thus \( c \in cset \). Hence
  \[ \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \text{comm}(c, \text{vexp}) \text{ iff} \]
  \[ \text{end}(\sigma) > \text{begin}(\sigma) \text{ and } (c, \text{vexp}) \gamma(\sigma, \text{begin}(\sigma)) \in \sigma(\text{begin}(\sigma)).\text{comm} \text{ iff, by definition and Lemma 6.5,} \]
  \[ \text{end}([\sigma]_{\text{cset}}) > \text{begin}([\sigma]_{\text{cset}}) \text{ and } (c, \text{vexp}) \gamma([\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}})) \in [\sigma]_{\text{cset}}(\text{begin}([\sigma]_{\text{cset}})).\text{comm} \text{ iff} \]
  \[ \langle [\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}}) \rangle \gamma \models \text{comm}(c, \text{vexp}). \]

- \( \varphi \equiv \text{wait}(c!) \). Thus \( \{c\} = \text{dch}(\varphi) \subseteq cset \) and then \( c! \in cset \). Therefore
  \[ \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \text{wait}(c!) \text{ iff} \]
  \[ \text{end}(\sigma) > \text{begin}(\sigma) \text{ and } c! \in \sigma(\text{begin}(\sigma)).\text{comm} \text{ iff, by the induction hypothesis,} \]
  \[ \text{end}([\sigma]_{\text{cset}}) > \text{begin}([\sigma]_{\text{cset}}) \text{ and } c! \in [\sigma]_{\text{cset}}(\text{begin}([\sigma]_{\text{cset}})).\text{comm} \text{ iff} \]
  \[ \langle [\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}}) \rangle \gamma \models \text{wait}(c!). \]

- \( \varphi \equiv \varphi_1 \lor \varphi_2 \). Then, for any \( i \in \{1,2\}, \text{dch}(\varphi_i) \subseteq \text{dch}(\varphi) \subseteq cset \). Hence
  \[ \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \lor \varphi_2 \text{ iff} \]
  \[ \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \text{ or } \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_2 \text{ iff, by the induction hypothesis,} \]
  \[ \langle [\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}}) \rangle \gamma \models \varphi_1 \text{ or } \langle [\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}}) \rangle \gamma \models \varphi_2 \text{ iff} \]
  \[ \langle [\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}}) \rangle \gamma \models \varphi_1 \lor \varphi_2. \]

- \( \varphi \equiv \varphi_1 \land \varphi_2 \). The proof is similar as for \( \varphi \equiv \varphi_1 \lor \varphi_2 \).

- \( \varphi \equiv \varphi_1 \land \varphi_2 \). Then, for any \( i \in \{1,2\}, \text{dch}(\varphi_i) \subseteq \text{dch}(\varphi) \subseteq cset \). Hence
  \[ \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \land \varphi_2 \text{ iff} \]
  \[ \text{there exist models } \sigma_1 \text{ and } \sigma_2 \text{ such that } \sigma = \sigma_1 \sigma_2, \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1, \text{ and } \langle \sigma_2, \text{begin}(\sigma_2) \rangle \gamma \models \varphi_2 \text{ iff, by the induction hypothesis,} \]
  \[ \text{there exist models } \sigma_1 \text{ and } \sigma_2 \text{ such that } \sigma = \sigma_1 \sigma_2, \langle [\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}}) \rangle \gamma \models \varphi_1, \text{ and } \langle [\sigma_2]_{\text{cset}}, \text{begin}([\sigma_2]_{\text{cset}}) \rangle \gamma \models \varphi_2 \text{ iff, by definition,} \]
  \[ \text{there exist models } [\sigma_1]_{\text{cset}} \text{ and } [\sigma_2]_{\text{cset}} \text{ such that } [\sigma]_{\text{cset}} = [\sigma_1]_{\text{cset}}[\sigma_2]_{\text{cset}}, \langle [\sigma_1]_{\text{cset}}, \text{begin}([\sigma_1]_{\text{cset}}) \rangle \gamma \models \varphi_1, \text{ and } \langle [\sigma_2]_{\text{cset}}, \text{begin}([\sigma_2]_{\text{cset}}) \rangle \gamma \models \varphi_2 \text{ iff} \]
  \[ \langle [\sigma]_{\text{cset}}, \text{begin}([\sigma]_{\text{cset}}) \rangle \gamma \models \varphi_1 \land \varphi_2. \]

- \( \varphi \equiv \text{wait}^{C}(\varphi_2) \). The proof is similar as for \( \varphi \equiv \varphi_1 \lor \varphi_2 \).
Proof for Lemma 6.10
Consider any vset ⊆ VAR and any assertion φ. Prove that if var(φ) ⊆ vset then, for any model σ and any environment γ, (σ, begin(σ))γ |= φ iff (σ ↠ vset), begin(σ ↠ vset))γ |= φ.
Proof: Similar to the proof for Lemma 6.9.

Proof of Lemma 6.11
Consider a model σ, an environment γ, and two sets cset₁, cset₂ ⊆ DCHAN. Prove that if (σ, begin(σ))γ |= □ empty(cset₂ – cset₁) then [σ]cset₁∪cset₂ = [σ]cset₁.
Proof: By the definition of projection, begin([σ]cset₁∪cset₂) = begin(σ) = begin([σ]cset₁), end([σ]cset₁∪cset₂) = end(σ) = end([σ]cset₁), and for any τ, begin(σ) ≤ τ ≤ end(σ), [σ]cset₁∪cset₂(τ).state = σ(τ).state = [σ]cset₁(τ).state. Then we only have to prove, for any τ, begin(σ) ≤ τ < end(σ), [σ]cset₁∪cset₂(τ).comm = [σ]cset₁(τ).comm. Since cset₁ ∪ cset₂ = cset₁ ∪ (cset₂ – cset₁), we obtain [σ]cset₁∪cset₂ = [σ]cset₁∪(cset₂–cset₁) and then [σ]cset₁∪cset₂(τ).comm = [σ]cset₁∪(cset₂–cset₁)(τ).comm = [σ]cset₁(τ).comm ∪ [σ](cset₂–cset₁)(τ).comm. We show that [σ](cset₂–cset₁)(τ).comm = ϕ.
Assume (σ, begin(σ))γ |= □ empty(cset₂ – cset₁). For any c ∈ cset₂ – cset₁, (σ, begin(σ))γ |= □ ¬comm(c). Thus, for all τ, begin(σ) ≤ τ < end(σ), and for all value θ ∈ VAL, (c, θ) ∈ σ(τ).comm. Similarly, for any c! ∈ cset₂ – cset₁, (σ, begin(σ))γ |= □ ¬wait(c!). Thus, for all τ, begin(σ) ≤ τ < end(σ), c! ∈ σ(τ).comm, and then σ(τ).comm ∩ {c! ∣ c! ∈ cset₂ – cset₁} = ϕ. We can also derive that σ(τ).comm ∩ {c? ∣ c? ∈ cset₂ – cset₁} = ϕ. Hence [σ](cset₂–cset₁)(τ).comm = ϕ and then [σ]cset₁∪cset₂(τ).comm = [σ]cset₁(τ).comm.

Proof of Lemma 6.12
Consider a model σ, an environment γ, and two sets vset₁, vset₂ ⊆ VAR. Prove that if (σ, begin(σ))γ |= inv(vset₂ – vset₁) then σ ↠ (vset₁ ∪ vset₂) = σ ↠ vset₁.
Proof: Follow the proof for Lemma 6.11.

Proof of Lemma 6.13
Consider a model σ and an environment γ. Prove that if dch(σ) ⊆ cset and (σ, begin(σ))γ |= WF_cset, then σ is well-formed.
Proof: Assume (σ, begin(σ))γ |= WF_cset. Then (σ, begin(σ))γ |= □ (MinWait_cset ∧ Exclusion_cset ∧ Unique_cset). Hence, for all τ ≥ begin(σ),
1. (σ, τ)γ |= ~ (wait(c!) ∧ wait(c?)), for all {c!, c?} ⊆ cset;
2. (σ, τ)γ |= ~ (comm(c) ∧ wait(c!)), for all {c, c!} ⊆ cset, and (σ, τ)γ |= ~ (comm(c) ∧ wait(c?)), for all {c, c?} ⊆ cset;
3. (σ, τ)γ |= comm(c, vexp₁) ∧ comm(c, vexp₂) → vexp₁ = vexp₂, for all c ∈ cset.
Given the interpretation of assertions (Section 4), this implies, for all τ ≥ begin(σ),
1. ~ (c! ∈ σ(τ).comm ∧ c? ∈ σ(τ).comm), for all {c!, c?} ⊆ cset;
2. There does not exist a value $\vartheta \in \text{VAL}$ such that

$$(c, \vartheta) \in \sigma(\tau).\text{comm} \land c! \in \sigma(\tau).\text{comm} \lor (c, \vartheta) \in \sigma(\tau).\text{comm} \land c? \in \sigma(\tau).\text{comm}.$$ 

Thus, for any value $\vartheta \in \text{VAL}$,

$$\neg((c, \vartheta) \in \sigma(\tau).\text{comm} \land c! \in \sigma(\tau).\text{comm}), \text{ for all } \{c, c!\} \subseteq \text{eset}, \text{ and}$$

$$\neg((c, \vartheta) \in \sigma(\tau).\text{comm} \land c? \in \sigma(\tau).\text{comm}), \text{ for all } \{c, c?\} \subseteq \text{eset};$$

3. $(e, \mathcal{V}(\text{vexp}_1)\gamma(\sigma, \tau)) \in \sigma(\tau).\text{comm} \land (e, \mathcal{V}(\text{vexp}_2)\gamma(\sigma, \tau)) \in \sigma(\tau).\text{comm}$

$\Rightarrow \mathcal{V}(\text{vexp}_1)\gamma(\sigma, \tau) = \mathcal{V}(\text{vexp}_2)\gamma(\sigma, \tau)$, for all $e \in \text{eset}$. Since $\text{vexp}_1$ and $\text{vexp}_2$ are arbitrary expressions of type $\text{VAL}$, let $\vartheta_1, \vartheta_2 \in \text{VAL}$ be such that $\vartheta_1 \equiv \text{vexp}_1$ and $\vartheta_2 \equiv \text{vexp}_2$. Hence $\vartheta_1 = \mathcal{V}(\text{vexp}_1)\gamma(\sigma, \tau)$ and $\vartheta_2 = \mathcal{V}(\text{vexp}_2)\gamma(\sigma, \tau)$. Thus, for all $\tau \geq \text{begin}(\sigma)$,

$$(c, \vartheta_1) \in \sigma(\tau).\text{comm} \land (c, \vartheta_2) \in \sigma(\tau).\text{comm} \Rightarrow \vartheta_1 = \vartheta_2, \text{ for all } e \in \text{eset}.$$ 

Notice that if $c! \notin \text{eset}$ then, by $dch(\sigma) \subseteq \text{eset}$, $c! \notin dch(\sigma)$, and thus $c! \notin \sigma(\tau).\text{comm}$, for all $\tau$, $\text{begin}(\sigma) \leq \tau < \text{end}(\sigma)$. Similarly, if $c? \notin \text{eset}$ then $c? \notin \sigma(\tau).\text{comm}$ and if $c \notin \text{eset}$ then for any value $\vartheta \in \text{VAL}$ $(c, \vartheta) \notin \sigma(\tau).\text{comm}$. Thus, for all $c \in \text{CHAN}$, for all values $\vartheta, \vartheta_1, \vartheta_2 \in \text{VAL}$, and for all $\tau$, $\text{begin}(\sigma) \leq \tau < \text{end}(\sigma)$:

1. $\neg(c! \in \sigma(\tau).\text{comm} \land c? \in \sigma(\tau).\text{comm});$

2. $\neg((c, \vartheta) \in \sigma(\tau).\text{comm} \land c! \in \sigma(\tau).\text{comm}) \text{ and } \neg((c, \vartheta) \in \sigma(\tau).\text{comm} \land c? \in \sigma(\tau).\text{comm});$

3. $(e, \vartheta_1) \in \sigma(\tau).\text{comm} \land (e, \vartheta_2) \in \sigma(\tau).\text{comm} \Rightarrow \vartheta_1 = \vartheta_2.$

Hence $\sigma$ is well-formed.  

\[ \blacksquare \]
Appendix B

Soundness Proof

In the soundness proof, we must show that every axiom in our proof system is indeed valid and every inference rule preserves validity, i.e. if the hypotheses of an inference rule are valid, so is the conclusion.

Well-Formedness

Consider a program S and a finite set cset ⊆ DCHAN. We prove that S sat WF_{cset} is valid. Consider any σ ∈ M(S) and any environment γ. Then, by Lemma 3.12, σ is well-formed, that is, for all τ, begin(σ) ≤ τ < end(σ), for all c ∈ CHAN, and for all ™1, ™2, ™ ∈ VAL:

1. ¬(c! ∈ σ(τ).comm ∧ ™ ∈ σ(τ).comm);
2. ¬((c, ™) ∈ σ(τ).comm ∧ ™! ∈ σ(τ).comm) and ¬((c, ™) ∈ σ(τ).comm ∧ ™ ∈ σ(τ).comm);
3. (c, ™1) ∈ σ(τ).comm ∧ (c, ™2) ∈ σ(τ).comm → ™1 = ™2.

For any expressions vexp1 and vexp2 of type VAL, we have V(vexp1)γ(σ, τ) ∈ VAL and V(vexp2)γ(σ, τ) ∈ VAL, for any τ, begin(σ) ≤ τ < end(σ). Since ™1 and ™2 are arbitrary values in VAL, we can replace ™1 and ™2 by V(vexp1)γ(σ, τ) and V(vexp2)γ(σ, τ), respectively. Thus, for all τ, begin(σ) ≤ τ < end(σ), for all ™ ∈ VAL, and for all expressions vexp1, vexp2:

1. ¬(c! ∈ σ(τ).comm ∧ ™ ∈ σ(τ).comm), for all c with {c!, ™} ⊆ cset;
2. ¬((c, ™) ∈ σ(τ).comm ∧ ™! ∈ σ(τ).comm), for all c with {c, ™!} ⊆ cset, and ¬((c, ™) ∈ σ(τ).comm ∧ ™ ∈ σ(τ).comm), for all c with {c, ™} ⊆ cset;
3. (c, V(vexp1)γ(σ, τ)) ∈ σ(τ).comm ∧ (c, V(vexp2)γ(σ, τ)) ∈ σ(τ).comm → V(vexp1)γ(σ, τ) = V(vexp2)γ(σ, τ), for all c ∈ cset.

By the interpretation of assertions, we obtain that, for all τ, begin(σ) ≤ τ < end(σ), for all ™ ∈ VAL, and for all vexp1 and vexp2:

1. ⟨σ, τ⟩γ ⊨ Λ_{c, ™!}∈cset ¬(wait(c!)) ∧ wait(™!));
2. ⟨σ, τ⟩γ ⊨ Λ_{c, ™}∈cset ¬(comm(c) ∧ wait(c!)) ∧ Λ_{c, ™!}∈cset ¬(comm(c) ∧ wait(™!));
3. ⟨σ, τ⟩γ ⊨ Λ_{c∈cset} comm(c, vexp1) ∧ comm(c, vexp2) → vexp1 = vexp2.

Furthermore, for all τ′ ≥ end(σ), for all c ∈ cset, and for all vexp, we have ⟨σ, τ′⟩γ ⊨ ¬wait(c!) ∧ ¬wait(™!) ∧ ¬comm(c) ∧ ¬comm(c, vexp). Thus, for all τ ≥ begin(σ), and for all vexp1 and vexp2:

1. ⟨σ, τ⟩γ ⊨ Λ_{c, ™!}∈cset ¬(wait(c!)) ∧ wait(™!));
2. ⟨σ, τ⟩γ ⊨ Λ_{c, ™}∈cset ¬(comm(c) ∧ wait(c!)) ∧ Λ_{c, ™!}∈cset ¬(comm(c) ∧ wait(™!));
3. ⟨σ, τ⟩γ ⊨ Λ_{c∈cset} comm(c, vexp1) ∧ comm(c, vexp2) → vexp1 = vexp2.

Thus, by definition, ⟨σ, begin(σ)⟩γ ⊨ □(MinWait_{cset} ∧ Exclusion_{cset} ∧ Unique_{cset}) and then ⟨σ, begin(σ)⟩γ ⊨ WF_{cset}. Hence, S sat WF_{cset} is indeed valid.
Communication Invariance

Consider a program $S$ and a set $cset \subseteq DCHAN$ such that $cset \cap dch(S) = \emptyset$. We prove that $S$ sat $\Box \emptyset(cset)$ is valid. Consider any $\sigma \in \mathcal{M}(S)$ and any environment $\gamma$. From Theorem 3.12, we obtain that $dch(\sigma) \subseteq dch(S)$ and then $cset \cap dch(\sigma) = \emptyset$. Thus, by definition of $dch(\sigma)$, for all $\tau$, $\begin{vmatrix} \text{begin}(\sigma) \leq \tau < \text{end}(\sigma) \end{vmatrix}$:

1. If $c \in cset$ then there does not exist any value $\vartheta$ such that $(c, \vartheta) \in \sigma(\tau).\text{comm}$;
2. If $c! \in cset$ then $c! \notin \sigma(\tau).\text{comm}$;
3. If $c? \in cset$ then $c? \notin \sigma(\tau).\text{comm}$.

Thus, for all $\tau$, $\begin{vmatrix} \text{begin}(\sigma) \leq \tau < \text{end}(\sigma) \end{vmatrix}$:

1. $\langle \sigma, \tau \rangle \gamma \models \neg \text{comm}(c)$, for all $c \in cset$;
2. $\langle \sigma, \tau \rangle \gamma \models \neg \text{wait}(c!)$, for all $c! \in cset$;
3. $\langle \sigma, \tau \rangle \gamma \models \neg \text{wait}(c?)$, for all $c? \in cset$.

Furthermore, for any $c \in CHAN$, for all $\tau' \geq \text{end}(\sigma)$, we have

$\langle \sigma, \tau' \rangle \gamma \models \neg \text{comm}(c) \land \neg \text{wait}(c!) \land \neg \text{wait}(c?)$. Thus, for all $\tau \geq \text{begin}(\sigma)$, $\langle \sigma, \tau \rangle \gamma \models \emptyset(cset)$ and then $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \square \emptyset(cset)$. Hence, $S$ sat $\Box \emptyset(cset)$ is valid.

Variable Invariance

Consider a program $S$ and any program variable $x$ such that $x \notin \text{wvar}(S)$. We prove that $S$ sat $x = v \rightarrow \Box(x = v)$ is valid. Consider any $\sigma \in \mathcal{M}(S)$ and any environment $\gamma$. Assume $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models x = v$. Then $\begin{vmatrix} \forall(x) \gamma(\sigma, \text{begin}(\sigma)) = \gamma(v) \end{vmatrix}$. Using Lemma 6.1, $\sigma(\text{begin}(\sigma)).\text{state}(x) = \mathcal{E}(x)(\sigma(\text{begin}(\sigma)).\text{state}) = \forall(x) \gamma(\sigma, \text{begin}(\sigma)) = \gamma(v)$. By the induction on the structure of $S$, we can prove that, for all $\tau$, $\begin{vmatrix} \text{begin}(\sigma) \leq \tau \leq \text{end}(\sigma) \end{vmatrix}$, for all $x \notin \text{wvar}(S)$, $\sigma(\tau).\text{state}(x) = \sigma(\text{begin}(\sigma)).\text{state}(x)$. Thus, for all $\tau$, $\begin{vmatrix} \text{begin}(\sigma) \leq \tau \leq \text{end}(\sigma) \end{vmatrix}$, $\begin{vmatrix} \forall(x) \gamma(\sigma, \tau) = \sigma(\tau).\text{state}(x) = \sigma(\text{begin}(\sigma)).\text{state}(x) = \gamma(v) \end{vmatrix}$ and then $\langle \sigma, \tau \rangle \gamma \models x = v$. For all $\tau' > \text{end}(\sigma)$, $\begin{vmatrix} \forall(x) \gamma(\sigma, \tau') = \sigma(\text{end}(\sigma)).\text{state}(x) = \gamma(v) \end{vmatrix}$ and then $\langle \sigma, \tau' \rangle \gamma \models x = v$. Hence $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \Box(x = v)$ and $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models x = v \rightarrow \Box(x = v)$.

Conjunction

Consider a program $S$ such that $S$ sat $\varphi_1$ and $S$ sat $\varphi_2$ are valid. We prove that $S$ sat $\varphi_1 \land \varphi_2$ is also valid. Consider any $\sigma \in \mathcal{M}(S)$ and any environment $\gamma$. Since $S$ sat $\varphi_1$ is valid, we obtain $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1$. Similarly, $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_2$. Hence $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \land \varphi_2$.

Consequence

Consider a program $S$ such that $S$ sat $\varphi_1$ and $\varphi_1 \rightarrow \varphi_2$ are valid. We prove that $S$ sat $\varphi_2$ is also valid. Since $S$ sat $\varphi_1$ is valid, we obtain $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1$. By the implication, $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_2$. Thus $S$ sat $\varphi_2$ is valid.

34
Skip

We prove that the Skip Axiom (Axiom 5.6) is valid. Consider any \( \sigma \in \mathcal{M}(\text{skip}) \) and any environment \( \gamma \). Then \( \text{begin}(\sigma) = \text{end}(\sigma) \) and hence \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \text{term} = \text{start} \).

Assignment

We prove that the Assignment Axiom (Axiom 5.7) is valid. Consider any \( a \in \mathcal{M}(x := e) \) and any environment \( \gamma \). Assume \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models x = v \). We show that \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models a(T) \text{.state} = a(\text{begin}(\sigma)).\text{state} \) and then \( \langle \sigma, \text{end}(\sigma) \rangle \gamma \models \text{term} = \text{start} \). By the assumption, \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models x = v \). Thus for all \( \tau \), \( \langle \sigma, \tau \rangle \gamma \models \text{term} = \text{start} \). Hence \( \langle \sigma, \text{end}(\sigma) \rangle \gamma \models \text{term} = \text{start} \).

Delay

We prove that the Delay Axiom (Axiom 5.8) is valid. Consider any \( \sigma \in \mathcal{M}(\text{delay } e) \) and any environment \( \gamma \). By Lemma 6.1, \( \mathcal{E}(\text{term}) = \mathcal{E}(\text{delay } e) \). Hence \( \langle \sigma, \text{end}(\sigma) \rangle \gamma \models \text{term} = \text{start} + e \). Since \( \sigma \in \mathcal{M}(\text{delay } e), \text{end}(\sigma) = \text{begin}(\sigma) + \mathcal{E}(\text{delay } e), \text{end}(\sigma) \), we obtain that \( \langle \sigma, \text{end}(\sigma) \rangle \gamma \models \text{term} = \text{start} + e \). Hence, \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models x = v \). Thus, \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models x = v \).

Output

We prove that the Output Axiom (Axiom 5.9) is valid. Consider any \( \sigma \in \mathcal{M}(\text{e!}) \) and any environment \( \gamma \). Then,

1. Either \( \text{end}(\sigma) = \infty \) and \( \sigma \in \text{Wait}(\text{e!}) \), i.e., for all \( \tau \geq \text{begin}(\sigma), \sigma(\tau).\text{comm} = \{\text{e!}\}; \)
2. Or there exist models \( \sigma_1 \) and \( \sigma_2 \) such that \( \sigma = \sigma_1 \sigma_2, \sigma_1 \in \text{Wait}(\text{e!}), \sigma_2 \in \text{Send}(\text{e}, \text{e}), \) and \( \text{end}(\sigma_1) < \infty \). Then, there exists a \( \tau \in \text{TIME} \) such that, \( \text{end}(\sigma_1) = \tau, \) for all \( \tau_1, \text{begin}(\sigma_1) \leq \tau_1 < \text{end}(\sigma_1), \sigma_1(\tau_1).\text{state} = \text{e!}, \sigma_1(\tau_1).\text{comm} = \{\text{e!}\}, \sigma_1(\text{end}(\sigma_1)).\text{state} = \text{begin}(\sigma_1)).\text{state} \), and then \( \text{end}(\sigma_2) = \text{begin}(\sigma_2) + K_e, \) for all \( \tau_2, \text{begin}(\sigma_2) \leq \tau_2 < \text{end}(\sigma_2), \sigma_2(\tau_2).\text{comm} = \{\text{e!}\}, \sigma_2(\text{end}(\sigma_2)).\text{state} = \text{begin}(\sigma_2)).\text{state} \), and then \( \text{end}(\sigma_2) = \text{begin}(\sigma_2)).\text{state} \).

That is,

1. Either \( \text{end}(\sigma) = \infty \) and, for all \( \tau \geq \text{begin}(\sigma), \langle \sigma, \tau \rangle \gamma \models \text{wait}(\text{e!}) \). Hence \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \square \text{wait}(\text{e!}); \)
2. Or, from \( \sigma = \sigma_1 \sigma_2 \), we can derive that there exists a \( \tau \in \text{TIME} \) such that, for all \( \tau_1 \), \( \text{begin}(\sigma) \leq \tau_1 < \tau \), \( \langle \sigma, \tau_1 \rangle \gamma \models \text{wait}(c!) \). Since \( \text{end}(\sigma_1) \leq \infty \), we obtain \( \text{begin}(\sigma_2) = \text{end}(\sigma_1) = \tau \). By Lemma 6.1, for all \( \tau_2 \), \( \tau \leq \tau_2 < \text{end}(\sigma), \mathcal{E}(e)(\text{begin}(\sigma_2)).\text{state} = \mathcal{V}(e)\gamma(\sigma_2, \tau_2) \). Thus we have \( \langle \sigma, \tau_2 \rangle \gamma \models \text{comm}(c, e) \). Since \( \text{end}(\sigma_2) = \text{begin}(\sigma_2) + K_c \), we obtain \( \text{end}(\sigma) = \tau + K_c \) and then \( \langle \sigma, \tau \rangle \gamma \models T = \text{term} - K_c \) as well as \( \langle \sigma, \text{end}(\sigma) \rangle \gamma \models T = \text{term} \). Hence \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \text{wait}(c!) \cup (T = \text{term} - K_c \land (\text{comm}(c, e) \cup T = \text{term})) \).

Hence \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \text{wait}(c!) \cup (T = \text{term} - K_c \land (\text{comm}(c, e) \cup T = \text{term})) \).

**Input**

We prove that the Input Axiom (Axiom 5.10) is valid. Consider any \( \sigma \in M(c?x) \) and any environment \( \gamma \). Assume \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \text{wait}(c!) \) and \( \sigma(\tau).\text{state} = \{c?\} \), and \( \sigma(\tau).\text{state} = \sigma(\text{begin}(\sigma)).\text{state} \).

2. Or there exist models \( \sigma_1 \) and \( \sigma_2 \) such that \( \sigma = \sigma_1 \sigma_2, \sigma_1 \in \text{Wait}(c?), \sigma_2 \in \text{Receive}(c, x) \), and \( \text{end}(\sigma_1) < \infty \). Thus, there exists a \( \tau \in \text{TIME} \) such that, \( \text{end}(\sigma_1) = \tau \), for all \( \tau_1, \text{begin}(\sigma_1) \leq \tau_1 < \text{end}(\sigma_1), \sigma_1(\tau_1).\text{state} = \sigma_1(\text{begin}(\sigma_1)).\text{state}, \sigma_1(\tau_1).\text{comm} = \{c?\}, \sigma_1(\text{end}(\sigma_1)).\text{state} = \sigma_1(\text{begin}(\sigma_2)).\text{state}, \text{end}(\sigma_2) = \text{begin}(\sigma_2) + K_c \), there exists a value \( T \in \text{VAL} \) such that, for all \( \tau_2, \text{begin}(\sigma_2) \leq \tau_2 < \text{end}(\sigma_2), \sigma_2(\tau_2).\text{comm} = \{(c, \theta)\}, \sigma_2(\tau_2).\text{state} = \sigma_2(\text{begin}(\sigma_2)).\text{state}, \text{and} \sigma_2(\text{end}(\sigma_2)).\text{state} = \sigma_2(\text{begin}(\sigma_2)).\text{state} : x \mapsto \theta \).

That is,

1. Either \( \text{end}(\sigma) = \infty \), for all \( \tau \geq \text{begin}(\sigma), \langle \sigma, \tau \rangle \gamma \models \text{wait}(c!) \) and \( \langle \sigma, \tau \rangle \gamma \models \text{wait}(c!) \). Hence, \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \Box (\text{wait}(c!)) \).

2. Or, from \( \sigma = \sigma_1 \sigma_2 \), we obtain \( \text{begin}(\sigma_2) = \text{end}(\sigma_1) = \tau \). Thus for all \( \tau_1, \text{begin}(\sigma) \leq \tau_1 < \tau, \langle \sigma, \tau_1 \rangle \gamma \models \text{wait}(c!) \), for all \( \tau_2, \tau \leq \tau_2 < \text{end}(\sigma), \langle \sigma, \tau_2 \rangle \gamma \models \text{wait}(c!) \). Since \( \text{end}(\sigma_2) = \text{begin}(\sigma_2) + K_c \), we obtain \( \text{end}(\sigma) = \tau + K_c \) and then \( \langle \sigma, \tau \rangle \gamma \models T = \text{term} \). Hence we have \( \langle \sigma, \tau \rangle \gamma \models T = \text{term} - K_c \land (x = v \land \text{comm}(c, x) = \theta) \). We can also derive that, for all \( \tau_2, \tau \leq \tau_2 < \text{end}(\sigma), \langle \sigma, \tau \rangle \gamma \models (x = v \land \text{comm}(c, last(x))) \cup T = \text{term} \). Therefore \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models (x = v \land \text{wait}(c!)) \cup (T = \text{term} - K_c \land (x = v \land \text{comm}(c, last(x)))) \cup T = \text{term}) \).

Hence \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models x = v \rightarrow [(x = v \land \text{wait}(c!)) \cup (T = \text{term} - K_c \land (x = v \land \text{comm}(c, last(x)))) \cup T = \text{term})] \).
Sequential Composition

We prove that the Sequential Composition Rule (Rule 5.11) preserves validity. Assume \( S_1 \) sat \( \varphi_1 \) and \( S_2 \) sat \( \varphi_2 \) are valid. We show that \( S_1; S_2 \) sat \( \varphi_1 \wedge \varphi_2 \) is also valid. Consider any \( \sigma \in \mathcal{M}(S_1; S_2) \) and any environment \( \gamma \). Then there exist \( \sigma_1 \in \mathcal{M}(S_1) \) and \( \sigma_2 \in \mathcal{M}(S_2) \) such that \( \sigma = \sigma_1 \sigma_2 \). By definition, \( \text{end}(\sigma_1) \geq \text{begin}(\sigma_1) \). From \( S_1 \) sat \( \varphi_1 \) and \( S_2 \) sat \( \varphi_2 \), we obtain \( \langle \sigma_1, \text{begin}(\sigma_1) \rangle \gamma \models \varphi_1 \) and \( \langle \sigma_2, \text{begin}(\sigma_2) \rangle \gamma \models \varphi_2 \). By the definition of the \( \wedge \) operator, we can derive \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \wedge \varphi_2 \), i.e., \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \wedge \varphi_2 \). Hence, \( S_1; S_2 \) sat \( \varphi_1 \wedge \varphi_2 \) is indeed valid.

Guarded Command with Purely Boolean Guards

Consider \( G \equiv [\bigwedge_{i=1}^{n} b_i \rightarrow S_i] \). We prove that the Guarded Command Evaluation Axiom (Axiom 5.12) is valid for \( G \). Let \( \sigma \in \mathcal{M}(G) \) and \( \gamma \) be any environment. There are two possibilities:

1. Either \( B(\neg b_G)(\sigma(\text{begin}(\sigma)).\text{state}) \) and \( \sigma \in \mathcal{M}(\text{delay } K_g) \);
2. Or there exists a \( k, 1 \leq k \leq n \), such that \( B(b_k)(\sigma(\text{begin}(\sigma)).\text{state}) \) and \( \sigma \in \mathcal{M}(\text{delay } K_g; S_k) \).

That is,

1. Either, from \( B(\neg b_G)(\sigma(\text{begin}(\sigma)).\text{state}) \) by Lemma 6.2, \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \neg b_G \). Since \( \sigma \in \mathcal{M}(\text{delay } K_g) \), \( \text{end}(\sigma) = \text{begin}(\sigma) + K_g \) and then \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \text{term} = \text{start} + K_g \). Hence, \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \neg b_G \rightarrow \text{term} = \text{start} + K_g \). Assume \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \bigwedge_{y \in \text{var}(G)} y = v_y \). From the semantics, for all \( \tau_1 \), \( \text{begin}(\sigma) \leq \tau_1 \leq \text{end}(\sigma) \), \( \sigma(\tau_1).\text{state} = \sigma(\text{begin}(\sigma)).\text{state} \) and then \( \langle \sigma, \tau_1 \rangle \gamma \models \bigwedge_{y \in \text{var}(G)} y = v_y \). Also, for all \( \tau_2 \), \( \text{begin}(\sigma) \leq \tau_2 < \text{end}(\sigma) \), \( \sigma(\tau_2).\text{comm} = \emptyset \). Thus,
   - for all \( c \in \text{dch}(G) \), \( c \notin \sigma(\tau_2).\text{comm} \);
   - for all \( c \in \text{dch}(G) \), \( c \notin \sigma(\tau_2).\text{comm} \);
   - for all \( c \in \text{dch}(G) \), there does not exist any value \( \vartheta \) such that \( (c, \vartheta) \in \sigma(\tau_2).\text{comm} \).

Then \( \langle \sigma, \tau_2 \rangle \gamma \models \bigwedge_{c \in \text{dch}(G)} \neg \text{wait}(c) \wedge \bigwedge_{c \in \text{dch}(G)} \neg \text{wait}(c) \wedge \bigwedge_{c \in \text{dch}(G)} \neg \text{comm}(c) \), i.e. \( \langle \sigma, \tau_2 \rangle \gamma \models \text{empty}(\text{dch}(G)) \). We can also derive that \( \langle \sigma, \text{end}(\sigma) \rangle \gamma \models \text{term} = \text{start} + K_g \). Hence, \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \bigwedge_{y \in \text{var}(G)} y = v_y \wedge \text{empty}(\text{dch}(G)) \) \( \cup \) \( \text{term} = \text{start} + K_g \wedge \bigwedge_{y \in \text{var}(G)} y = v_y \). Thus, \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \text{Quiet} \cup \text{term} = \text{start} + K_g \wedge \bigwedge_{y \in \text{var}(G)} y = v_y \). Hence, \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \bigwedge_{y \in \text{var}(G)} y = v_y \rightarrow (\text{Quiet} \cup \text{term} = \text{start} + K_g \wedge \bigwedge_{y \in \text{var}(G)} y = v_y) \wedge (\neg b_G \rightarrow \text{term} = \text{start} + K_g) \);

2. Or, by \( B(b_k)(\sigma(\text{begin}(\sigma)).\text{state}) \), we obtain \( B(b_G)(\sigma(\text{begin}(\sigma)).\text{state}) \) and then \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models b_G \). Then \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models b_G \rightarrow \text{term} = \text{start} + K_g \). Since \( \sigma \in \mathcal{M}(\text{delay } K_g; S_k) \), there exist models \( \sigma_1 \in \mathcal{M}(\text{delay } K_g) \) and \( \sigma_2 \in \mathcal{M}(S_k) \) such that \( \sigma = \sigma_1 \sigma_2 \). From \( \sigma_1 \in \mathcal{M}(\text{delay } K_g) \), we can obtain the same result as previous case, i.e. \( \langle \sigma_1, \text{begin}(\sigma_1) \rangle \gamma \models \bigwedge_{y \in \text{var}(G)} y = v_y \rightarrow (\text{Quiet} \cup \text{term} = \text{start} + K_g \wedge \bigwedge_{y \in \text{var}(G)} y = v_y) \). Thus, \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \bigwedge_{y \in \text{var}(G)} y = v_y \rightarrow (\text{Quiet} \cup \text{term} = \text{start} + K_g \wedge \bigwedge_{y \in \text{var}(G)} y = v_y) \wedge (\neg b_G \rightarrow \text{term} = \text{start} + K_g) \).
Hence, we can conclude that the Guarded Command Evaluation Axiom is indeed valid for $G \equiv [\bigotimes_{i=1}^{n} b_i \rightarrow S_i]$.

Now we prove that the Guarded Command with Purely Boolean Guards Rule (Rule 5.13) preserves validity. Assume $S_i \text{ sat } \varphi_i$ are valid, $i = 1, 2, \ldots, n$. Consider any $\sigma \in \mathcal{M}(G)$ and any environment $\gamma$. Then,

1. If $B(b_G)(\sigma(\text{begin}(\sigma)).\text{state}), \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \neg b_G$ and then $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models b_G \rightarrow (\text{term} = \text{start} + K_g) \cup \bigotimes_{i=1}^{n} b_i \land \varphi_i$;
2. If $B(b_G)(\sigma(\text{begin}(\sigma)).\text{state})$ and then $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models b_G$.

Since $\sigma \in \mathcal{M}(\text{delay } K_g; S_k)$, there exist models $\sigma_1 \in \mathcal{M}(\text{delay } K_g)$ and $\sigma_2 \in \mathcal{M}(S_k)$ such that $\sigma = \sigma_1 \sigma_2$. Thus $\text{end}(\sigma_1) = \text{begin}(\sigma_1) + K_g$ and hence $\langle \sigma_1, \text{begin}(\sigma_1) \rangle \gamma \models \text{term} = \text{start} + K_g$. From the assumption, $S_i \text{ sat } \varphi_i$ are valid, $i = 1, 2, \ldots, n$. Since $\sigma_2 \in \mathcal{M}(S_k)$, $\langle \sigma_2, \text{begin}(\sigma_2) \rangle \gamma \models \varphi_k$. From $B(b_G)(\sigma(\text{begin}(\sigma)).\text{state})$, we obtain $B(b_G)(\sigma_2(\text{begin}(\sigma_2)).\text{state})$ and then $\langle \sigma_2, \text{begin}(\sigma_2) \rangle \gamma \models b_G$. Thus $\langle \sigma_2, \text{begin}(\sigma_2) \rangle \gamma \models b_G \land \varphi_k$ and hence $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \text{term} = \text{start} + K_g \cup \bigotimes_{i=1}^{n} (b_i \land \varphi_i)$. From the definition of the $C$ operator, $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models (\text{term} = \text{start} + K_g) \cup \bigotimes_{i=1}^{n} (b_i \land \varphi_i)$.

Then we can conclude that the Rule 5.13 preserves validity.

**Guarded Command with IO-guards**

Now we consider $G \equiv [\bigotimes_{i=1}^{n} b_i; c_i?x_i \rightarrow S_i \parallel b; \text{delay } e \rightarrow S]$. We first prove that the Guarded Command Evaluation Axiom (Axiom 5.12) is also valid for $G$. Let $\sigma \in \mathcal{M}(G)$ and $\gamma$ be any environment. There are four possibilities:

1. $B(\neg b_G)(\sigma(\text{begin}(\sigma)).\text{state})$ and $\sigma \in \mathcal{M}(\text{delay } K_g)$;
2. Or $\sigma \in \text{SEQ}(\mathcal{M}(\text{delay } K_g), \text{BoundWait}(G), \text{Comm}(G))$;
3. Or $\sigma \in \text{SEQ}(\mathcal{M}(\text{delay } K_g), \text{Timeout}(G), \mathcal{M}(S))$;
4. Or $\sigma \in \text{SEQ}(\mathcal{M}(\text{delay } K_g), \text{AnyWait}(G), \text{Comm}(G))$.

Following the proof of Axiom 5.12 for the case $G \equiv [\bigotimes_{i=1}^{n} b_i; c_i?x_i \rightarrow S_i]$, we can conclude that the Axiom 5.12 is also valid for $G \equiv [\bigotimes_{i=1}^{n} b_i; c_i?x_i \rightarrow S_i \parallel b; \text{delay } e \rightarrow S]$.

Next we prove that the Guarded Command with IO-guards Rule (Rule 5.14) preserves validity. Assume $c_i?x_i; S_i \text{ sat } \varphi_i, i = 1, 2, \ldots, n$ and $S \text{ sat } \varphi$ are valid. Then,

1. If $B(\neg b_G)(\sigma(\text{begin}(\sigma)).\text{state})$, then $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \neg b_G$. Thus $\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \Lambda_{y \in \text{var}(G)} y = v_y \land b_G \rightarrow \text{Eval} C (\text{Comm} \lor \text{Timeout})$.
2. If $\sigma \in \text{SEQ}(\mathcal{M}(\text{delay } K_g), \text{BoundWait}(G), \text{Comm}(G))$, there exist models $\sigma_1 \in \mathcal{M}(\text{delay } K_g), \sigma_2 \in \text{BoundWait}(G)$, and $\sigma_3 \in \text{Comm}(G)$ such that $\sigma = \sigma_1 \sigma_2 \sigma_3$. From $\sigma_1 \in \mathcal{M}(\text{delay } K_g)$, $\text{end}(\sigma_1) = \text{begin}(\sigma_1) + K_g$ and then $\langle \sigma_1, \text{begin}(\sigma_1) \rangle \gamma \models \text{term} = \text{start} + K_g$; i.e., $\langle \sigma_1, \text{begin}(\sigma_1) \rangle \gamma \models \text{Eval}$. From $\sigma_2 \in \text{BoundWait}(G)$, we obtain $\text{end}(\sigma_2) < \text{begin}(\sigma_2) + \mathcal{E}(e)(\sigma_2(\text{begin}(\sigma_2)).\text{state})$, 

38
3. If \( B(b)(\sigma_2(begin(\sigma_2)).state) \), for all \( \tau_2, begin(\sigma_2) \leq \tau_2 < end(\sigma_2), \sigma_2(\tau_2).state = \sigma_2(begin(\sigma_2)).state, \sigma_2(\tau_2).comm = \{ c_i^? \ | \ B(b)(\sigma_2(begin(\sigma_2)).state), 1 \leq i \leq n \} \), and \( \sigma_2(end(\sigma_2)).state = \sigma_2(begin(\sigma_2)).state \). Assume \( \langle \sigma, begin(\sigma) \rangle \gamma \models \bigwedge_{y \in var(G)} y = v_y \). Then \( \langle \sigma_2, begin(\sigma_2) \rangle \gamma \models \bigwedge_{y \in var(G)} y = v_y \). For all \( \tau_2, begin(\sigma_2) \leq \tau_2 \leq end(\sigma_2), \langle \sigma_2, \tau_2 \rangle \gamma \models \bigwedge_{y \in var(G)} y = v_y \). For all \( \tau_2, begin(\sigma_2) \leq \tau_2 < end(\sigma_2) \), we can derive that \( c_i \in \sigma_2(\tau_2).comm \) iff \( B(b_i)(\sigma_2(begin(\sigma_2)).state) \), for all \( i, 1 \leq i \leq n \). Then \( \langle \sigma_2, \tau_2 \rangle \gamma \models wait(c_i) \) iff \( \langle \sigma_2, begin(\sigma_2) \rangle \gamma \models b_i \) iff \( \langle \sigma_2, \tau_2 \rangle \gamma \models b_i \). Thus \( \langle \sigma_2, \tau_2 \rangle \gamma \models \bigwedge_{i=1}^n b_i \leftrightarrow wait(c_i) \). Since \( \sigma \in \mathcal{M}(G) \), by Theorem 3.12, \( \sigma \) is well-formed. By the definition of well-formedness, we can obtain \( \langle \sigma_2, \tau_2 \rangle \gamma \models empty(dch(G) - \{ c_1^?, \ldots, c_n^? \}) \). From \( end(\sigma_2) < begin(\sigma_2) + E(e)(\sigma_2(begin(\sigma_2)).state) \), for all \( \tau_2, begin(\sigma_2) \leq \tau_2 \leq end(\sigma_2), \langle \sigma_2, \tau_2 \rangle \gamma \models T < start + e \). From \( B(b)(\sigma_2(begin(\sigma_2)).state) \), \( \langle \sigma_2, \tau_2 \rangle \gamma \models b \) and then \( \langle \sigma, begin(\sigma) \rangle \gamma \models b_G \). Thus \( \langle \sigma_2, \tau_2 \rangle \gamma \models b \rightarrow T < start + e \). It is obvious that \( \langle \sigma_2, end(\sigma_2) \rangle \gamma \models T = term \). Hence \( \langle \sigma_2, begin(\sigma_2) \rangle \gamma \models (\bigwedge_{y \in var(G)} y = v_y \land empty(dch(G) - \{ c_1^?, \ldots, c_n^? \}) \land \bigwedge_{i=1}^n b_i \leftrightarrow wait(c_i)) \land (b \rightarrow T < start + e) \) \( \cup \) \( \bigwedge_{y \in var(G)} y = v_y \land T = term \land (b \rightarrow T < start + e) \). Then we obtain \( \langle \sigma_2, begin(\sigma_2) \rangle \gamma \models Wait U InTime \).

From \( \sigma_3 \in Comm(G) \), there exists a \( k, 1 \leq k \leq n, \) such that \( B(b_k)(\sigma_3(begin(\sigma_3)).state) \) and \( \sigma_3 \in SEQ(c_k, x_k, S_k) \). Then \( \langle \sigma_3, begin(\sigma_3) \rangle \gamma \models b_k \). \( \sigma_3 \in \mathcal{M}(c_k, x_k, S_k) \), and \( \langle \sigma_3, begin(\sigma_3) \rangle \gamma \models \text{comm}(c_k) \). By the assumption, \( \sigma \) is valid. Thus \( \langle \sigma_3, begin(\sigma_3) \rangle \gamma \models \varphi_k \) and then \( \langle \sigma_3, begin(\sigma_3) \rangle \gamma \models b_k \land \varphi_k \land \text{comm}(c_k) \). Hence \( \langle \sigma_3, begin(\sigma_3) \rangle \gamma \models \bigwedge_{i=1}^n (b_i \land \varphi_i \land \text{comm}(c_i)) \). Then \( \langle \sigma_2, begin(\sigma_2) \rangle \gamma \models (Wait U InTime) C \bigwedge_{i=1}^n (b_i \land \varphi_i \land \text{comm}(c_i)) \), i.e., \( \langle \sigma_2, begin(\sigma_2) \rangle \gamma \models \text{Comm} \). By \( \sigma = \sigma_1 \sigma_2 \sigma_3 \), \( \langle \sigma, begin(\sigma) \rangle \gamma \models Eval C \text{Comm} \).

Hence \( \langle \sigma, begin(\sigma) \rangle \gamma \models \bigwedge_{y \in var(G)} y = v_y \land b_G \rightarrow Eval C \text{Comm} \).

3. If \( \sigma \in SEQ(M(delay K_2), \text{Timeout}(G), \mathcal{M}(S)) \), there exist models \( \sigma_1 \in \mathcal{M}(delay K_2), \sigma_2 \in \text{Timeout}(G), \) and \( \sigma_3 \in \mathcal{M}(S) \) such that \( \sigma = \sigma_1 \sigma_2 \sigma_3 \). \( \sigma_1 \in \mathcal{M}(delay K_2) \) implies \( \langle \sigma_1, begin(\sigma_1) \rangle \gamma \models Eval \).

\( \sigma_2 \in \text{Timeout}(G) \) implies \( B(b)(\sigma_2(begin(\sigma_2)).state) \) and then \( \langle \sigma_2, begin(\sigma_2) \rangle \gamma \models b \). Hence \( \langle \sigma, begin(\sigma) \rangle \gamma \models b_G \). Since \( E(e)(\sigma_2(begin(\sigma_2)).state) = E(e)(\sigma_2(end(\sigma_2)).state) \), \( \langle \sigma_2, end(\sigma_2) \rangle \gamma \models T = start + e \). Assume \( \langle \sigma, begin(\sigma) \rangle \gamma \models \bigwedge_{y \in var(G)} y = v_y \). We can also derive that, for all \( \tau_2, begin(\sigma_2) \leq \tau_2 < end(\sigma_2), \langle \sigma_2, \tau_2 \rangle \gamma \models empty(dch(G) - \{ c_1^?, \ldots, c_n^? \}) \land \bigwedge_{i=1}^n (b_i \rightarrow wait(c_i)) \land (b \rightarrow T < start + e) \), for all \( \tau_2', begin(\sigma_2) \leq \tau_2' \leq end(\sigma_2), \langle \sigma_2, \tau_2' \rangle \gamma \models \bigwedge_{y \in var(G)} y = v_y \), \( \langle \sigma_2, \tau_2' \rangle \gamma \models b \), and \( \langle \sigma_2, end(\sigma_2) \rangle \gamma \models T = term \land start + e \). Hence, \( \langle \sigma_2, begin(\sigma_2) \rangle \gamma \models Wait U EndTime \).

Since \( S \) is valid, \( \langle \sigma_3, begin(\sigma_3) \rangle \gamma \models \varphi \).

Thus \( \langle \sigma_2 \sigma_3, begin(\sigma_2) \rangle \gamma \models (Wait U EndTime) C \varphi \), i.e., \( \langle \sigma_2 \sigma_3, begin(\sigma_2) \rangle \gamma \models Timeout \). By \( \sigma = \sigma_1 \sigma_2 \sigma_3, \langle \sigma, begin(\sigma) \rangle \gamma \models Eval C Timeout \).

Hence \( \langle \sigma, begin(\sigma) \rangle \gamma \models \bigwedge_{y \in var(G)} y = v_y \land b_G \rightarrow (Eval C Timeout) \);
4. If \( \sigma \in SEQ(M(\text{delay } K_2), \text{AnyWait}(G), \text{Comm}(G)) \), there exist models \\
\( \sigma_1 \in M(\text{delay } K_2) \), \( \sigma_2 \in \text{AnyWait}(G) \), and \( \sigma_3 \in \text{Comm}(G) \) such that \( \sigma = \sigma_1 \sigma_2 \sigma_3 \). \\
\( \sigma_1 \in M(\text{delay } K_2) \) implies \( \langle \sigma_1, \text{begin}(\sigma_1) \rangle \models \text{Eval} \). \\
\( \sigma_2 \in \text{AnyWait}(G) \) implies \( B(\neg b)(\sigma_2(\text{begin}(\sigma_2)).\text{state}) \) and then \( \langle \sigma_2, \text{begin}(\sigma_2) \rangle \models \neg b \). \\
Thus \( \langle \sigma_2, \text{begin}(\sigma_2) \rangle \models b \rightarrow T < \text{start} + e \). From the semantics, we can also obtain \( B(b)(\sigma_2(\text{begin}(\sigma_2)).\text{state}) \) and then \( \langle \sigma_2, \text{begin}(\sigma_2) \rangle \models b \), i.e., \( \langle \sigma, \text{begin}(\sigma) \rangle \models b \). \\
Assume \( \langle \sigma, \text{begin}(\sigma) \rangle \models \bigwedge_{y \in \text{var}(G)} y = v_y \). Similarly as previous case, we can derive that, for all \( \tau_2 \), \( \text{begin}(\sigma_2) \leq \tau_2 < \text{end}(\sigma_2) \), \( \langle \sigma_2, \tau_2 \rangle \models \text{empty}(\text{dch}(G) - \{ c_1, \ldots, c_n \}) \land \bigwedge_{i=1}^{n} (b_i \rightarrow \text{wait}(c_i)) \), for all \( \tau_2 \), \( \text{begin}(\sigma_2) \leq \tau_2 < \text{end}(\sigma_2) \), \( \langle \sigma_2, \tau_2 \rangle \models \bigwedge_{y \in \text{var}(G)} y = v_y \land (b \rightarrow T < \text{start} + e) \), and \( \langle \sigma_2, \text{end}(\sigma_2) \rangle \models T = \text{term} \). If \( \text{end}(\sigma_2) = \infty \), \( \langle \sigma_2, \text{begin}(\sigma_2) \rangle \models \square \text{Wait} \). If \( \text{end}(\sigma_2) < \infty \), \( \langle \sigma_2, \text{begin}(\sigma_2) \rangle \models \text{Wait} \cup \text{InTime} \). \\
Hence, \( \langle \sigma_2, \text{begin}(\sigma_2) \rangle \models \text{Wait} \cup \text{InTime} \). \\
\( \sigma_3 \in \text{Comm}(G) \) implies \( \langle \sigma_3, \text{begin}(\sigma_3) \rangle \models \bigwedge_{i=1}^{n} (b_i \land \phi_i \land \text{comm}(c_i)) \). \\
Thus \( \langle \sigma_2, \sigma_3, \text{begin}(\sigma_2) \rangle \models \left( \text{Wait} \cup \text{InTime} \right) C \bigwedge_{i=1}^{n} (b_i \land \phi_i \land \text{comm}(c_i)) \), i.e., \( \langle \sigma_2, \sigma_3, \text{begin}(\sigma_2) \rangle \models \text{Comm} \). By \( \sigma = \sigma_1 \sigma_2 \sigma_3 \), \( \langle \sigma, \text{begin}(\sigma) \rangle \models \text{Eval} \lor \text{Comm} \). \\
Hence \( \langle \sigma, \text{begin}(\sigma) \rangle \models \bigwedge_{y \in \text{var}(G)} y = v_y \land b \rightarrow \left( \text{Eval} \lor \text{Comm} \right) \). \\
Therefore, the Guarded Command with IO-guards Rule (Rule 5.14) preserves validity.

**Iteration**

We prove that the Iteration Rule (Rule 5.15) preserves validity. Assume \( G \models \varphi \) is valid. We prove that \( *G \models (b_\neg G \land \varphi) C^* (\neg b \land \varphi) \) is also valid. Consider any \( \sigma \in M(*G) \) and any environment \( \gamma \). There are two possibilities:

1. Either there exist a \( k \in \mathbb{N} \), \( k \geq 1 \), and models \( \sigma_1, \sigma_2, \ldots, \sigma_k \) such that \( \sigma = \sigma_1 \sigma_2 \ldots \sigma_k \), for all \( i, 1 \leq i \leq k \), \( \sigma_i \in M(G) \), for all \( j, 1 \leq j \leq k - 1 \), \( \text{end}(\sigma_j) < \infty \), \( B(b_\neg G)(\sigma_j(\text{begin}(\sigma_j)).\text{state}) \), and either \( \text{end}(\sigma_k) = \infty \) and \( B(b_\neg G)(\sigma_k(\text{begin}(\sigma_k)).\text{state}) \) or \( B(\neg b)(\sigma_k(\text{begin}(\sigma_k)).\text{state}) \);

2. Or there exist an infinite sequence of models \( \sigma_1, \sigma_2, \ldots \) such that \( \sigma = \sigma_1 \sigma_2 \ldots \), for all \( i \geq 1 \), \( \sigma_i \in M(G) \), \( \text{end}(\sigma_i) < \infty \), and \( B(b_\neg G)(\sigma_i(\text{begin}(\sigma_i)).\text{state}) \).

Since \( G \models \varphi \) is valid, we obtain \( \langle \sigma_i, \text{begin}(\sigma_i) \rangle \models \varphi \), for all \( \sigma_i \in M(G) \). Then,

1. Either there exist a \( k \in \mathbb{N} \), \( k \geq 1 \), and models \( \sigma_1, \sigma_2, \ldots, \sigma_k \) such that \( \sigma = \sigma_1 \sigma_2 \ldots \sigma_k \), for all \( j, 1 \leq j \leq k - 1 \), \( \langle \sigma_j, \text{begin}(\sigma_j) \rangle \models \varphi \), \( \text{end}(\sigma_j) < \infty \). From \( B(b_\neg G)(\sigma_j(\text{begin}(\sigma_j)).\text{state}) \), by Lemma 6.2, \( \langle \sigma_j, \text{begin}(\sigma_j) \rangle \models b \). Then \( \langle \sigma_j, \text{begin}(\sigma_j) \rangle \models b \land \phi \). If \( \text{end}(\sigma_j) = \infty \), from \( B(b_\neg G)(\sigma_k(\text{begin}(\sigma_k)).\text{state}) \), we obtain \( \langle \sigma_k, \text{begin}(\sigma_k) \rangle \models b \). By \( \langle \sigma_k, \text{begin}(\sigma_k) \rangle \models \varphi \), \( \langle \sigma_k, \text{begin}(\sigma_k) \rangle \models b \land \phi \). If \( \text{end}(\sigma_k) < \infty \), from \( B(\neg b)(\sigma_k(\text{begin}(\sigma_k)).\text{state}) \), we can derive \( \langle \sigma_k, \text{begin}(\sigma_k) \rangle \models \neg b \) and hence \( \langle \sigma_k, \text{begin}(\sigma_k) \rangle \models \neg b \land \phi \);

2. Or there exist an infinite sequence of models \( \sigma_1, \sigma_2, \ldots \) such that \( \sigma = \sigma_1 \sigma_2 \ldots \), for all \( i \geq 1 \), \( \langle \sigma_i, \text{begin}(\sigma_i) \rangle \models \varphi \), \( \text{end}(\sigma_i) < \infty \), and \( \langle \sigma_i, \text{begin}(\sigma_i) \rangle \models b \). Thus, for all \( i \geq 1 \), \( \langle \sigma_i, \text{begin}(\sigma_i) \rangle \models b \land \phi \).

By the definition of the \( C^* \) operator, \( \langle \sigma, \text{begin}(\sigma) \rangle \models (b \land \phi) C^* (\neg b \land \phi) \).
Parallel Composition

We prove the soundness of the General Parallel Composition Rule. Then the soundness of the Simple Parallel Composition Rule follows directly. Assume \( S_1 \text{ sat } \varphi_1, S_2 \text{ sat } \varphi_2 \), \( \psi_1 \equiv \text{inv}(\text{var}(S_2)) \land \square \text{empty}(\text{dch}(S_2)), \psi_2 \equiv \text{inv}(\text{var}(S_1)) \land \square \text{empty}(\text{dch}(S_1)) \), 
\( \text{dch}(\varphi_1) \subseteq \text{dch}(S_1), \text{dch}(\varphi_2) \subseteq \text{dch}(S_2), \text{var}(\varphi_1) \subseteq \text{var}(S_1), \) and \( \text{var}(\varphi_2) \subseteq \text{var}(S_2) \). We show the validity of \( S_1 || S_2 \text{ sat } (\varphi_1 \land (\varphi_2 \text{ C } \psi_1)) \lor (\varphi_2 \land (\varphi_2 \text{ C } \psi_2)) \).

Consider any \( \sigma \in \mathcal{M}(S_1 || S_2) \) and any environment \( \gamma \). Then \( \text{dch}(\sigma) \subseteq \text{dch}(S_1) \cup \text{dch}(S_2) \), and for \( i \in \{1,2\} \), there exist \( \sigma_i \in \mathcal{M}(S_i) \) such that \( \text{begin}(\sigma) = \text{begin}(\sigma_1) = \text{begin}(\sigma_2) \), \( \text{end}(\sigma) = \max(\text{end}(\sigma_1), \text{end}(\sigma_2)) \). Suppose \( \text{end}(\sigma_1) \geq \text{end}(\sigma_2) \). Then \( \text{end}(\sigma) = \text{end}(\sigma_1) \). We prove \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \land (\varphi_2 \text{ C } \psi_1) \).

- First we prove \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \). From the semantics, we have that, for all \( \tau_1, \text{begin}(\sigma_1) \leq \tau_1 < \text{end}(\sigma_1), [\sigma \uparrow \text{var}(S_1)]_{\text{dch}(S_1)}(\tau_1).\text{comm} = \sigma_1(\tau_1).\text{comm} \), for all \( \tau'_1, \text{begin}(\sigma_1) \leq \tau'_1 \leq \text{end}(\sigma_1), [\sigma \uparrow \text{var}(S_1)]_{\text{dch}(S_1)}(\tau'_1).\text{state} = \sigma_1(\tau'_1).\text{state} \). Since \( \text{begin}(\sigma_1) \leq \tau'_1 \leq \text{end}(\sigma_1) \), \( [\sigma \uparrow \text{var}(S_1)]_{\text{dch}(S_1)}(\tau'_1) = \text{begin}(\sigma_1) \), \( \text{end}(\sigma_1) = \text{end}(\sigma_1) \), we can obtain \( [\sigma \uparrow \text{var}(S_1)]_{\text{dch}(S_1)} = \sigma_1 \). Since \( \sigma_1 \in \mathcal{M}(S_1) \) and \( S_1 \text{ sat } \varphi_1 \), we have \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \). Since \( \text{dch}(\varphi_1) \subseteq \text{dch}(S_1) \) and \( \text{var}(\varphi_1) \subseteq \text{var}(S_1) \), Lemma 6.9 and Lemma 6.10 lead to \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \).

- Next we prove \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_2 \text{ C } \psi_1 \).

  - If \( \text{end}(\sigma_2) = \infty \), then \( \text{end}(\sigma_1) = \infty \) and \( \text{end}(\sigma_2) = \text{end}(\sigma) = \infty \). Similarly as the proof above, we can also derive that \( [\sigma \uparrow \text{var}(S_2)]_{\text{dch}(S_2)} = \sigma_2 \). From \( \sigma_2 \in \mathcal{M}(S_2) \) and \( S_2 \text{ sat } \varphi_2 \), we obtain \( \langle \sigma \uparrow \text{var}(S_2) \rangle_{\text{dch}(S_2)} = \varphi_2 \). Since \( \text{dch}(\varphi_2) \subseteq \text{dch}(S_2) \) and \( \varphi(\varphi_2) \subseteq \text{var}(S_2) \), Lemma 6.9 and Lemma 6.10 lead to \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_2 \). From \( \psi_1 = \text{inv}(\text{var}(S_2)) \land \square \text{empty}(\text{dch}(S_2)) \), we can easily find a model which satisfies \( \psi_1 \). Let \( \sigma' \) be such a model that \( \langle \sigma', \text{begin}(\sigma') \rangle \gamma \models \varphi_1 \). By the definition of \( C \) operator, \( \langle \sigma', \text{begin}(\sigma) \rangle \gamma \models \varphi_2 \text{ C } \psi_1 \).

  - If \( \text{end}(\sigma_2) < \infty \), from \( S_2 \text{ sat } \varphi_2 \) and \( \sigma_2 \in \mathcal{M}(S_2) \), we obtain \( \langle \sigma_2, \text{begin}(\sigma_2) \rangle \gamma \models \varphi_2 \).

We define a model \( \sigma_3 \) such that \( \text{begin}(\sigma_3) = \text{end}(\sigma_2), \text{end}(\sigma_3) = \text{end}(\sigma) \), for all \( \tau_1, \text{begin}(\sigma_3) \leq \tau_1 < \text{end}(\sigma_3), \sigma_3(\tau_1).\text{comm} = \sigma(\tau_1).\text{comm} \), for all \( \tau'_1, \text{begin}(\sigma_3) \leq \tau'_1 \leq \text{end}(\sigma_3), \) for all \( x \in \text{VAR} \), if \( x \in \text{var}(S_2) \), \( \sigma_3(\tau'_1).\text{state}(x) = \sigma(\tau'_1).\text{state}(x) \), if \( x \notin \text{var}(S_2) \), \( \sigma_3(\tau'_1).\text{state}(x) = \sigma(\text{end}(\sigma_2)).\text{state}(x) \). From the semantics, for all \( \tau'_1, \text{end}(\sigma_3) \leq \tau'_1 \leq \text{end}(\sigma), \) for all \( x \in \text{var}(S_2) \), \( \sigma(\tau'_1).\text{state}(x) = \sigma(\text{end}(\sigma_2)).\text{state}(x) \). Thus for all \( \tau'_1, \text{begin}(\sigma_3) \leq \tau'_1 \leq \text{end}(\sigma_3), \sigma_3(\tau'_1).\text{state}(x) = \sigma_3(\text{end}(\sigma_2)).\text{state}(x) \). Therefore, for all \( \tau_2 > \text{end}(\sigma_3) \), \( \forall x [\gamma(\sigma_3, \tau_2) = \sigma_3(\text{end}(\sigma_2)).\text{state}(x) ] \).

Hence \( \langle \sigma_3, \text{begin}(\sigma_3) \rangle \gamma \models \text{inv}(\text{var}(S_2)) \). Again from the semantics, for all \( \tau_1, \text{end}(\sigma_2) \leq \tau_1 < \text{end}(\sigma), [\sigma]_{\text{dch}(S_2)}(\tau_1).\text{comm} = \emptyset \). Thus for all \( \tau_1, \text{begin}(\sigma_3) \leq \tau_1 < \text{end}(\sigma_3), [\sigma]_{\text{dch}(S_2)}(\tau_1).\text{comm} = \emptyset \). Then \( \langle \sigma_3, \tau_1 \rangle \gamma \models \text{empty}(\text{dch}(S_2)) \). By definition, for all \( \tau_2 \geq \text{end}(\sigma_3) \), \( \langle \sigma_3, \tau_2 \rangle \gamma \models \text{empty}(\text{dch}(S_2)) \). Thus we have \( \langle \sigma_3, \text{begin}(\sigma_3) \rangle \gamma \models \text{empty}(\text{dch}(S_2)) \), i.e. \( \langle \sigma_3, \text{begin}(\sigma_3) \rangle \gamma \models \psi_1 \). By the definition of \( C \) operator, we obtain
\langle \sigma_2 \sigma_3, \text{begin}(\sigma_2) \rangle \models \varphi_2 \land (\varphi_2 \land \psi_1). \] By definitions, we can also derive \([\sigma \uparrow \text{var}(S_2)]_{\text{dch}(S_2)} = \sigma_2 \sigma_3\). Thus \([\sigma \uparrow \text{var}(S_2)]_{\text{dch}(S_2)} \models \varphi_2 \land \psi_1\). Since \(\text{dch}(\varphi_2) \subseteq \text{dch}(S_2)\) and \(\text{var}(\varphi_2) \subseteq \text{var}(S_2)\), we have \(\text{dch}(\varphi_2 \land \psi_1) \subseteq \text{dch}(S_2)\) and \(\text{var}(\varphi_2 \land \psi_1) \subseteq \text{var}(S_2)\). Then Lemma 6.9 and Lemma 6.10 lead to \(\langle \sigma, \text{begin}(\sigma) \rangle \models \varphi_2 \land \psi_1\).

Therefore we have proved \(\langle \sigma, \text{begin}(\sigma) \rangle \models \varphi_1 \land (\varphi_2 \land \psi_1)\).

Similarly, for \(\text{end}(\sigma_1) \leq \text{end}(\sigma_2)\), we can show \(\langle \sigma, \text{begin}(\sigma) \rangle \models \varphi_2 \land (\varphi_1 \land \psi_2)\). Hence the General Parallel Composition Rule is sound.
Appendix C

Preciseness Proof

To prove Theorem 6.16 we show that for every program \( S \) we can prove \( S \text{ sat } \varphi \) where \( \varphi \) is precise for \( S \). To show that an assertion \( \varphi \) is precise for a program \( S \), we have to prove

1. \( S \text{ sat } \varphi \), i.e. \( \langle \sigma, \text{begin}(\sigma) \rangle \models \varphi \) for all \( \sigma \in \mathcal{M}(S) \) and any environment \( \gamma \);
2. If \( \sigma \) is a well-formed model, \( \text{dch}(\sigma) \subseteq \text{dch}(S) \), for any program variable \( x \notin \text{wvar}(S) \), \( x \) is invariant with respect to \( \sigma \), and \( \langle \sigma, \text{begin}(\sigma) \rangle \models \varphi \) for any environment \( \gamma \), then \( \sigma \in \mathcal{M}(S) \); and
3. \( \text{dch}(\varphi) = \text{dch}(S) \) and \( \text{var}(\varphi) = \text{var}(S) \).

By induction on the structure of \( S \), we show that, for every program \( S \), \( S \text{ sat } \varphi \) with \( \varphi \) precise for \( S \).

Skip

By the Skip Axiom, \( \text{skip sat term} = \text{start} \). We show that the assertion \( \text{term} = \text{start} \) is precise for statement \( \text{skip} \). From the Soundness Theorem 6.3, \( \text{skip sat term} = \text{start} \) is valid. Let \( \gamma \) be any environment. Consider a well-formed model \( \sigma \) such that \( \langle \sigma, \text{begin}(\sigma) \rangle \models \text{term} = \text{start} \). Then \( \text{end}(\sigma) = \text{begin}(\sigma) \), and hence \( \sigma \in \mathcal{M}(\text{skip}) \). We also have \( \text{dch(term} = \text{start}) = \emptyset = \text{dch(skip)} \) and \( \text{var(term} = \text{start}) = \emptyset = \text{var(skip)} \). Hence \( \text{term} = \text{start} \) is precise for \( \text{skip} \).

Assignment

Let \( \varphi \equiv x = v \rightarrow (x = v \text{ U (T = term} = \text{start} + K_\alpha \land x = e[v/x])) \). By the Assignment Axiom, \( x := e \text{ sat } \varphi \). We show that \( \varphi \) is a precise specification for \( x := e \). By the Soundness Theorem 6.3, \( x := e \text{ sat } \varphi \) is valid. Consider a well-formed model \( \sigma \) such that \( \text{dch}(\sigma) \subseteq \text{dch}(x := e) \) and every program variable \( y \notin \text{wvar}(x := e) \) is invariant with respect to \( \sigma \). Thus \( \text{dch}(\sigma) = \emptyset \), for all \( \tau_1, \text{begin}(\sigma) \leq \tau_1 < \text{end}(\sigma) \), \( \sigma(\tau_1).\text{comm} = \emptyset \), and for every program variable \( y \neq x \), for all \( \tau_2, \text{begin}(\sigma) \leq \tau_2 \leq \text{end}(\sigma) \), \( \sigma(\tau_2).\text{state}(y) = \sigma(\text{begin}(\sigma)).\text{state}(y) \). Let \( \gamma \) be such an environment that \( \gamma(v) = \sigma(\text{begin}(\sigma)).\text{state}(x) \). Thus we have \( \langle \sigma, \text{begin}(\sigma) \rangle \models x = v \). Assume \( \langle \sigma, \text{begin}(\sigma) \rangle \models \varphi \). Then we obtain \( \langle \sigma, \text{begin}(\sigma) \rangle \models x = v \text{ U (T = term} = \text{start} + K_\alpha \land x = e[v/x]) \). Then \( \text{end}(\sigma) = \text{begin}(\sigma) + K_\alpha \) and for all \( \tau_1, \text{begin}(\sigma) \leq \tau_1 < \text{end}(\sigma) \), \( \sigma(\tau_1).\text{state}(x) = \sigma(\text{begin}(\sigma)).\text{state}(x) \). Furthermore, \( \sigma(\text{end}(\sigma)).\text{state}(x) = \forall (e[v/x])\gamma(\sigma, \text{end}(\sigma)) = \forall (e[v/x])\gamma(\sigma, \text{begin}(\sigma)) = \forall (e)\gamma(\sigma, \text{begin}(\sigma)) = E(e)\gamma(\sigma, \text{begin}(\sigma)) = \text{state} \). Thus, for all \( \tau_1, \text{begin}(\sigma) \leq \tau_1 < \text{end}(\sigma) \), \( \sigma(\tau_1).\text{state} = \sigma(\text{begin}(\sigma)).\text{state} \). \( \sigma(\text{end}(\sigma)).\text{state} = (\sigma(\text{begin}(\sigma)).\text{state} : x \mapsto E(e)\sigma(\text{begin}(\sigma)).\text{state}) \). Hence \( \sigma \in \mathcal{M}(x := e) \). It is obvious that \( \text{dch}(\varphi) = \emptyset = \text{dch}(x := e) \) and \( \text{var}(\varphi) = \{x\} \cup \text{var}(e) = \text{var}(x := e) \). Hence \( \varphi \) is a precise specification for \( x := e \).
Delay

Let \( \varphi \equiv \text{term} = \text{start} + e \). By the Delay Axiom, \( \text{delay} \ e \sat \varphi \). We show that \( \varphi \) is a precise specification for \( \text{delay} \ e \). By the Soundness Theorem 6.3, \( \text{delay} \ e \sat \varphi \) is valid. Consider a well-formed model \( \sigma \) such that \( \text{dch}(\sigma) \subseteq \text{dch}(\text{delay} \ e) \) and every program variable \( y \notin \text{wvar}(\text{delay} \ e) \) is invariant with respect to \( \sigma \). Thus \( \text{dch}(\sigma) = \emptyset \), for all \( \tau_1 \), begin(\( \sigma \)) \leq \( \tau_1 \) end(\( \sigma \)), \( \sigma(\tau_1) \).comm = \( \sigma \), and for every program variable \( y \), for all \( \tau_2 \), begin(\( \sigma \)) \leq \( \tau_2 \) end(\( \sigma \)), \( \sigma(\tau_2) \).state(\( y \)) = \( \sigma(\text{begin}(\sigma)) \).state(\( y \)). Then \( \sigma(\tau_2) \).state = \( \sigma(\text{begin}(\sigma)) \).state. Let \( \gamma \) be an arbitrary environment. Assume \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \vDash \varphi \). Thus \( \text{end}(\sigma) = \text{begin}(\sigma) + \forall(\varepsilon) \gamma(\sigma, \text{begin}(\sigma)) = \text{begin}(\sigma) + \varepsilon(\varepsilon) \sigma(\text{begin}(\sigma)) \).state. Hence \( \sigma \in M(\text{delay} \ e) \). Finally, it is obvious that \( \text{dch}(\varphi) = \emptyset = \text{dch}(\text{delay} \ e) \) and \( \text{var}(\varphi) = \text{var}(e) = \text{var}(\text{delay} \ e) \). Therefore \( \varphi \) is a precise specification for \( \text{delay} \ e \).

Output

Let \( \varphi \equiv \text{wait}(c!) \cup (T = \text{term} - K_e \land (\text{comm}(c, e) \cup T = \text{term})) \). By the Output Axiom, \( c!\text{ sat } \varphi \). We show that \( \varphi \) is precise for \( c! \). By the Soundness Theorem 6.3, \( c!\text{ sat } \varphi \) is valid. Consider a well-formed model \( \sigma \) such that \( \text{dch}(\sigma) \subseteq \text{dch}(c!) \) and every program variable \( y \notin \text{wvar}(c!) \) is invariant with respect to \( \sigma \). Then \( \text{dch}(\sigma) \subseteq \{c, e\} \) and for every program variable \( y \), for all \( \tau \), begin(\( \sigma \)) \leq \( \tau \) end(\( \sigma \)), \( \sigma(\tau) \).state(\( y \)) = \( \sigma(\text{begin}(\sigma)) \).state(\( y \)). Hence \( \sigma(\tau) \).state = \( \sigma(\text{begin}(\sigma)) \).state. Let \( \gamma \) be any environment. Assume \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \vDash \varphi \). Then there are two possibilities:

- Either \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \vDash \Box \text{wait}(c!) \);
- Or \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \vDash \text{wait}(c!) \cup (T = \text{term} - K_e \land (\text{comm}(c, e) \cup T = \text{term})) \).

That is,

- Either for all \( \tau \geq \text{begin}(\sigma) \), \( \langle \sigma, \tau \rangle \gamma \vDash \text{wait}(c!) \). Then \( \text{begin}(\sigma) \leq \tau \leq \text{end}(\sigma) \) and hence \( \text{end}(\sigma) = \infty \). By definition, for all \( \tau \geq \text{begin}(\sigma) \), \( c! \in \sigma(\tau) \).comm. Since \( \sigma \) is a well-formed model, for any value \( \vartheta \in \text{VAL} \) and any \( \tau \), begin(\( \sigma \)) \leq \( \tau \leq \text{end}(\sigma) \), \( \neg(\vartheta) \in \sigma(\tau) \).comm \land (\( c, \vartheta \) \in \sigma(\tau) \).comm) is valid. Then \( \sigma(\tau) \).comm = \( \{c!\} \). Together with \( \sigma(\tau) \).state = \( \sigma(\text{begin}(\sigma)) \).state, we obtain \( \sigma \in M(c!) \);

- Or there exists a \( \tau \geq \text{begin}(\sigma) \), \( \tau \in \text{TIME} \), such that, for all \( \tau_1 \), begin(\( \sigma \)) \leq \( \tau_1 \), \( \tau \), \( \langle \sigma, \tau_1 \rangle \gamma \vDash \text{wait}(c!) \) and \( \langle \sigma, \tau \rangle \gamma \vDash T \equiv \text{term} - K_e \land (\text{comm}(c, e) \cup T = \text{term}) \). We split \( \sigma \) as two models \( \sigma_1 \) and \( \sigma_2 \) such that \( \sigma = \sigma_1 \sigma_2 \) with \( \text{end}(\sigma_1) = \tau \). Thus \( \text{begin}(\sigma_2) = \text{end}(\sigma_1) = \tau \). Then we obtain that, for all \( \tau_1 \), begin(\( \sigma_1 \)) \leq \( \tau_1 \), \( \text{end}(\sigma_1) \), \( \sigma_1(\tau_1) \).comm = \( \{c!\} \). Together with \( \sigma(\tau) \).state = \( \sigma(\text{begin}(\sigma)) \).state, for all \( \tau \), begin(\( \sigma \)) \leq \( \tau \leq \text{end}(\sigma) \), we obtain \( \sigma_1 \in \text{Wait}(c!) \). From \( \langle \sigma, \tau \rangle \gamma \vDash T = \text{term} - K_e \), we obtain \( \tau = \text{end}(\sigma) - K_e \) and then \( \text{end}(\sigma_2) = \tau + K_e = \text{begin}(\sigma_2) + K_e \). From \( \langle \sigma, \tau \rangle \gamma \vDash \text{comm}(c, e) \cup T = \text{term} \), we can derive that, for all \( \tau_2 \), begin(\( \sigma_2 \)) \leq \( \tau_2 \), \( \in \text{end}(\sigma_2) \), \( (c, V(e) \gamma(\sigma_2, \tau_2)) \in \sigma_2(\tau_2) \).comm. By the well-formedness of \( \sigma \) and the invariance of program variables, \( \sigma_2(\tau_2) \).comm = \( \{c, V(e) \gamma(\sigma_2, \text{begin}(\sigma_2))\} \). Together with \( \sigma(\tau) \).state = \( \sigma(\text{begin}(\sigma)) \).state, for all \( \tau \), begin(\( \sigma \)) \leq \( \tau \leq \text{end}(\sigma) \), we obtain \( \sigma_2 \in \text{Send}(c, e) \) and hence \( \sigma \in M(c!) \).
It is easy to see that \( \text{deh}(\varphi) = \{e, e!\} = \text{deh}(e!e) \) and \( \text{var}(\varphi) = \text{var}(e) = \text{var}(e!e) \). Therefore \( \varphi \) is precise for \( e!e \).

**Input**

Let \( \varphi \equiv x = v \rightarrow ((x = v \land \text{wait}(c?)) \cup (T = \text{term} - K_c \land ((x = v \land \text{comm}(c, \text{last}(x))) \cup T = \text{term})) \). By the Input Axiom, \( e?x \) sat \( \varphi \). We show that \( \varphi \) is precise for \( e?x \). From the Soundness Theorem 6.3, \( e?x \) sat \( \varphi \). Consider a well-formed model \( \sigma \) such that \( \text{deh}(\sigma) \subseteq \text{deh}(e?x) \) and every program variable \( y \notin \text{wvar}(e?x) \) is invariant with respect to \( \sigma \). Then \( \text{deh}(\sigma) \subseteq \{e, e?\} \) and, for all \( \tau \), \( \text{begin}(\sigma) \leq \tau \leq \text{end}(\sigma) \), for any program variable \( y \neq x \), \( \sigma(\tau).\text{state}(y) = \sigma(\text{begin}(\sigma)).\text{state}(y) \). Let \( \gamma \) be such that \( \gamma(v) = \sigma(\text{begin}(\sigma)).\text{state}(x) \). Then we have \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models x = v \). Assume \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi \). Thus we obtain \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models (x = v \land \text{wait}(c?)) \cup (T = \text{term} - K_c \land ((x = v \land \text{comm}(e, \text{last}(x))) \cup T = \text{term})) \). There are two possibilities:

- Either \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \Box (x = v \land \text{wait}(c?)) ; \)
- Or \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models (x = v \land \text{wait}(c?)) \cup (x = v \land \text{wait}(c?)) \cup T = \text{term} \) \].

That is,

- Either \( \text{end}(\sigma) = \infty \), for all \( \tau \geq \text{begin}(\sigma) \), \( \sigma(\tau).\text{state}(x) = \sigma(\text{begin}(\sigma)).\text{state}(x) \), and \( e? \in \sigma(\tau).\text{comm} \). From the invariance of program variables different from \( x \) and the well-formedness of \( \sigma \), we obtain that, for all \( \tau \geq \text{begin}(\sigma) \), \( \sigma(\tau).\text{state} = \sigma(\text{begin}(\sigma)).\text{state} \) and \( \sigma(\tau).\text{comm} = \{e, e?\} \). Hence \( \sigma \in \mathcal{M}(e?x) \);
- Or there exists a \( \tau \geq \text{begin}(\sigma) \), \( \tau \in \text{TIME} \), such that, for all \( \tau_1 \), \( \text{begin}(\sigma) \leq \tau_1 < \tau \), \( \langle \sigma, \tau_1 \rangle \gamma \models x = v \land \text{wait}(c!) \) and \( \langle \sigma, \tau \rangle \gamma \models T = \text{term} - K_c \land ((x = v \land \text{comm}(e, \text{last}(x))) \cup T = \text{term}) \). We split \( \sigma \) as two models \( \sigma_1 \) and \( \sigma_2 \) such that \( \sigma = \sigma_1 \sigma_2 \) with \( \text{end}(\sigma_1) = \tau \). Then \( \text{begin}(\sigma_2) = \text{end}(\sigma_1) = \tau \). We can obtain that, for all \( \tau_1 \), \( \text{begin}(\sigma_1) \leq \tau_1 < \text{end}(\sigma_1) \), \( \sigma_1(\tau_1).\text{state} = \sigma_1(\text{begin}(\sigma_1)).\text{state} \), \( \sigma_1(\tau_1).\text{comm} = \{e, e?\} \). From \( \langle \sigma, \tau \rangle \gamma \models T = \text{term} - K_c \), we have \( \tau = \text{end}(\sigma) - K_c \) and thus \( \text{end}(\sigma_2) = \text{begin}(\sigma_2) + K_c \). We can also derive that, for all \( \tau_2 \), \( \text{begin}(\sigma_2) \leq \tau_2 < \text{end}(\sigma_2) \), \( \sigma_2(\tau_2) \gamma \models x = v \land \text{comm}(e, \text{last}(x)) \). Together with the invariance of program variables different from \( x \), we then have \( \sigma_2(\tau_2).\text{state} = \sigma_2(\text{begin}(\sigma_2)).\text{state} \). Since \( \sigma = \sigma_1 \sigma_2 \) and \( \sigma_1(\text{end}(\sigma_1)).\text{state}(x) = \sigma_2(\text{begin}(\sigma_2)).\text{state}(x) = \gamma(v) \), we obtain that \( \sigma_1(\text{end}(\sigma_1)).\text{state} = \sigma_1(\text{begin}(\sigma_1)).\text{state} \). Thus \( \sigma_1 \in \mathcal{W}(\text{last}(x)) \). By definition, \( \mathcal{W}(\text{last}(x)) \gamma(\sigma_2(\tau_2)) = \sigma_2(\text{end}(\sigma_2)).\text{state}(x) \). Let \( \theta = \sigma_2(\text{end}(\sigma_2)).\text{state}(x) \). Hence by the well-formedness of \( \sigma \), we obtain that, for all \( \tau_2 \), \( \text{begin}(\sigma_2) \leq \tau_2 < \text{end}(\sigma_2) \), \( \sigma_2(\tau_2).\text{comm} = \{(e, \theta)\} \). Furthermore, we also have \( \sigma_2(\text{end}(\sigma_2)).\text{state} = (\sigma_2(\text{begin}(\sigma_2)).\text{state} : x \mapsto \theta) \). Hence \( \sigma_2 \in \mathcal{R}(e, x) \) and then \( \sigma \in \mathcal{M}(e?x) \).

It is obvious that \( \text{deh}(\varphi) = \{e, e?\} = \text{deh}(e?x) \) and \( \text{var}(\varphi) = \{x\} = \text{var}(e?x) \). Hence \( \varphi \) is precise for \( e?x \).
\[\forall y \in \text{var}(G) \, y = v_y \rightarrow (\text{Quiet } U (T = \text{start} + K_g \land \forall y \in \text{var}(G) \, y = v_y)) \land (\neg b_G \rightarrow \text{term} = \text{start} + K_g)\] 
\[\neg [b_G \rightarrow (\text{term} = \text{start} + K_g)] \land [b_G \rightarrow (\text{term} = \text{start} + K_g)] \] 
\[\forall [b_G \rightarrow (\text{term} = \text{start} + K_g)] \land \forall [b_G \rightarrow (\text{term} = \text{start} + K_g)] \] 
\[\text{inv}(\text{wvar}(G) - \text{wvar}(S_i))\].

We prove that \(\varphi\) is precise for \(G\). By the Soundness Theorem 6.3, if \(G\) sat \(\varphi\) is valid. Consider a well-formed model \(\sigma\) such that \(\text{dch}(\sigma) \subseteq \text{dch}(G)\) and any program variable \(y \notin \text{wvar}(G)\) is invariant with respect to \(\sigma\). Let \(y\) be such an environment that, for all logical variables \(v_y\) where \(y \in \text{var}(G)\), \(\gamma(v_y) = \sigma(\text{begin}(\sigma)).\text{state}(y)\). Then we have \(\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \forall y \in \text{var}(G) \, y = v_y\). Assume \(\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi\). Then \(\langle \sigma, \text{begin}(\sigma) \rangle \gamma \models (\text{Quiet } U (T = \text{start} + K_g \land \forall y \in \text{var}(G) \, y = v_y)) \land (\neg b_G \rightarrow \text{term} = \text{start} + K_g)\). Thus, there exists a \(\tau_1 \geq \text{begin}(\sigma)\) such that for all \(\tau_1'\), \(\text{begin}(\sigma) \leq \tau_1' < \tau_1\), \(\langle \sigma, \tau_1' \rangle \gamma \models \text{Quiet}\) and \(\langle \sigma, \tau_1 \rangle \gamma \models T = \text{start} + K_g \land \forall y \in \text{var}(G) \, y = v_y\). That is, \(\langle \sigma, \tau_1' \rangle \gamma \models \forall y \in \text{var}(G) \, y = v_y\) and \(\tau_1 = \text{begin}(\sigma) + K_g\). Hence, for any \(y \in \text{var}(G)\), for all \(\tau_2\), \(\text{begin}(\sigma) \leq \tau_2 \leq \tau_1\), \(\langle \sigma, \tau_2 \rangle \gamma \models \text{begin}(\sigma) + K_g\). From \(\text{begin}(\sigma) = \text{begin}(\sigma_1) + K_g\), we obtain \(\sigma_1 \in \mathcal{M}(\text{delay } K_g)\). Since \(\text{end}(\sigma_1) < \infty\), by the definition of \(\sigma_1\), we have \(\text{end}(\sigma_1) = \text{begin}(\sigma_2)\) and \(\sigma_1(\text{end}(\sigma_1))\text{.state} = \sigma_2(\text{begin}(\sigma_2))\text{.state}\). Furthermore, there must exist a \(k, 1 \leq k \leq n\), such that \(\langle \sigma_2, \text{begin}(\sigma_2) \rangle \gamma \models b_k \land \varphi_k \land \exists \text{empty}(\text{dch}(G) - \text{dch}(S_k))\land \text{inv}(\text{wvar}(G) - \text{wvar}(S_k))\). From \(\langle \sigma_2, \text{begin}(\sigma_2) \rangle \gamma \models b_k\), by Lemma 6.2, \(B(b_k)(\sigma_2(\text{begin}(\sigma_2))\text{.state})\). From \(\langle \sigma_2, \text{begin}(\sigma_2) \rangle \gamma \models \text{inv}(\text{wvar}(G) - \text{wvar}(S_k))\), every program variable \(x \in \text{wvar}(G) - \text{wvar}(S_k)\) is invariant with respect to \(\sigma_2\). By the assumption, any program variable \(y \notin \text{wvar}(G)\) is invariant with respect to \(\sigma\). Thus, any program variable \(x \notin \text{wvar}(S_k)\) is invariant with respect to \(\sigma_2\). From \(\langle \sigma_2, \text{begin}(\sigma_2) \rangle \gamma \models \exists \text{empty}(\text{dch}(G) - \text{dch}(S_k))\), Lemma 6.11 leads to \([\sigma_2]\text{dch}(G) = [\sigma_2]\text{dch}(S_k)\). Since \(\text{dch}(G) \cup \text{dch}(S_k) = \text{dch}(G)\), we obtain \([\sigma_2]\text{dch}(G) = [\sigma_2]\text{dch}(S_k)\). From \(\sigma = \sigma_1\sigma_2\) and \(\text{dch}(\sigma) \subseteq \text{dch}(G)\), we can derive \(\text{dch}(\sigma_2) \subseteq \text{dch}(G)\). By Lemma 6.4, it implies \(\text{dch}(\sigma_2) \subseteq \text{dch}(G)\). By the Lemma 6.4 again, we obtain \(\text{dch}(\sigma_2) \subseteq \text{dch}(S_k)\). Since \(\sigma\) is a well-formed model, \(\sigma_1\) and \(\sigma_2\) are also well-formed. Together with \(\langle \sigma_2, \text{begin}(\sigma_2) \rangle \gamma \models \varphi_k\) and the preciseness of \(\varphi_k\) for \(S_k\), \(\sigma_2 \in \mathcal{M}(S_k)\). By \(\sigma = \sigma_1\sigma_2\) and \(\sigma_1 \in \mathcal{M}(\text{delay } K_g)\), we obtain \(B(b_k)(\sigma(\text{begin}(\sigma))\text{.state})\). By the definition of \(\text{SEQ}\), we have \(\sigma \in \mathcal{M}(\text{delay } K_g, S_k)\).
Both cases lead to \( \sigma \in \mathcal{M}([\llbracket \prod_{i=1}^{n} b_i \rightarrow S_i \rrbracket]) \). It is obvious that \( \text{dch}(\varphi) = \text{dch}(G) \) and \( \text{var}(\varphi) = \text{var}(G) \). Hence \( \varphi \) is precise for \([\llbracket \prod_{i=1}^{n} b_i \rightarrow S_i \rrbracket]\).

Guarded Command with IO-guards

Consider \( G \equiv [\llbracket \prod_{i=1}^{n} b_i ; c_i ? x_i \rightarrow S_i \rrbracket ] \& b \& \text{delay} \ e \rightarrow S] \). By the induction hypothesis, we can derive \( c_i ? x_i ; S_i \text{ sat } \varphi_i \) where \( \varphi_i \) is precise for \( c_i ? x_i ; S_i \). By the Communication Invariance Axiom, the Variable Invariance Axiom, and the Conjunction Rule, we obtain \( c_i ? x_i ; S_i \text{ sat } \varphi_i \land \square \text{ empty}(\text{dch}(G) - \text{dch}(c_i ? x_i ; S_i)) \land \text{ inv}(\text{var}(G) - \text{var}(c_i ? x_i ; S_i)) \). Similarly, we can derive \( S \text{ sat } \varphi \land \square \text{ empty}(\text{dch}(G) - \text{dch}(S)) \land \text{ inv}(\text{var}(G) - \text{var}(S)) \) where \( \varphi \) is precise for \( S \). By the Guarded Command Evaluation Axiom, the Guarded Command with IO-guards Rule, the Conjunction Rule, and the Consequence Rule, we obtain \( G \text{ sat } \psi \) with

\[
\psi \equiv \bigwedge_{y \in \text{var}(G)} y = v_y \rightarrow [(\text{Quiet } u (T = \text{start} + K_g \land \bigwedge_{y \in \text{var}(G)} y = v_y)) \land
(\neg b_G \rightarrow \text{ term} = \text{ start} + K_g)] \land [b_G \rightarrow (\text{Eval } C (N\text{Comm } \lor N\text{TimeOut}))]
\]

where

\[
N\text{Comm} \equiv (\text{Wait } \lor \text{ InTime}) C \psi_1, \quad N\text{TimeOut} \equiv (\text{Wait } \lor \text{ EndTime}) C \psi_2
\]

with

\[
\psi_1 \equiv \bigvee_{i=1}^{n} [b_i \land \varphi_i \land \text{comm}(c_i) \land \square \text{ empty}(\text{dch}(G) - \text{dch}(c_i ? x_i ; S_i))] \land \\
\text{ inv}(\text{var}(G) - \text{var}(c_i ? x_i ; S_i))]
\]

\[
\psi_2 \equiv \varphi \land \square \text{ empty}(\text{dch}(G) - \text{dch}(S)) \land \text{ inv}(\text{var}(G) - \text{var}(S)).
\]

We prove that \( \psi \) is precise for \( G \). From the Soundness Theorem 6.3, \( G \text{ sat } \psi \) is valid. Consider a well-formed model \( \sigma \) such that \( \text{dch}(\sigma) \subseteq \text{dch}(G) \) and any program variable \( y \notin \text{var}(G) \) is invariant with respect to \( \sigma \). Let \( \gamma \) be such an environment that, for all logical variables \( v_y \) where \( y \in \text{var}(G) \), \( \gamma(v_y) = \sigma(\text{begin}(\sigma)).\text{state}(y) \). Then we have \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma = \bigwedge_{y \in \text{var}(G)} y = v_y \). Assume \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \psi \). We prove that \( \sigma \in \mathcal{M}(G) \). Similarly as the preciseness proof for \( G \equiv [\llbracket \prod_{i=1}^{n} b_i \rightarrow S_i \rrbracket] \), we can derive that, for all \( \tau_1 \), \( \text{begin}(\sigma) \leq \tau_1 \leq \text{begin}(\sigma) + K_g \), \( \sigma(\tau_1).\text{state} = \sigma(\text{begin}(\sigma)).\text{state} \) and, for all \( \tau_1' \), \( \text{begin}(\sigma) \leq \tau_1' < \text{begin}(\sigma) + K_g \), \( \sigma(\tau_1').\text{comm} = \emptyset \). Next consider the validity of \( b_G \). There are two possibilities:

- If \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \neg b_G \), Lemma 6.2 leads to \( B(\neg b_G)(\sigma(\text{begin}(\sigma)).\text{state}). \) By assumption, \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \text{ term} = \text{ start} + K_g \) and hence \( \text{end}(\sigma) = \text{ begin}(\sigma) + K_g \). Then \( \sigma \in \mathcal{M}(\text{delay } K_g) \). Thus \( \sigma \in \mathcal{M}(G) \);
- If \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models b_G \), then \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models (\text{term} = \text{start} + K_g) \ C \)
(\((\text{Wait } \lor \text{ InTime}) C \psi_1) \lor (\text{Wait } \lor \text{ EndTime}) C \psi_2))

For this case, consider the further three possibilities:

1. If \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models (\text{term} = \text{start} + K_g) C ((\text{Wait } \lor \text{ InTime}) C \psi_1) \), there exist models \( \sigma_1 \) and \( \sigma_2 \) such that \( \sigma = \sigma_1 \sigma_2 \), \( \langle \sigma_1, \text{begin}(\sigma_1) \rangle \gamma \models \text{ term} = \text{start} + K_g \), and \( \langle \sigma_2, \text{begin}(\sigma_2) \rangle \gamma \models (\text{Wait } \lor \text{ InTime}) C \psi_1 \). Then \( \text{end}(\sigma_1) = \text{begin}(\sigma_1) + K_g \). By \( \text{begin}(\sigma) = \text{begin}(\sigma_1) \), we obtain \( \sigma_1 \in \mathcal{M}(\text{delay } K_g) \). Furthermore, there exist models \( \sigma_{21} \) and \( \sigma_{22} \) such that \( \sigma_2 = \sigma_{21} \sigma_{22} \), \( \langle \sigma_{21}, \text{begin}(\sigma_{21}) \rangle \gamma \models \text{ Wait } \lor \text{ InTime} \), and \( \langle \sigma_{22}, \text{begin}(\sigma_{22}) \rangle \gamma \models \psi_1 \). We prove that \( \sigma_{21} \in \text{BoundWait}(G) \lor \text{ AnyWait}(G) \) and \( \sigma_{22} \in \text{Comm}(G) \). There
exists a \( \tau_2 \geq \text{begin}(\sigma_{21}) \) such that, for all \( \tau'_2 \), \( \text{begin}(\sigma_{21}) \leq \tau'_2 < \tau_2 \), \( \langle \sigma_{21}, \tau'_2 \rangle \vdash \Lambda_{y \in \text{var}(G)} y = v_y \land \text{empty}(\text{dch}(G) - \{c_1, \ldots, c_n\}) \land (b \leftarrow T < \text{start} + e) \land \Lambda_{i = 1}^n (b_i \leftarrow \text{wait}(c_i)) \) and \( \langle \sigma_{21}, \tau_2 \rangle \vdash \Lambda_{y \in \text{var}(G)} y = v_y \land (T = \text{term}) \land (b \leftarrow T < \text{start} + e) \). Then, \( \text{end}(\sigma_{21}) = \tau_2 \) and, for any \( y \in \text{var}(G) \), for all \( \tau'_2 \), \( \text{begin}(\sigma_{21}) \leq \tau'_2 \leq \tau_2 \), \( \sigma_{21}(\tau'_2).\text{state}(y) = \sigma_{21}(\text{begin}(\sigma_{21})).\text{state}(y) \). Together with the invariance of the variables \( y \notin \text{var}(G) \), we obtain \( \sigma_{21}(\tau'_2).\text{state} = \sigma_{21}(\text{begin}(\sigma_{21})).\text{state} \). Since \( \sigma \) is a well-formed model, so are \( \sigma_{21} \) and \( \sigma_{22} \). Since \( \langle \sigma_{21}, \tau'_2 \rangle \vdash \text{empty}(\text{dch}(G) - \{c_1, \ldots, c_n\}) \land \Lambda_{i = 1}^n (b_i \leftarrow \text{wait}(c_i)) \), we obtain that \( \sigma_{21}(\tau'_2).\text{comm} = \{c_i | E(b_i)(\text{begin}(\sigma_{21})).\text{state} \} \). By assumption, \( \langle \sigma, \text{begin}(\sigma) \rangle \vdash b_G \). By Lemma 6.2, \( B(b_G)(\sigma(\text{begin}(\sigma)).\text{state}) \) and hence \( B(b_G)(\sigma(\text{begin}(\sigma)).\text{state}) \). Next we prove \( \text{end}(\sigma_{21}) = \infty \). Suppose \( \text{end}(\sigma_{21}) < \infty \). By definition, for all \( \tau_3 \geq \text{end}(\sigma_{21}) \), we can derive \( \langle \sigma_{21}, \tau_3 \rangle \vdash \text{empty}(\text{dch}(G)) \). By assumption,
Since $B(\neg b)(\sigma(\text{begin}().\text{state}))$, there exists a $k$, $1 \leq k \leq n$, such that $(\sigma, \text{begin}().\gamma) \models b_k$. Then, for all $\tau_2 \geq \text{begin}().$, $(\sigma_2, \tau_2)\gamma \models \text{wait}(c_k)$ and hence $(\sigma_2, \tau_2)\gamma \models \text{empty}(\text{dch}(G))$. This contradiction leads to $\text{end}(). = \infty$. We can also derive that, for all $\tau_2 \geq \text{begin}().$, $\sigma_2(\tau_2).\text{state} = \sigma(\text{begin}().).\text{state}$. Moreover, $\sigma_2(\tau_2).\text{comm} = \{c? | B(b)(\sigma(\text{begin}().).\text{state}), 1 \leq i \leq n\}$. Hence $\sigma_2 \in \text{AnyWait}(G)$. We can easily find a model which belongs to $\text{Comm}(G)$. Let $\sigma_3$ be such a model that $\sigma_3 \in \text{Comm}(G)$.

By the definition of $\text{SEQ}$, we have $\sigma_2 \sigma_3 \in \text{SEQ}(\text{AnyWait}(G), \text{Comm}(G))$. Since $\text{end}(). = \infty$, we have $\sigma_2 \sigma_3 = \sigma_2$. Thus $\sigma_2 \in \text{SEQ}(\text{AnyWait}(G), \text{Comm}(G))$. Together with $\sigma = \sigma_1 \sigma_2$ and $\sigma_1 \in \mathcal{M}(\text{delay } K_g)$, we obtain $\sigma \in \text{SEQ}(\mathcal{M}(\text{delay } K_g), \text{AnyWait}(G), \text{Comm}(G))$ and hence $\sigma \in \mathcal{M}(G)$.

3. If $(\sigma, \text{begin}().)\gamma \models (\text{term} = \text{start} + K_g)C(\text{Wait U EndTime})C\psi_2$, there exist $\sigma_1$ and $\sigma_2$ such that $\sigma = \sigma_1 \sigma_2$, $(\sigma_1, \text{begin}().)\gamma \models \text{term} = \text{start} + K_g$, and $(\sigma_2, \text{begin}().)\gamma \models (\text{Wait U EndTime})C\psi_2$. Thus $\sigma_1 \in \mathcal{M}(\text{delay } K_g)$. Furthermore, there exist models $\sigma_2$ and $\sigma_2$ such that $\tau_2 = \text{begin}()$ and $(\sigma_2, \text{begin}().)\gamma \models (\text{Wait U EndTime})C\psi_2$. We prove that $\sigma_2 \in \text{TimeOut}(G)$ and $\sigma_2 \in \mathcal{M}(S)$. There exists a $\tau_2 \geq \text{begin}()$ such that, for all $\tau_2$, $\text{begin}() \leq \tau_2 < \tau_2$, $(\sigma_2, \tau_2')\gamma \models \text{Wait}$ and $(\sigma_2, \tau_2')\gamma \models \text{EndTime}$. That is, $(\sigma_2, \tau_2')\gamma \models \bigwedge_{y \in \text{var}(G)} y = v_y \land b \land T = \text{term} = \text{start} + e$. Hence $\text{end}(). = \tau_2 = \text{begin}(). + \epsilon$. $(\sigma_2, \tau_2')\gamma \models \text{EndTime}$, and, by Lemma 6.2, $B(b)(\sigma_2(\tau_2).\text{state})$. We can also derive that, for all $\tau_2$, $\text{begin}(). \leq \tau_2 < \tau_2$, $\sigma_2(\tau_2).\text{state} = \sigma(\text{begin}().).\text{state}$ and, for all $\tau_2$, $\text{begin}(). \leq \tau_2 < \tau_2$, $\sigma_2(\tau_2).\text{comm} = \{c? | B(b)(\sigma(\text{begin}().).\text{state}), 1 \leq i \leq n\}$. Thus $\text{end}(). = \text{begin}(). + \epsilon$ and $\text{end}(). = \text{begin}(). + \epsilon$, and hence $\sigma_2 \in \text{TimeOut}(G)$.

Next consider $\sigma_2$. Since $(\sigma_2, \text{begin}().)\gamma \models \psi_2$, every program variable $x \in \text{var}(G)$ is invariant with respect to $\sigma_2$. By the assumption, any program variable $y \notin \text{var}(S)$ is invariant with respect to $\sigma_2$. Hence, any program variable $x \notin \text{var}(S)$ is invariant with respect to $\sigma_2$. By Lemma 6.11, $[\sigma_2]dch(G) \cup dch(S) = [\sigma_2]dch(G)$ and then $[\sigma_2]dch(G) = [\sigma_2]dch(S)$. Using $dch(\sigma) \subseteq dch(G)$, we can derive $dch(\sigma_2) \subseteq dch(\sigma) \subseteq dch(G)$. By Lemma 6.4, $\sigma_2 = [\sigma_2]dch(G)$ and hence $\sigma_2 = [\sigma_2]dch(S)$. By Lemma 6.4 again, $dch(\sigma_2) \subseteq dch(S)$. Together with the well-formedness of $\sigma_2$, $(\sigma_2, \text{begin}().)\gamma \models \phi$, and the preciseness of $\phi$ for $S$, we obtain $\sigma_2 \in \mathcal{M}(S)$.

By $\sigma_2 = \sigma_2 \sigma_2$, $\sigma_2 \in \text{SEQ}(\text{TimeOut}(G), \mathcal{M}(S))$. By $\sigma = \sigma_1 \sigma_2$, $\sigma \in \text{SEQ}(\mathcal{M}(\text{delay } K_g), \text{TimeOut}(G), \mathcal{M}(S))$ and hence $\sigma \in \mathcal{M}(G)$.

Therefore all the cases lead to $\sigma \in \mathcal{M}(G)$. It is clear that $dch(\psi) = dch(G)$ and $\text{var}(\psi) = \text{var}(G)$. Hence, $\psi$ is precise for $G \equiv \{\|_{i=1}^{n} b_i; c_i \| x_i; S_i - S_i \| b; \text{delay } e \rightarrow S\}$.

**Iteration**

Consider $*G$. By the induction hypothesis, we can derive $G$ sat $\varphi$ where $\varphi$ is precise for $G$.

By the Iteration Rule, $*G$ sat $\psi$ with $\psi \equiv (b_G \land \varphi)C^* (\neg b_G \land \varphi)$. We prove that $\psi$ is precise for $*G$. By the Soundness Theorem 6.3, $*G$ sat $\psi$ is valid. Consider a well-formed model $\sigma$ such
that \( dch(\sigma) \subseteq dch(*G) \) and every program variable \( y \notin \text{wvar}(*G) \) is invariant with respect to \( \sigma \). Thus, \( dch(\sigma) \subseteq dch(G) \) and every program variable \( y \notin \text{wvar}(G) \) is invariant with respect to \( \sigma \). Let \( \gamma \) be an arbitrary environment. Assume \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \psi \). By definition of the \( C^* \) operator, there are two possibilities:

1. Either there exists a \( k \geq 1 \) and models \( \sigma_1, \sigma_2, \ldots, \sigma_k \) such that \( \sigma = \sigma_1 \sigma_2 \ldots \sigma_k \), for all \( j, \ 1 \leq j \leq k - 1 \), \( \end(\sigma_j) < \infty \), \( \langle \sigma_j, \text{begin}(\sigma_j) \rangle \gamma \models \neg b_G \land \varphi \), and if \( \end(\sigma_k) \leq \infty \), then \( \langle \sigma_k, \text{begin}(\sigma_k) \rangle \gamma \models \neg b_G \land \varphi \), otherwise \( \langle \sigma_k, \text{begin}(\sigma_k) \rangle \gamma \models b_G \land \varphi \);

2. Or there exist infinite models \( \sigma_1, \sigma_2, \ldots \) such that \( \sigma = \sigma_1 \sigma_2 \ldots \), for all \( j \geq 1 \), \( \end(\sigma_j) < \infty \), \( \langle \sigma_j, \text{begin}(\sigma_j) \rangle \gamma \models b_G \land \varphi \).

That is,

1. Either there exists a \( k \geq 1 \) and models \( \sigma_1, \sigma_2, \ldots, \sigma_k \) such that \( \sigma = \sigma_1 \sigma_2 \ldots \sigma_k \), for all \( j, \ 1 \leq j \leq k - 1 \), \( \end(\sigma_j) < \infty \), \( B(b_G)(\sigma_j(\text{begin}(\sigma_j)).\text{state}) \) (by Lemma 6.2). Since \( \sigma \) is well-formed, so are \( \sigma_1, \sigma_2, \ldots, \sigma_k \). Using \( dch(\sigma) \subseteq dch(G) \), we obtain \( dch(\sigma_j) \subseteq dch(G) \). Together with the invariance of program variables \( y \notin \text{wvar}(G) \) and the preciseness of \( \varphi \) for \( G, \ \sigma_j \in M(G) \). Similarly, \( \sigma_k \in M(G) \). If \( \end(\sigma_k) \leq \infty \), by Lemma 6.2, \( B(-b_G)(\sigma_k(\text{begin}(\sigma_k)).\text{state}) \). Otherwise \( \end(\sigma_k) = \infty \);

2. Or there exist infinite models \( \sigma_1, \sigma_2, \ldots \) such that \( \sigma = \sigma_1 \sigma_2 \ldots \), for all \( j \geq 1 \), \( \end(\sigma_j) < \infty \), \( B(b_G)(\sigma_j(\text{begin}(\sigma_j)).\text{state}) \), and \( \sigma_j \in M(G) \).

Therefore those two cases lead to \( \sigma \in M(*G) \). It is obvious that \( dch(\psi) = dch(*G) \) and \( \text{var}(\psi) = \text{var}(*G) \). Hence, \( (b_G \land \varphi)C^*(-b_G \land \varphi) \) is precise for \( *G \).

**Parallel Composition**

Consider \( S = S_1 || S_2 \). By the induction hypothesis, we can derive \( S_1 \text{ sat } \varphi_1 \) and \( S_2 \text{ sat } \varphi_2 \) with \( \varphi_1 \) and \( \varphi_2 \) precise for \( S_1 \) and \( S_2 \), respectively. From preciseness, \( dch(\varphi_i) \subseteq dch(S_i) \) and \( \text{var}(\varphi_i) \subseteq \text{var}(S_i) \), for \( i = 1, 2 \). Then we can apply the General Parallel Composition Rule, obtaining \( S_1 || S_2 \text{ sat } \psi \) with \( \psi \equiv (\varphi_1 \land (\varphi_2 \land c \psi_1)) \lor (\varphi_2 \land (\varphi_1 \land c \psi_2)) \) where \( \psi_i \equiv \text{inv}(\text{var}(S_2)) \land \Box \text{empty}(dch(S_2)) \) and \( \psi_2 \equiv \text{inv}(\text{var}(S_1)) \land \Box \text{empty}(dch(S_1)) \). We prove that \( \psi \) is precise for \( S_1 || S_2 \).

By the Soundness Theorem 6.3, \( S_1 || S_2 \text{ sat } \psi \) is valid. Let \( \sigma \) be a well-formed model such that \( dch(\sigma) \subseteq dch(S_1 || S_2) \) and any program variable \( y \notin \text{wvar}(S_1 || S_2) \) is invariant with respect to \( \sigma \). Let \( \gamma \) be an arbitrary environment. Assume \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \psi \). We show \( \sigma \in M(S_1 || S_2) \).

By the well-formedness of \( \sigma \), for any \( c \in \text{CHAN} \), for any \( \tau \), \( \begin{align*} \text{begin}(\sigma) & \leq \tau < \end(\sigma), \neg(\exists \ell \in \sigma(\tau).\text{comm} \land c \in \sigma(\tau).\text{comm}) \end{align*} \) is valid. Suppose \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \land (\varphi_2 \land c \psi_1) \). Define \( \sigma_1 \) as \( [\sigma \uparrow \text{var}(S_1)]_{\text{dch}(S_1)} \). From \( \langle \sigma, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \) and \( \text{var}(\varphi_1) \subseteq \text{var}(S_1) \), Lemma 6.10 leads to \( \langle \sigma \uparrow \text{var}(S_1), \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \). By \( dch(\varphi_1) \subseteq dch(S_1) \) and Lemma 6.9, we obtain \( \langle [\sigma \uparrow \text{var}(S_1)]_{\text{dch}(S_1)}, \text{begin}(\sigma) \rangle \gamma \models \varphi_1 \). Since \( \sigma \) is well-formed, \( \sigma_1 \) is also well-formed. By the definition of \( \sigma_1 \), any program variable \( y \notin \text{wvar}(S_1) \) is invariant with respect to \( \sigma_1 \). Together with the preciseness of \( \varphi_1 \) for \( S_1 \) and \( dch(\sigma_1) \subseteq dch(S_1) \), we obtain \( \sigma_1 \in M(S_1) \).
Next consider \((\sigma, \text{begin}(\sigma)) \gamma \models \varphi_2 \land C \psi_1\). By the definition of \(C\) operator, there exist models \(\sigma_3\) and \(\sigma_4\) such that \(\sigma = \sigma_3 \sigma_4\), \((\sigma_3, \text{begin}(\sigma_3)) \gamma \models \varphi_2\), and \((\sigma_4, \text{begin}(\sigma_4)) \gamma \models \psi_1\). Define \(\sigma_2\) as 
\[
[\sigma_3 \uparrow \text{var}(S_2)]_{dch(S_2)}\gamma 
\]
Since \(\text{var}(\varphi_2) \subseteq \text{var}(S_2)\), Lemma 6.10 leads to
\[
((\sigma_3 \uparrow \text{var}(S_2), \text{begin}(\sigma_3)) \gamma \models \varphi_2). \quad \text{From } dch(\varphi_2) \subseteq dch(S_2) \text{ and Lemma 6.9, we obtain }
\]
\[
[\sigma_3 \uparrow \text{var}(S_2)]_{dch(S_2)}(\sigma_3) \gamma \models \varphi_2, \text{ i.e. } (\sigma_2, \text{begin}(\sigma_2)) \gamma \models \varphi_2. \quad \text{Since } \sigma \text{ is well-formed, so is } \sigma_2. \quad \text{By the definition of } \sigma_2, \text{ any program variable } y \notin \text{var}(S_2) \text{ is invariant with respect to } \sigma_2.
\]
Together with the preciseness of \(\varphi_2\) for \(S_2\) and \(dch(\sigma_2) \subseteq dch(S_2)\), we obtain \(\sigma_2 \in \mathcal{M}(S_2)\).

Notice that \(\text{end}(\sigma) = \text{end}(\sigma_3 \sigma_4) \geq \text{end}(\sigma_3) = \text{end}(\sigma_2) \) and \(\text{end}(\sigma) = \text{end}(\sigma_1)\), hence \(\text{end}(\sigma) = \text{max}(\text{end}(\sigma_1), \text{end}(\sigma_2))\). Using the definition of \(\sigma_1, \sigma_2, \text{inv}, \text{ and } \text{empty}\), we can derive that, for \(i = 1, 2\),
\[
[\sigma]_{dch(S_1)}(\tau).\text{comm} = \begin{cases} 
\sigma_i(\tau).\text{comm} & \text{begin}(\sigma_i) \leq \tau < \text{end}(\sigma_i) \\
\varnothing & \text{end}(\sigma_i) \leq \tau < \text{end}(\sigma)
\end{cases}
\]
\[
(\sigma \uparrow \text{var}(S_1))(\tau).\text{state} = \begin{cases} 
\sigma_i(\tau).\text{state} & \text{begin}(\sigma_i) \leq \tau \leq \text{end}(\sigma_i) \\
\sigma_i(\text{end}(\sigma_i)).\text{state} & \text{end}(\sigma_i) < \tau \leq \text{end}(\sigma)
\end{cases}
\]
Thus \(\sigma \in \mathcal{M}(S_1 \parallel S_2)\).

Similarly, if \((\sigma, \text{begin}(\sigma)) \gamma \models \varphi_2 \land (\varphi_1 \land C \psi_2)\), we can also prove that \(\sigma \in \mathcal{M}(S_1 \parallel S_2)\).

It is quite obvious that \(dch(\psi) = dch(\varphi_1) \cup dch(\varphi_2) \cup dch(S_1) \cup dch(S_2) = dch(S_1 \parallel S_2)\) and \(\text{var}(\psi) = \text{var}(S_1) \cup \text{var}(S_2) = \text{var}(S_1 \parallel S_2)\). Therefore we have proved that \(\psi\) is indeed precise for \(S_1 \parallel S_2\).
**In this series appeared:**

<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>89/1</td>
<td>E.Zs.Lepoeter-Molnar</td>
<td>Reconstruction of a 3-D surface from its normal vectors.</td>
</tr>
<tr>
<td>89/2</td>
<td>R.H. Mak, P.Struik</td>
<td>A systolic design for dynamic programming.</td>
</tr>
<tr>
<td>89/3</td>
<td>H.M.M. Ten Eikelder, C. Hemerik</td>
<td>Some category theoretical properties related to a model for polymorphic lambda-calculus.</td>
</tr>
<tr>
<td>89/4</td>
<td>J.Zwiers, W.P. de Roever</td>
<td>Compositionality and modularity in process specification and design: A trace-state based approach.</td>
</tr>
<tr>
<td>89/5</td>
<td>Wei Chen, T.Verhoeff, J.T.Udding</td>
<td>Networks of Communicating Processes and their (De-)Composition.</td>
</tr>
<tr>
<td>89/6</td>
<td>T.Verhoeff</td>
<td>Characterizations of Delay-Insensitive Communication Protocols.</td>
</tr>
<tr>
<td>89/7</td>
<td>P.Struik</td>
<td>A systematic design of a parallel program for Dirichlet convolution.</td>
</tr>
<tr>
<td>89/9</td>
<td>K.M. van Hee, P.M.P. Rambags</td>
<td>Discrete event systems: Dynamic versus static topology.</td>
</tr>
<tr>
<td>89/10</td>
<td>S.Ramesh</td>
<td>A new efficient implementation of CSP with output guards.</td>
</tr>
<tr>
<td>89/11</td>
<td>S.Ramesh</td>
<td>Algebraic specification and implementation of infinite processes.</td>
</tr>
<tr>
<td>89/12</td>
<td>A.T.M.Aerts, K.M. van Hee</td>
<td>A concise formal framework for data modeling.</td>
</tr>
<tr>
<td>89/14</td>
<td>H.C.Haesen</td>
<td>ELDA, data manipulatie taal.</td>
</tr>
<tr>
<td>89/15</td>
<td>J.S.C.P. van der Woude</td>
<td>Optimal segmentations.</td>
</tr>
<tr>
<td>89/16</td>
<td>A.T.M.Aerts, K.M. van Hee</td>
<td>Towards a framework for comparing data models.</td>
</tr>
<tr>
<td>89/17</td>
<td>M.J. van Diepen, K.M. van Hee</td>
<td>A formal semantics for Z and the link between Z and the relational algebra.</td>
</tr>
<tr>
<td>Page</td>
<td>Author(s)</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>90/2</td>
<td>K.M. van Hee P.M.P. Rambags</td>
<td>Dynamic process creation in high-level Petri nets, pp. 19.</td>
</tr>
<tr>
<td>90/4</td>
<td>A. Peeters</td>
<td>Decomposition of delay-insensitive circuits, p. 25.</td>
</tr>
<tr>
<td>90/5</td>
<td>J.A. Brzozowski J.C. Ebergen</td>
<td>On the delay-sensitivity of gate networks, p. 23.</td>
</tr>
<tr>
<td>90/8</td>
<td>M.B. Josephs</td>
<td>Receptive Process Theory, p. 16.</td>
</tr>
<tr>
<td>90/9</td>
<td>A.T.M. Aerts P.M.E. De Bra K.M. van Hee</td>
<td>Combining the functional and the relational model, p. 15.</td>
</tr>
<tr>
<td>90/10</td>
<td>M.J. van Diepen K.M. van Hee</td>
<td>A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).</td>
</tr>
<tr>
<td>90/11</td>
<td>P. America F.S. de Boer</td>
<td>A proof system for process creation, p. 84.</td>
</tr>
<tr>
<td>90/12</td>
<td>P. America F.S. de Boer</td>
<td>A proof theory for a sequential version of POOL, p. 110.</td>
</tr>
<tr>
<td>90/14</td>
<td>F.S. de Boer</td>
<td>A proof system for the language POOL, p. 70.</td>
</tr>
<tr>
<td>90/15</td>
<td>F.S. de Boer</td>
<td>Compositionality in the temporal logic of concurrent systems, p. 17.</td>
</tr>
<tr>
<td>90/16</td>
<td>F.S. de Boer C. Palamidessi</td>
<td>A fully abstract model for concurrent logic languages, p. 23.</td>
</tr>
<tr>
<td>90/17</td>
<td>F.S. de Boer C. Palamidessi</td>
<td>On the asynchronous nature of communication in logic languages: a fully abstract model based on sequences, p. 29.</td>
</tr>
<tr>
<td>Volume</td>
<td>Author(s)</td>
<td>Title</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>91/02</td>
<td>R.P. Nederpelt, H.C.M. de Swart</td>
<td>Implication. A survey of the different logical analyses &quot;if...,then...&quot;, p. 26.</td>
</tr>
<tr>
<td>91/03</td>
<td>J.P. Katoen, L.A.M. Schoenmakers</td>
<td>Parallel Programs for the Recognition of P-invariant Segments, p. 16.</td>
</tr>
<tr>
<td>91/05</td>
<td>D. de Reus</td>
<td>An Implementation Model for GOOD, p. 18.</td>
</tr>
<tr>
<td>91/06</td>
<td>K.M. van Hee</td>
<td>SPECIFICATIEMETHODOEN, een overzicht, p. 20.</td>
</tr>
<tr>
<td>91/07</td>
<td>E. Poll</td>
<td>CPO-models for second order lambda calculus with recursive types and subtyping, p.</td>
</tr>
<tr>
<td>91/12</td>
<td>E. van der Sluis</td>
<td>A parallel local search algorithm for the travelling salesman problem, p. 12.</td>
</tr>
<tr>
<td>91/14</td>
<td>P. Lemmens</td>
<td>The PDB Hypermedia Package. Why and how it was built, p. 63.</td>
</tr>
</tbody>
</table>