On the existence of sound and complete axiomatizations of the monitor concept

Citation for published version (APA):

Document status and date:
Published: 01/01/1987

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.
On the Existence of Sound and Complete Axiomatizations of the Monitor Concept

by

Rob Gerth
On the Existence of Sound and Complete Axiomatizations of the Monitor Concept

by

Rob Gerth

January, 1989
ON THE EXISTENCE OF SOUND AND COMPLETE AXIOMATIZATIONS OF THE MONITOR CONCEPT

Rob Gerth*1
Eindhoven University of Technology 1
January 1989

Abstract
This paper presents an axiomatization for the partial correctness properties of Communicating Modules, a monitor-based programming language. This axiomatization is certified through soundness and (relative) completeness proofs, which constitute the major part of the paper. The system is based on the well-known notions of cooperation and interference freedom, however does not incorporate them as second order notions and is syntax-directed in a formal way.

1 Introduction
Monitors constitute the first attempt at formulating concurrent programming concepts beyond low-level mutual exclusion primitives such as semaphores ([Dij68]). The concept originates from Hoare and Brinch Hansen ([Hoa74, BHa73]) in their work on resource management in operating systems and is influenced by the SIMULA class concept ([DMN67]).

Monitors combine ideas from abstract data types with a notion of concurrency that derives from the independence of the various sets of data in the system achieved through the aforementioned data-abstraction. Programming languages based on it, include Concurrent Pascal [BHa75], Distributed Processes [BHa78], Mesa [MMS79] and most recently, Wirth's Modula (-2) [Wir84]. Also, the rendezvous concept of Ada [ARM83] clearly derives from it. In a different but related area, these ideas surface in the concept of object in object oriented systems [Am86].

Although the monitor concept is a relatively old one, it has never been adequately axiomatized (in the sense of [Hoa69]). It is well known that previous attempts, [Hoa74, How76], resulted in sets of rules that are incomplete in the technical sense of the word. Recent attempts to correct this, [ABSI], remained unsuccessful.

This contrasts sharply with work on other concepts of concurrent programming: Concurrent languages based on the sharing of variables between concurrent processes were successfully axiomatized by Owicki, [OG76]. Hoare's "Communicating Sequential Processes" has been dealt with in [AFdeR80, LG81, Sou84] and Ada in [GdeR84].

*1 The author was financially supported, through the Foundation for Computer Science Research in the Netherlands (SION), by the Netherlands Organisation for the Advancement of Pure Research (ZWO).
1 The author is currently working in and partially supported by ESPRIT project 937: "Debugging and Specification of Ada Real-Time Embedded Systems (DESCARTES)".
Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, The Netherlands. Electronic mail: mcvax!eutrc3!wsinrobg.UUCP or WSDCROBGC HEITUE5.BITNET.
The current paper, implicitly, offers an explanation for this state of affairs. The reported axiomatisation in fact combines the notions of interference freedom — developed to axiomatize shared variable languages — and cooperation (in its generalized form of [GdeR84]) — introduced to axiomatize synchronously communicating languages. Both notions, although formulated in different ways, reflect the insight that the specification or verification of a concurrent agent in isolation from an interacting environment, has to incorporate assumptions about such interactions. Assumptions that have to be proved compatible or non-conflicting when combining the separate agents' specifications. It is this basic insight that is missing in earlier axiomatisation attempts.

The present axiomatisation evolved from work on Brinch Hansen's Distributed Processes, DP ([BH78]), reported in [RDKdeR81, GdeRR82a, GdeRR82b] with the following difference: usually, cooperation and interference freedom is formulated as the way to combine proofs of specifications rather than as the way to combine specifications themselves. Hence, the resulting proof systems might be better called verification algorithms, instead of formal axiomatisations. The paper shows that this traditional rendering is not forced by these notions, and gives an axiomatisation that complies with the usual meaning of the term. This axiomatisation is syntax-directed in the sense that every program constructor has an associated formal proof rule and every basic statement an associated axiom.

The current paper is a companion to [GdeR86]. That paper informally introduces and motivates the proposed axiomatisation. Here, the concentration is on the formal justification of the proof system. Formally speaking, this paper independent from [GdeR86]. However, the reader seeking (more) motivation is referred to that paper.

For a different approach towards axiomatizing monitors in the context of DP, the reader should consult [SS85, Gje88]. Those works are based on the Soundararajan-Dahl approach, [SoD82] and are most easily described as the rendering of an operational semantics into proof rules and assertions. The resulting set of rules is complete, too. In this approach one reasons directly with the computation history, whereas here one may choose auxiliary variables that retain just the needed information about the history. This has the advantage of ease of use; it has the technical disadvantage that the completeness of the proof system is more restricted. See also Section 7.3 and the conclusions in Section 8.

The next section introduces the language Communicating Modules, CM. Then its syntax is extended to provide "hooks" for the axiomatisation presented in Section 5. Section 3 gives a syntax directed operational semantics that will be used in the soundness and completeness proofs. Next, in Section 4, two essential theorems are proved: the local and global merging theorems. These theorems express when separate executions of program parts are compatible in the sense that they are part of an execution of the combination of the program parts. The theorems have non-trivial content, since the program parts may interact. Section 5 introduces the specification language and the proof system. The section concludes with a substitution lemma used in the next two sections. The soundness and completeness proofs are the subject of Section 6 and 7. A final discussion occupies Section 8.

This introduction concludes with

1.1 A summary of some notation
The (well-ordered) set of integers, \{0, 1, 2, \ldots\}, is denoted by \(\omega\); its cardinality by \(\aleph_0\). There is a denumerable, well-ordered set of variables \(\text{Var}\). For any \(n \in \omega\), \(\text{Var}_n \subseteq \text{Var}\) contains the...
first \( n \) variables. Hence \( \text{Var} = \bigcup_{n \in \mathbb{N}} \text{Var}_n \). If \( \Sigma \) is some signature or similarity type, then \( Tm(\Sigma) \) and \( L(\Sigma) \) denote the set of terms and first order formulae over \( \Sigma \). Elements of \( L(\Sigma) \) are usually denoted by letters \( p, q, r \); terms by \( e, t \) and variables by \( u, v, x, y, z \). As usual, \( p[e/x] \) stands for the term obtained by substituting the term or expression \( e \) for every free occurrence of \( x \) in \( t \). The set of free variables of a formula or term \( \varphi \) is denoted by \( FV(\varphi) \).

If \( \sim \) is some signature or similarity type, then \( Tm(\sim) \) and \( L(\sim) \) denote the set of terms and first order formulae over \( \sim \). Elements of \( L(\sim) \) are usually denoted by letters \( p, q, r \); terms by \( e, t \) and variables by \( u, v, x, y, z \). As usual, \( p[e/x] \) stands for the term obtained by substituting the term or expression \( e \) for every free occurrence of \( x \) in \( t \). The set of free variables of a formula or term \( \varphi \) is denoted by \( FV(\varphi) \).

\( \sim \)-structures or \( \sim \)-algebra's, \( A \), are defined as usual and give interpretations of the symbols in \( \Sigma \). Given a \( \Sigma \)-structure \( A \), states \( \sigma, \tau, \nu \) are partial functions from \( \text{Var} \) into \( |A| \), the universe of \( A \). The domain of a state (or any other function) \( \sigma \), is \( \text{Dom}(\sigma) \); its range is \( \text{Ran}(\sigma) \). \( \Omega \) denotes the totally undefined state; i.e., the partial function \( \text{Var} \mapsto |A| \) with \( \text{Dom}(\Omega) = \emptyset \). Define \( \text{St}_n = \{ \sigma : \text{Var} \mapsto |A| \mid \text{Dom}(\sigma) \subseteq \text{Var}_n \} \) and \( \text{St} = \bigcup_{n \in \mathbb{N}} \text{St}_n \). For a state \( \sigma \), \( \sigma \upharpoonright V \) restricts the domain of \( \sigma \) to \( V \). Let \( \sigma(a/x) \) (\( a \in |A| \)) denotes the state \( \sigma' \) such that for \( y \neq x \), \( \sigma'(y) = \sigma(y) \) and \( \sigma'(x) = a \). For states \( \sigma, \tau \) such that \( \sigma \upharpoonright W = \tau \upharpoonright W \) where \( W = \text{Dom}(\sigma) \cap \text{Dom}(\tau) \), \( \sigma \cup \tau \) is defined in the obvious way. For any states, \( \sigma \) and \( \tau \), let \( \sigma(\tau) = \sigma(\tau(x)/x, x \in \text{Dom}(\tau)) \) denotes the state \( \sigma' \) such that for \( y \neq x \), \( \sigma'(y) = \sigma(y) \) and \( \sigma'(x) = \tau \). The value of a term \( t \) in a state \( \sigma(t) \), and the truth-value or satisfaction of a formula \( p \) in a state \( \sigma \), \( \sigma \models p \) or \( \mathcal{A}, \sigma \models p \) are defined as usual, provided \( FV(p) \subseteq \text{Dom}(\sigma) \), and are undefined otherwise. In the sequel, states are always assumed to have large enough domains so that they valuate every variable in a term or formula used. Write \( \mathcal{A} \models p \) if \( p \in L(\sim) \) is valid in \( \mathcal{A} \), i.e., if \( \mathcal{A}, \sigma \models p \) holds for every state \( \sigma \) (for which \( FV(p) \subseteq \text{Dom}(\sigma) \)). Let \( \text{Th}(\mathcal{A}) \) denote the set \( \{ p \in L(\Sigma) \mid \mathcal{A} \models p \} \), where \( \Sigma \) is \( \mathcal{A} \)'s signature. The term computable will be used in its informal sense; recursive will be its formalisation.

2 The language communicating modules, CM

As stated in the introduction, monitors combine

1. the notion of abstract data type, where the concepts of data and the set of legal (=meaningful) operations on it, are seen as logically inseparable, with

2. a notion of concurrency that derives from the logical independence of the various data sets within a system, originating in the abstraction of (1).

Interactions within a system occur through the data type interface. The logical integrity of the data type abstraction and the functionality of the operations are maintained by

1. imposing strict synchronization between a monitor and the entity requesting its service, while performing this service and by

2. allowing a monitor to be serving at most one request at a time.

This does not imply that the data type operations of a monitor are strictly serialized, but merely that a monitor cannot be serving two or more requests at the same time. In fact, it implies the possibility of a finer grain of serialization in monitors than there is in sequential abstract data types.

When a monitor is servicing a requestor, hence is performing a data type operation, the requestor is said to be active within the monitor and to have acquired the monitor-lock, effectively locking out (the servicing of) any other requestor until the former relinquishes the lock. At such a time a monitor may start (again) servicing someone else.
Even at this level of description one can discern an interleaving of actions: actions associated with the data type operations and bracketed in time by the acquiring and relinquishing of the monitor lock by the requestors of the particular operations.

Traditionally, a monitor is a passive object, only reacting to external requests. Modules are a generalization of this concept and may also initiate requests to other modules.

2.1 Syntax

Let $\Sigma$ be a signature. Partition $\text{Var}$ as $\text{Var}^R \cup \text{Var}^{IC}$, $\text{Var}^{IC} = \{i_{c_i} \mid i \in \omega\}$. The programming language $CM(\Sigma)$ is generated by the following Backus-Naur grammar ($\Sigma, \alpha, \beta \subseteq \text{Var}^R$, $\gamma \in \text{Var}^R$, $\varepsilon \subseteq Tm(\Sigma)$ and $b \in L(\Sigma)$ quantifier-free):

- $CM(\Sigma) ::= [MD_1 | \cdots | MD_n]$  $n \geq 1$
- $MD ::= M_i ::= DC : [S]$
- $DC ::= a_j(\alpha \# \varepsilon) : [S] DC$
- $S ::= z := e | \text{wait} | \text{call } M_i.a_j(\# \varepsilon) \mid S_1 ; S_2 | \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi } | \text{while } b \text{ do } S \text{ od}$

Note that no variable of the form $i_{c_i}$ can appear in any $CM(\Sigma)$-program. These so-called instance counters have a special meaning in the proof system of Section 5.2. The non-terminal $DC$ may also produce the empty string.

2.2 Informal semantics

A module consists of a number of procedure-declarations and an initial statement. Execution of a module starts with its initial statement and continues until either the initial statement terminates or a wait-statement is encountered. Such statements are the primary way to establish synchronization.

Synchronization occurs in a module:

- when arriving at a wait-statement,
- upon completing the execution of the initial statement, and
- upon completing the execution of an (instance of an) entry procedure.

Such points are called waiting points.

By synchronisation is meant either

- the act of honouring and arbitrarily selected call to one of the module's entries (global synchronisation) or
- the act of passing some wait-statement having been a waiting point at an earlier time and hence possibly different from the one execution just arrived at (local synchronisation).

Observe that at waiting points the monitor-lock is relinquished.

Honouring a call constitutes the only form of communication and synchronisation between modules. It results in a new procedure-instance being activated and executed. Until its completion, the caller remains fully synchronized, whence its execution suspended, with the callee. The parameter-mechanism is call-by-value-result and within a procedure-body no assignment to any
value-parameter is allowed. Formal parameters are local with respect to the procedure-body. All other variables are global with respect to the module. Modules do not share variables.

In Section 3, the meaning of a program or module is formally defined in the form of an operational semantics.

The following notation and conventions will be used throughout the paper:

- the modules in a program comprising of \( n \) modules, are named \( M_1, M_2, \ldots, M_n \);
- module \( M_i \) contains procedures \( a_{i1}, a_{i2}, \ldots, a_{in} \); its initial statement is denoted by \( a_{i0} \);
- procedure \( a_{ij} \) in \( M_i \) (\( j \geq 0 \)) is declared as \( a_{ij}(u_{ij}, v_{ij});[S_j] \); the initial statement, \( a_{i0} \), as \( :[S_0] \);
- \( L_{i,j} = \{ u_{ij}, v_{ij} \} \) \( (L_{i,0} = \emptyset) \); \( V_{i,j} \) is the set of variables, not including \( L_{i,j} \), that appear free within procedure \( a_j \) of module \( M_i \); \( V_i = \bigcup_{j=0}^{n} V_{i,j} \), \( L_i = \bigcup_{j=0}^{n} L_{i,j} \).

I shall usually ignore the superscripts in procedure-names. Note that by this convention, I assume procedure-names to be unique. \( L_{i,j} \) collects the local variables of procedure \( a_{ij} \).

The following context conditions apply to any program:

- \( V_i \cap V_j = \emptyset \) if \( i \neq j \),
- \( L_{i,k} \cap L_{j,l} = \emptyset \) if \( i \neq j \) or \( k \neq l \),
- \( V_i \cap \bigcup_{j=1}^{n} L_j = \emptyset \),
- in any statement call \( M_i.a_j(e\#f) \), the variables in \( f \) are pairwise disjoint and \( FV(e) \cap \{e\} = \emptyset \).

2.3 The extended language, \( CM^e(\Sigma) \)

Some syntactic extensions to \( CM(\Sigma) \) are necessary in order to obtain the proof system of Section 5. Specifically, labels are introduced and so-called bracketed sections.

Introduce a denumerable set, \( Lab \), of labels with typical elements \( l, l', l'', \ldots \) such that \( Lab \cap \text{Var} = \emptyset \). In the grammar generating \( CM^e(\Sigma) \), the same convention as in Section 2.1 is used.

\[
CM(\Sigma) ::= [MD_1 || \cdots || MD_n] \quad n \geq 1
\]

\[
MD ::= M_i :: DC : [S,l]
\]

\[
DC ::= a_j(u\#v) : [T_1; l); S-l'.(; T_2) \quad DC |
\]

\[
T ::= S |
\]

\[
S ::= e \mid l . \text{wait}.l' \mid l.(;T_1; l'.call M_i.a_j(e\#\bar{e}); T_2; l'.) \mid S_1; S_2 |
\]

if \( b \) then \( S_1 \) else \( S_2 \) fi | while \( b \) do \( S \) od

In a procedure body, \( T_1 \) and \( T_2 \) are called the prelude respectively the postlude of the procedure. Statements \( 'l.(;T_1;l'.call M_i.a_j(e\#\bar{e}); T_2;l'.) ' \) are called bracketed sections. A label that either labels the front of a wait or the initial statement will be said to label a waiting point.

In addition to the ones introduced in Section 2.2, the following context conditions apply to \( CM^e(\Sigma) \) programs:
any label may have at most one occurrence

the statements $T_1$ and $T_2$ within bracketed sections, may contain no wait or call-statements.

These syntactic extensions induce no semantic differences, other than that executing a bracket, $l.$ or $l.)$, is equivalent with executing a skip-action (e.g. $x := x$).

For any $CM^e(\Sigma)$-statement, procedure, module or program $T$, $FLB(T)$ denotes the set of labels occurring in $T$. If $l \in FLB(\alpha_j)$ and $\alpha_j$ is a procedure of $M_i$ then $L(l)$ denotes $L_{i,j}$.

3 Operational semantics for $CM^e(\Sigma)$

In this section the behaviour or semantics of $CM^e(\Sigma)$ programs is formally defined. I will use the structured operational semantics approach as advocated by G. Plotkin [Pl82].

3-1 DEFINITION. A labeled $\Sigma$ transition system, $lts_\Sigma$, is a tuple $(C, \rightarrow, SC, TC, TL, A)$, where

- $C$ is a set of configurations
- $SC \subseteq C$ is a set of start configurations
- $TC \subseteq C$ is the set of terminal configurations
- $TL$ is a set of transitions labels or records
- $\rightarrow \subseteq C \times TL \times C$ is the transition relation
- $A$ is a $\Sigma$-structure

A structured operational semantics is an $lts$ which transition relation is defined using induction on the structure of the configurations. In the current case this implies that also the semantics of $CM^e(\Sigma)$ modules and statements has to be defined, independent of a surrounding program.

To define the set of configurations, $CM^e(\Sigma)$ has to be extended, again, to $CM^*(\Sigma)$:

3-2 DEFINITION. $CM^*(\Sigma)$ is generated as in Section 2.3 except that the productions for the non-terminals $DC$ and $S$ are replaced by

- $DC ::= \alpha_j(\bar{u}\#\bar{v}) : [S] DC$
- $S ::= x := e | l.wait.l' | block.l | l.(| l.) | end_{e,i,j} | l.callM_i,\alpha_j(\bar{e}\#\bar{v}) | l.rend_{e,i,j} | S_1; S_2 | if b then S_1 else S_2 | while b do S od | A$

A block.$l$-action signifies that a wait has been encountered and that the monitor lock was released at this point. In the configuration of a caller, $l.rend_{e,i,j}$ records that an instance of procedure $\alpha_j$ in $M_i$ has been activated but not yet completed. The set of value-result parameters of the call is $\bar{v}$. Statements $end_{e,i,j}$ are used to identify the caller (procedure $\alpha_j$ in $M_i$) which caused the current instance to be created. Here, $\bar{v}$ denotes the set of formal value-result parameters of the corresponding procedure. $A$ denoted the empty statement.

Fix some signature $\Sigma$ and a $\Sigma$-structure $A$. From now on, I assume that $\Sigma$ contains $+,-,0$ and 1 and that $A$ contains as sub-structure the standard model $(\omega,+, -,0,1)$. This assumption is needed to valuate the instance counters and, later on, is needed to reason about them. The semantics of $CM(\Sigma)$-programs, however, is not influenced.
3-3 Definition (The configuration set \( C \)).

- A local configuration, \( LC \), has the form \( I_d : (S, \theta) \), where
  - \( I_d \equiv (i, j, k) \in \omega^3 \); \( i \) indexes a module, \( j \) a procedure and \( k \) characterizes the instance,
  - \( S \) is a \( CM^*(\Sigma) \)-statement or procedure,
  - \( \theta = (v, \rho) \), where \( v \in St \) with \( Dom(v) \subseteq \text{Var} \) and \( \rho \in \text{Lab} \cup \{0\} \).

- A module configuration \( MC \) has the form \( (X | \sigma) \) or \( (X | a, LC_0, ..., LC_n, \sigma) \), where
  - \( X \) is either \( \Lambda \) or \( a \) a \( CM(E) \) module,
  - \( a \in \{-1, 0, 1, ..., n\} \), the activity pointer
  - for \( 0 \leq i \leq n \), \( LC_i \) is a local configuration: \( I_d_i : (S_i, \nu_i, \rho_i) \) and \( I_d_j \neq I_d_i \) for \( i \neq j \),
  - \( \sigma \) is a state such that \( Dom(\sigma) \cap Dom(\nu_i) = \emptyset \) for \( i = 1 ... n \).

- A program configuration \( PC \), has the form \( (MC_0, ..., MC_m) \), where
  - \( MC_0, ..., MC_m \) are module configurations.

- The configuration set, \( C \), consists of all program and module configurations.

During execution of a module, several procedure instances coexist together with the module's initial statement. The local configurations record the progress in each of these instances. Since formal parameters are local variables, each \( LC \) has a local state; the label records control information. The identification, \( I_d \), of a local configuration indicates where the configuration originates from:

\( I_d \equiv (i, j, k) \neq (0, 0, k) \) implies that it records a configuration of procedure or initial statement \( a_j \) while \( k \) identifies the instance; \( I_d = (0, 0, k) \) gives no such information and is the "identification" of a configuration of some statement. The variables that are global in the module, are valued by the \( MC \)'s state. Its activity pointer points to the \( LC \) that is currently active or equals \(-1\), signifying a change of activity at a waiting point.

The variables in \( \text{Var} IC \) are proof-variables and are used to count the number of instances of a procedure. I assume that there is some canonical enumeration of the procedure names in a program, so that I can write \( ic.a_j \) for the \( \text{Var} IC \)-variable that refers to procedure \( a_j \) (in module \( M_i \)).

Conventions: As generic indicants for module configurations, \( C, C', C'', C_i, ... \) are used. \( C \) has the form \( (M_j | a \ | LC_0, ..., LC_m, \sigma) \) where \( LC_i \equiv I_d_i : (S_i, \theta_i) \) and \( \theta_i \equiv (\nu_i, \rho_i) \). In \( C^\sigma \) the components of \( C \) inherit the sub and superscripts (except for a module text):

\[
(M_j \ | \ a^\sigma_j \ | \ I_d_i^\sigma_{j,0} : (S_i^\sigma, \theta_i^\sigma_{j,0}), ..., I_d_m^\sigma_{j,m} : (S_m^\sigma, \theta_m^\sigma_{j,m}), \sigma_{j}^\sigma).
\]

Program configurations are generically denoted as \( C, C', ... \). \( C \) has the form \( (C^1, ..., C^n) \) and otherwise follows the above conventions. Finally, sometimes only part of a configuration is explicitly given; the other parts are then assumed to follow these conventions. Similar conventions hold for program configurations.

3-4 Definition (The transition label set, \( TL \)).

- \( TL = \{ \epsilon \} \cup \{ (i, j, h, \nu, \ell, \sigma) \mid i, j, h \in \omega, \nu \in St, \ell \in \text{Lab} \} \cup \{ (i, j, k, l, \text{ind}) \mid i, j, k, l \in \omega, \text{ind} \in \{\text{in}, \text{ex}\} \} \).

- Records of the form \( (i, j, \text{ind}) \) are called global records; the other ones are called local records.

7
Transition labels will be interpreted as words or sequences consisting of 0 or 1 records. Global records originate from module communications; θ is the sequence of transmitted values, the first index pair — a module and procedure index — specifies the sender, the second pair the receiver and ind indicates whether the record witnesses the initiation of a rendezvous (in) or the termination of one (te). Non-empty local records originate from waiting points and the index pair points to the originator. The pair of states, ν and σ, record the local state, ν, of the procedure instance and the global state, σ; ℓ identifies the waiting point. Finally, h characterizes the executing procedure instance.

3–5 Definition (The front label of a statement, S, \( \text{lab}(S) \)).
- for \( S \equiv x := e \), if \( b \) then \( S' \) else \( S'' \) fi, while \( b \) do \( S \) od, \( a(\bar{u}#\bar{v}) : [\ell,S] \), \( \text{ends}_{s,i,j} \) or \( \Lambda \):
  \( \text{lab}(S) = \emptyset \),
- for \( S \equiv \ell \cdot \text{wait} \cdot \ell' \), block \( \ell \), \( i \cdot (\ell, \ell') \), \ell \cdot \text{call} \ M_i, a_j(\bar{e}#\bar{z}) \) or \( \ell \cdot \text{rend}_{s,i,j} \): \( \text{lab}(S) = \{ \ell \} \)
- for \( S \equiv S_1; S_2 \): \( \text{lab}(S) = \text{lab}(S_2) \) if \( S_1 \equiv \Lambda \)

All is set now for the definition of the transition relation. In its definition, whenever an \( M \in C \) of the form \( (X, a_1 \cdots LC \cdots, \sigma) \), occurs, the shown \( LC \) is the one pointed to by \( a \), unless \( a = -1 \).

Write \( \lambda \in \gamma \in C' \) instead of \( (C, \lambda, C') \in \rightarrow \).

3–6 Definition (The transition relation, \( \rightarrow \)).
- \( \rightarrow \subseteq C \times TL \times C \) is the smallest relation satisfying
  1. \( (\Lambda \mid 0 \mid Id : (x := e, \nu, \emptyset), \sigma) \xrightarrow{\epsilon} (\Lambda \mid 0 \mid Id : (\Lambda, \nu', \emptyset), \sigma') \), where
     \( \nu', \sigma' = \nu(\nu \cup \sigma(e)/x), \sigma \) if \( x \in \text{Dom}(\nu) \)
     \( = \nu(\sigma(\nu \cup \sigma(e)/x)) \) if \( x \notin \text{Dom}(\nu) \)
  2. \( (\Lambda \mid 0 \mid Id : (\ell \cdot \text{wait} \cdot \ell', \nu', \ell), \sigma) \xrightarrow{\epsilon} (\Lambda \mid -1 \mid Id : (\text{block} \cdot \ell', \nu, \ell'), \sigma) \)
  3. \( (\Lambda \mid -1 \mid Id : (\text{block} \cdot \ell, \nu, \ell), \sigma) \xrightarrow{\epsilon} (\Lambda \mid 0 \mid Id : (\Lambda, \nu, \emptyset), \sigma) \)
  4. \( (\Lambda \mid 0 \mid Id : (\text{block} \cdot \ell, \nu, \ell), \sigma) \xrightarrow{\epsilon} (\Lambda \mid 0 \mid Id : (\Lambda, \nu, \emptyset), \sigma) \)
  5. \( (\Lambda \mid 0 \mid r, s, k : (\ell \cdot \text{call} \ M_i, a_j(\bar{e}#\bar{z}), \nu, \ell), \sigma) \xrightarrow{(r, s, \sigma \cup \nu(\bar{z}), i, j, in)} (\Lambda \mid 0 \mid r, s, k : (\ell \cdot \text{rend}_{s,i,j} \cdot \nu, \ell), \sigma) \)
  6. \( (\Lambda \mid 0 \mid r, s, k : (\ell \cdot \text{rend}_{s,i,j} \cdot \nu, \ell), \sigma) \xrightarrow{(i, j, \bar{a}, r, s, te)} (\Lambda \mid 0 \mid r, s, k : (\Lambda, \nu', \emptyset), \sigma') \)
where for \( x_k \in \bar{z} \)
  \( \nu'(x_k), \sigma'(x_k) = a_k, \sigma(x_k) \) if \( x_k \in \text{Dom}(\nu) \)
  \( = \nu(x_k), a_k \) if \( x_k \notin \text{Dom}(\nu) \)
  \( \nu' \uparrow \bar{z} \sigma = \nu \uparrow \bar{z} \sigma, \sigma' \uparrow \bar{z} \sigma = \sigma \uparrow \bar{z} \sigma \)
  7. \( (\Lambda \mid 0 \mid Id : (\ell, \Omega, \ell), \sigma) \xrightarrow{\epsilon} (\Lambda \mid -1 \mid Id : (\ell, \Omega, \ell), \sigma) \)
  8. \( (\Lambda \mid -1 \mid Id : (\ell, \Omega, \ell), \sigma) \xrightarrow{\epsilon} (\Lambda \mid 0 \mid Id : (\Lambda, \Omega, \ell), \sigma) \)
  9. \( (\Lambda \mid 0 \mid Id : (\ell, \nu, \ell), \sigma) \xrightarrow{\epsilon} (\Lambda \mid 0 \mid Id : (\Lambda, \nu, \emptyset), \sigma) \)

\(^1\)Remember that any state is implicitly assumed to evaluate all the appropriate variables.
10. \( \Delta \mid 0 \mid Id : (\ell, \nu, \ell), \sigma) \xrightarrow{\epsilon} (\Delta \mid 0 \mid Id : (\ell, \nu, \theta), \sigma) \\
11. \( \Delta \mid 0 \mid Id : (\Delta, \theta), \sigma) \xrightarrow{\epsilon} (\Delta \mid 0 \mid \sigma) \\
12. \( \Delta \mid 0 \mid i, j, k : (a^j_i (\bar{u} \# \bar{v}) : [\mathcal{S}], \nu, \theta), \sigma) \xrightarrow{(\tau, s, \bar{e}, \bar{e}, i, j, \text{in})} (\Delta \mid 0 \mid i, j, k : (S; \text{end}_{s, r, s, \Omega \{b, \bar{e}, \bar{v}, \theta\}}, \nu, \theta), \sigma), \) \\
where \( \bar{b} = \sigma \cup \nu (\bar{u}) \) and \( \bar{e} = \sigma \cup \nu (\theta) \) \\
13. \( \Delta \mid 0 \mid Id : ([S, \ell], \Omega, \theta), \sigma) \xrightarrow{\epsilon} (\Delta \mid 0 \mid Id : (S, \ell, \Omega, \theta), \sigma) \\
14. \( \Delta \mid 0 \mid i, j, k : (\text{end}_{\ell, r, s, \nu, \theta}, \sigma) \xrightarrow{(i, j, \nu(\theta), r, s, \text{te})} (\Delta \mid 0 \mid i, j, k : (A, \nu, \theta), \sigma) \) \\
15. \( \Delta \mid 0 \mid Id : (if \ b \ then \ S_1 \ else \ S_2 \ fi, \nu, \theta), \sigma) \xrightarrow{\epsilon} (\Delta \mid 0 \mid Id : (S_1, \nu, \text{lab}(S_1)), \sigma) \) \\
if \( \Delta, \sigma \cup \nu \vdash b \) \\
16. \( \Delta \mid 0 \mid Id : (if \ b \ then \ S_1 \ else \ S_2 \ fi, \nu, \theta), \sigma) \xrightarrow{\epsilon} (\Delta \mid 0 \mid Id : (S_2, \nu, \text{lab}(S_2)), \sigma) \) \\
if \( \Delta, \sigma \cup \nu \not\vdash b \) \\
17. \( \Delta \mid 0 \mid Id : (\text{while} \ b \ do \ S \ od, \nu, \theta), \sigma) \xrightarrow{\epsilon} (\Delta \mid 0 \mid Id : (S; \text{while} \ b \ do \ S \ od, \nu, \text{lab}(S)), \sigma) \) \\
if \( \Delta, \sigma \cup \nu \vdash b \) \\
18. \( \Delta \mid 0 \mid Id : (\text{while} \ b \ do \ S \ od, \nu, \theta), \sigma) \xrightarrow{\epsilon} (\Delta \mid 0 \mid Id : (\Delta, \nu, \theta), \sigma) \) \\
if \( \Delta, \sigma \cup \nu \not\vdash b \) \\
19. \( \Delta \mid a \mid Id : (S, \nu, \rho), \sigma) \xrightarrow{\lambda} (\Delta \mid a' \mid Id : (S', \nu', \rho'), \sigma') \) \\
where \( \bar{S} \equiv S'; T \) if \( S' \neq \Lambda \) and \( T \) otherwise \\
20. \( \Delta \mid a \mid Id : (S, \theta), \sigma) \xrightarrow{\lambda} (\Delta \mid a' \mid Id : (S', \theta'), \sigma') \) \\
provided \( (M \mid i \cdots Id : (\ell, \theta), \sigma) \) \\
21. \( \Delta \mid 0 \mid i, j, k : (a^j_i (\bar{u} \# \bar{v}) : [\mathcal{S}], \Omega, \theta), \sigma) \xrightarrow{\lambda} (\Delta \mid 0 \mid i, j : (S', \theta'), \sigma') \) \\
provided \( a_j \) is declared in \( M_i \) as in the premiss\(^2\), \( Id, i \not\in \{l, j, k \} \) for \( i < n \) and \( \sigma'' = \sigma' \{ic \cdot a^j_i + 1/ic \cdot a^j_i \} \). \\
\(^2\)In case \( j = 0 \), the declaration stands for the initial statement of \( M_i \) and I assume the parameter part to be empty.
3. Let \( \sigma' = \sigma \{ i.c.a_j^i - 1/i.c.a_j^i \} \).

24. \( (M_i \mid -1 \mid Id : (\ell, \theta), \sigma) \xrightarrow{\ell} (M_i \mid 0 \mid \sigma) \)

25. \( (M_k \mid \cdots) \xrightarrow{\lambda} (M_k \mid \cdots)' \)

\[
(\langle M_1 \mid \cdots \rangle \cdots \langle M_k \mid \cdots \rangle \cdots \langle M_n \mid \cdots \rangle) \xrightarrow{\lambda} (\langle M_1 \mid \cdots \rangle \cdots \langle M_k \mid \cdots \rangle' \cdots \langle M_n \mid \cdots \rangle)
\]

provided \( \lambda \) is local and \( (M_k \mid \cdots)' \neq (M_k \mid 0 \mid \tau) \) for any \( \tau \).

26. \( (M_i \mid \cdots) \xrightarrow{\lambda} (M_i \mid \cdots)' \), \( (M_j \mid \cdots) \xrightarrow{\lambda} (M_j \mid \cdots)' \)

\[
(\langle M_1 \mid \cdots \rangle \cdots \langle M_i \mid \cdots \rangle \cdots \langle M_j \mid \cdots \rangle \cdots \langle M_n \mid \cdots \rangle) \xrightarrow{\lambda}
(\langle M_1 \mid \cdots \rangle \cdots \langle M_i \mid \cdots \rangle' \cdots \langle M_j \mid \cdots \rangle' \cdots \langle M_n \mid \cdots \rangle)
\]

provided \( \lambda \) is global.

\[
((M_1 \mid -1 \mid Id : (\ell_1, \theta_1), \sigma_1), \ldots, (M_n \mid -1 \mid Id_n : (\ell_n, \theta_n), \sigma_n)) \xrightarrow{\ell} ((M_1 \mid 0 \mid \sigma_1), \ldots, (M_n \mid 0 \mid \sigma_n))
\]

- \( \to^* \) is the reflexive and transitive closure of \( \to \).

Remarks: The above definition intends to capture not only the behaviour of \( CM^*(\Sigma) \)-programs, but also the behaviour of modules and statements, independent from their context. So, e.g., the states in axiom 3 can change arbitrarily, reflecting the fact that nothing is known about the module text, hence about the statements that can be interleaved at waiting points. In contrast, rule 20 does not allow such changes. Likewise for axiom 8 versus axiom 24. Axioms 3, 7, 8 and 11 are irrelevant as far as the transitions of modules and programs are concerned. Additionally, rule 24 can be ignored for program transitions.

3–7 Definition (Start and terminal configurations). Let \( \tau \) be any state.

- for a statement or initial statement \( S \)
  \( SC_\tau(S) = \{(\Lambda \mid 0 \mid 0, 0 : (S, \Omega, lab(S)), \tau)\} \)
  \( TC_\tau(S) = (\Lambda \mid 0 \mid \tau) \)

- for a declaration, \( D \), of a procedure \( a_j^i \) \( (j \neq 0) \)
  \( SC_\tau(D) = \{(\Lambda \mid 0 \mid i, j, k : (D, \Omega, \theta), \tau) \mid k \in \omega \} \)
  \( TC_\tau(D) = (\Lambda \mid 0 \mid \tau) \)

- for a module \( M_i \) with initial statement \( S_0, \ell \)
  \( SC_\tau(M_i) = (M_i \mid 0 \mid i, 0, 0 : (S_0, \ell, \Omega, 0), \tau), \) where  \( \tau = \tau'0/ic.a_j^i \) \( j = 1..n_i \).  
  \( TC_\tau(M_i) = (M_i \mid 0 \mid \tau) \)

- for a program \( P \equiv [M_i \mid \cdots \mid M_n] \)
  \( SC_\tau(P) = (SC_\tauTV(M_i), \ldots, SC_\tauTV(M_n)) \)
- $TC_r(P) = (TC_rV_1(M_1), \ldots, SC_rV_n(M_n))$, where $V_i = \text{Var} \cup \bigcup_{j \neq i} V_j$
  \[ i \leq i \leq n \]

- $\text{SC} = \{ SC_r(X) \mid \tau \text{ a state, } X \text{ a } CM^*(\Sigma)\text{-statement, procedure, module or program} \}$

- $\text{TC} = \{ TC_r(X) \mid \tau \text{ a state, } X \text{ a } CM^*(\Sigma)\text{-statement, procedure, module or program} \}$

3-8 Definition (Configuration states, $s(C)$).

- for a module configuration $C \equiv (X \mid a) \vdash \text{Id} : (S_0, \theta_0), \ldots, \text{Id}_n : (S_n, \theta_n), \sigma,$

  \[ s(C) = \{ (\delta_0, \ldots, \delta_n, \sigma) \mid \text{for } 0 \leq i \leq n, \text{ if } a = -1 \text{ or } a = i \text{ then } \delta_i = \theta_i \text{ else } \delta_i = (\nu_i, \emptyset) \} \]

- for a program configuration $C = (MC_1, \ldots, MC_m)$

  \[ s(C) = \{ \{ \delta_1, \ldots, \delta_m \} \mid \delta_i \in s(MC_i), 1 \leq i \leq m \} \]

Configuration states extract from configurations the information on which satisfaction of specifications depend. Configuration states, usually denoted by $\delta, \delta', \ldots$, inherit the notational conventions as introduced for configurations.

3-9 Definition ($CM^*(\Sigma)$-computations and semantics).

For any $CM^*(\Sigma)$-statement, procedure, module or program $T$, set of states $X$ and set of configurations $Y$:

- Define

  \[ \text{Comp}(T, X, Y) = \{ C_0 \xrightarrow{\lambda_1} C_1 \xrightarrow{\lambda_2} \cdots \xrightarrow{\lambda_n} C_n \mid C_0 \in \text{SC}_r(T), \tau \in X, \forall 0 \leq i \leq n C_i \in Y \} \]

- The set of $(X, Y$-legal) computations, $\text{Comp}(T, X, Y)$, is defined by

  - $\text{Comp}(T, X, Y)$ if $T$ is a module or program, and
  - $\text{Comp}(T, X, Y) \cup$

    \[ \{ C_m \xrightarrow{\lambda_{m+1} \cdots \lambda_n} C_n \mid \exists C_0 \xrightarrow{\lambda_1 \cdots \lambda_m} C_m \text{ such that } \alpha_{m-1} = -1 \text{ and } C_0 \xrightarrow{\lambda_1 \cdots \lambda_n} C_n \in \text{Comp}(T, S_t, Y) \} \]

  if $T$ is a statement or procedure

- its semantics, $\llbracket T \rrbracket (X, Y)$, is

  \[ \{ (\delta_0 \cdots \delta_n, t) \mid \exists C_0 \xrightarrow{\lambda_1 \cdots \lambda_n} C_n \in \text{Comp}(T, X, Y) \text{ such that } \text{if } C_n = TC_{\alpha_n}(T) \text{ then } t = \bot \text{ otherwise } t = T \text{ and } \delta_i = s(C_i) \text{ for } 0 \leq i \leq n, \} \]

This definition of computation sequence, and hence of semantics, reflects the knowledge of the state at waiting points: only in modules and programs is the state known when execution resumes at a waiting point; for statements and procedures the state at these points is arbitrary (see, e.g., the transition rule 3 in Definition 3-6). For that reason, suffixes of computation sequences are
included, too. This will become clearer when defining satisfaction of specifications in Definition 5-3.

Observe that the usual notion of computation and semantics is obtained as $\text{Comp}(\cdot, \text{St}, C)$ and $[\tau](\text{St}, C)$.

As a matter of notation, if no restrictions are placed on states or configurations, the corresponding sets are not mentioned. E.g., instead of $\text{Comp}(\cdot, \text{St}, C)$ I shall write $\text{Comp}(\cdot)$. Moreover, whenever I associate configuration states with the configurations in a computation, the conditions of the second part of Definition 3-8 will be assumed.

For future use, there are four simple observations to be made:

The relation between instance-counters $(\text{Var}^{IC})$ and configurations is given by the following.

3-10 Observation. Let $M_t$ be some module, let $C_0 \xrightarrow{\lambda} C_n \in \text{Comp}(T)$ and let $a_k^i$ be some procedure in $M_t$. Then

1. $\sigma_n(\text{ic}, a_k^i) = m$ if there are $m$ LC's, $I_d^j : (S_j, \theta_j), \in C_n$ such that $I_d^j = (l, k, h)$ for some $h$ and for $j = 1..m$.

This is obvious.

3-11 Observation. Given, for $i = 0, 1, C_0 \xrightarrow{\lambda} \ast C_n \in \text{Comp}(M_t, \tau)$, assume that $\lambda_0 = \lambda_1$ and $\sigma_0 = \sigma_1$. Then, if for some $i$ and $j$, $\theta_0 = \theta_1$, $I_d^j, i \equiv I_d^j, i$ and $\rho_0 = \rho_1 = \emptyset$, conclude that $L\sigma_0 = L\sigma_1$.

If $M_t$ only contains straightline code (i.e., no while loops) this is clear. Next, observe that between any two successive arrivals "at" the procedure instance determined by $I_d^j, i$, at least one record is generated. Moreover, if this record is a global one, so that the generating procedure instance is not known, there is another local record that does identify the generating instance. In the sequel this observation shall mostly be used implicitly.

3-12 Observation. Let $C_0 \xrightarrow{\lambda} C_n \in \text{Comp}(M_t)$ with $a_n = -1$ and let $\nu \in \text{St}, \ell \in \text{Lab}$. Then the following two statements are equivalent:

1. $(\nu, \ell) = (\theta_n, i, \ell) \in \text{FLB}(a_j^i)$ and $I_d^i = (l, j, k)$
2. $\lambda = \lambda_0 \ast (\ell, j, k, \nu, \ell', \sigma) \ast \lambda_1$, where
   1. $\ell \in \text{FLB}(a_j^i)$,
   2. if $\ell$ labels the rear of a wait, then $\ell'$ labels its front and if $\ell$ labels the rear of $a_0^i$ then $\ell' = \ell$,
   3. $\lambda_1$ does not contain a local record of the form $(l, j, k, \nu, \ell, \sigma)$ for any $\nu, \sigma$ and $\ell$.

Notation: $(h, \ell, \nu)$ is determined by $\lambda$.

Because $a_n = -1$ every $\rho_n, i$ labels a waiting point. Hence, 'arriving' at any of the $\theta_n, i$'s has been witnessed by some local record. Consider the last time the computation arrived at $(\nu, \ell)$. Clearly, $\lambda$ has the form as suggested. Alternatively, if $\lambda$ has the above form then the configuration responsible for producing the local record cannot have moved during the $\lambda_1$-computation (remember, $a_n = -1$).
3–13 Observation. For any \( CM^*(\Sigma) \)-statement, procedure, module or program \( T \) and computable sets of states \( X \), and configurations, \( Y \), \( \text{Comp}(T,X,Y) \) and \( [T](X,Y) \) are computable relations.

4 The merging lemmas

The aim is a syntax directed axiomatization of communicating modules. Specifically, it must be possible (1) to combine the specifications of procedures and the initial statement into a module specification and (2) to combine module specifications into a program specification.

Now, specifications describe sets of allowed or legal computations of the specified program parts. Also, for completeness of the axiom system, specifications must be able to fully characterize the legal computations. Consequently the following question arises:

*Given computations of some statements. When are such computations compatible in the sense that they are sub-computations of the composite statement?*

In this section the two less trivial cases are treated. Section 4.1 contains the local merging lemma which states the conditions under which computations of a set of procedures and an initial statement can be combined into a single computation of the resulting module. In Section 4.2 the global merging lemma is formulated, which states when module-computations can be combined.

4.1 The local merging lemma, \( LML \)

4–1 Definition (local reachability, \( lr \)).

Let \( M_i \) be a \( CM^*(\Sigma) \)-module, \( \tau_0, \tau_1, \ldots, \tau_k, \sigma \in St \), \( K \) a set of configurations, \( \ell_0, \ldots, \ell_k \in \text{Lab} \), \( h_0, \ldots, h_k \in \omega \) and \( \lambda \) a record sequence. Define for \( i = 0 \ldots k \), \( j_i \in \text{FLB}(a_{\lambda,i}) \).

Then \( (h_0, \ell_0, \ldots, h_k, \ell_k) \) is \( \langle M_i, \tau, K, \nu_0, \ldots, \nu_k, \sigma, \lambda \rangle \)-\( lr \) iff

\[
\exists C_0 \xrightarrow{\lambda} C_n \in \text{Comp}(M_i, \{\tau\}, K) \text{ such that }
\]

- \( \sigma \upharpoonright V_1 = \sigma_n \upharpoonright V_2 \) and \( \lambda = \overrightarrow{\lambda} \),
- there is an injection, \( cf: \{0 \ldots k\} \mapsto \{0 \ldots m\} \), where \( m \) is the number of local records in \( C_n \), and \( \forall i = 0 \ldots k \ (\nu_i, \ell_i) = \theta_{n,cf}(i) \) and \( \text{Id}_{n,cf}(i) = (l, j_i, h_i) \)

So, configurations as characterized by \( (\nu_j, \ell_j) \), \( h_j \) and \( \sigma \), are locally reachable from some state \( \tau \), precisely if there is a computation of \( M_i \), starting in \( \tau \), reaching such a configuration via intermediate configurations in \( K \) and which associated record sequence is \( \lambda \).

4–2 Definition (LC-independence). Let \( C^0_i \xrightarrow{\lambda} C^0_n \in \text{Comp}(M_i, \tau, K) \) for \( i = 0, 1 \) and let \( C^1_n \in \mathcal{C} \) be obtained from \( C^0_n \) by replacing or adding LC's from \( C^1_n \). Then, \( K \) is called LC-independent if

\[
C^0 \xrightarrow{\lambda} C^0_n \in \text{Comp}(M_i, \tau) \implies C^0 \xrightarrow{\lambda} C^1_n \in \text{Comp}(M_i, \tau, K)
\]
LEMMA (Local Merging Lemma, LML).
With notation as in Definition 4-1, suppose that
- \((h_0, t_0, \ldots, h_{k-1}, t_{k-1})\) is \((M_1, \tau_0, K, \nu_0, \ldots, \nu_{k-1}, \sigma, \lambda)\)-ir,
- \((h_k, t_k)\) is \((M_1, \tau_1, K, \nu_k, \sigma, \lambda)\)-ir, and
- \(K\) is LC-independent.
Then \((h_0, \ell_0, \ldots, h_k, \ell_k)\) is \((M_1, \tau_1, K, \nu_0, \ldots, \nu_k, \sigma, \lambda)\)-ir, provided
- \(\tau_0 \cap \nu_i = \tau_1 \cap \nu_i = \tau_i\),
- \(\nu_i < k\),
- there is at most one \(\ell_i \in \text{FLB}(a_0^h)\) and
- at most one \(i\), does not label a waiting point.

PROOF. W.l.o.g., assume that \(\text{Dom}(\tau_i) \subseteq \nu_i\) for \(i \leq 2\) and that \(\text{Dom}(\sigma) \subseteq \nu_i\). Let \(\tau = \tau_0 = \tau_1 = \tau_2\). If \(\lambda\) does not contain local records, then \(t_0, \ldots, t_k \in \text{FLB}(a_0^h)\), so that the Lemma holds vacuously.

Otherwise, split \(\lambda\) as \(\lambda = \lambda_0 \cdot \lambda_1\), where \(\lambda_1\) is the largest suffix of \(\lambda\) that does not contain any local record. By assumption, there are computations, \(C_0 \xrightarrow{\lambda} C_m \xrightarrow{\lambda} C_n\) and \(\overline{C_0} \xrightarrow{\lambda} \overline{C_m} \xrightarrow{\lambda} \overline{C_n}\), (1)
both in \(\text{Comp}(M_1, \{\tau\}, K)\), such that \((\nu_i, \ell_i) = \theta_{n, cf(i)}\), \(I_{d_n, cf(i)} = (l, j_i, h_i)\) for \(i < k\) and \((\nu_k, \ell_k) = \theta_{n, cf(k)}\), \(I_{d_n, cf(k)} = (l, j_k, h_k)\); moreover, \(\sigma_n = \sigma = \sigma\). Define \(m\) as the (largest) index such that \(a_m = -1\); likewise, define \(m'\) such that \(a_{m'} = -1\).

Choose any \(\overline{L}_{C_{m,i}}\) from \(\overline{C_m}\) and apply Observation 3-12 to the second computation in (1): \(\lambda = \lambda_0 \cdot (l, j, p, \bar{r}_{m,i}, \bar{j}_m, \bar{s}) \cdot \lambda_1\) and \(\lambda_1\) does not witness any other activity in the corresponding procedure instance. Now, apply the Observation (in the reverse direction) to the first computation in (1) and conclude that \(\overline{\theta}_{m,i} = \theta_{m,j}\) for some \(j\). Repeating this argument gives that every local state in \(\overline{C_m}\) is also reached in \(C_m\). If \(m = n\), this concludes the proof.

Next, let \(m < n\). the \(C_{m-1} \rightarrow C_m\) and \(\overline{C}_{m-1} \rightarrow \overline{C}_m\)-transitions are both witnessed by the same last (local) record in \(\lambda_1\), so that \(\overline{\sigma}_m = \overline{\sigma}_m\).

Finally, the \(C_0 \rightarrow \overrightarrow{\lambda} \overline{C}_m\)-computation must be extended. There are two separate cases.

1) \(a_{m+1} \notin \text{Ran}(c_f)\). This means that \((\nu_i, \ell_i) = \theta_{m, cf(i)}\) for \(i < k\). If \(\overline{\theta}_{m, a_{m+1}}\) exists, define \(I\) as the index of the corresponding local state in \(C_m\); otherwise, let \(I\) be 1 larger than the number of LC's in \(C_m\). Define configurations \(C_{m+i}\) for \(1 \leq i \leq n - m\) by
- \(LC_{m+i,j} = LC_{m,j}\) for \(j \neq I\),
- \(LC_{m+i,j} = LC_{m+i, \overline{a}_{m+1}}\),
- \(\sigma_{m+i} = \overline{\sigma}_{m+i}\), and
- \(a_{m+i} = I\).

As \(K\) is LC-independent, I have \(C_0 \overset{\lambda}{\rightarrow} \overrightarrow{C_{m+n}} \subseteq \text{Comp}(M_1, \{\tau\}, K)\). Hence, \(\theta_{m+n-a_{m}, cf(i)} = (\nu_i, \ell_i)\) for \(i < k\). Moreover, by construction, \(\theta_{m+n-a_{m}} = (\nu_k, \ell_k)\) and \(\sigma_{m+n} = \sigma\). The Lemma follows.

2) \(\overline{a}_{m+1} \neq cf(k)\). Hence, \(\overline{\theta}_{m, cf(k)} = (\nu_k, \ell_k)\). The first computation in (1) proves the Lemma.
There seems to be a third case: \( a_{m+1} \in \text{Ran}(c_f) \) and \( a_{m+1} = c_f(k) \); say \( a_{m+1} = c_f(0) \). This, however, cannot happen because either \( \ell_0 \) or \( \ell_k \) labels a waiting point, but \( \lambda \) does not contain a local record.

4.2 The global merging lemma, GML

4-4 Definition (projection). Let \( \lambda \) be some record sequence of a program \([M_1] \parallel \cdots \parallel M_n]\).

- \( \lambda \upharpoonright l \) is the subsequence of \( \lambda \) obtained by erasing all records that do not involve module \( M_1 \).
- \( \lambda \upharpoonright_{\downarrow g} l \) is the subsequence of \( \lambda \upharpoonright l \) obtained by erasing all local records.

4-5 Definition (global reachability, \( GR \)). Let \( P \) be a \( CM^* \Sigma \)-program, \([M_1] \parallel \cdots \parallel M_n]\); \( \tau \), \( \nu_0, \ldots, \nu_k \in \omega \) and \( \ell_0, \ldots, \ell_k \in \text{Lab} \) such that \( \ell_i \in \text{FLB}(M(i)) \) for \( i \leq k \).

Then \((h_0, \ell_0, \ldots, h_k, \ell_k) \) is \( (P, \tau, \nu_0, \ldots, \nu_k, \sigma, \lambda) \)-\( GR \) iff

\[
\exists C_0 \xrightarrow{1} \cdots \xrightarrow{1} C_\infty \in \text{Comp}(P, \tau) \text{ such that }
\]

- \( \lambda \upharpoonright l = \lambda \upharpoonright i \) for every \( i \in \text{Ran}(l) \)
- there is a function \( c_f: \{0 \ldots k\} \rightarrow \omega \) such that for all \( i = 1 \ldots k \), \( \sigma \upharpoonright V_i(i) = \sigma_{n(i)} \upharpoonright V_i(i) \),
  \( \theta_{n, c_f(i)}(\nu_i, \ell_i) \) and \( Id_{n, c_f(i)}^i \equiv (i(i), \tau, h_i) \) with \( \ell_i \in \text{FLB}(\sigma_{\ell_i}^i) \).

4-6 Lemma (Global Merging Lemma, GML). With notation as above, suppose

- \((h_0, \ell_0, \ldots, h_k, \ell_k) \) is \( (P, \tau', \nu_0, \ldots, \nu_k, \sigma, \lambda') \)-\( GR \),
- \((h_k, \ell_k) \) is \( (P, \tau, \nu_k, \sigma, \lambda') \)-\( GR \),
- \((\ldots) \) is \( (P, \tau, \sigma, \lambda) \)-\( GR \), and
- \( \tau \upharpoonright V = \tau_j \upharpoonright V \) for \( i, j = 0 \ldots 2 \) and \( V = \bigcup_{i=0}^{n} V_i, \lambda \upharpoonright_{\downarrow g} l(i) = \lambda' \upharpoonright_{\downarrow g} l(i) \) for \( i < k \) and \( \lambda \upharpoonright_{\downarrow g} l(k) = \lambda'' \upharpoonright_{\downarrow g} l(k) \).

Then \((h_0, \ell_0, \ldots, h_k, \ell_k) \) is \( (P, \tau, \nu_0, \ldots, \nu_k, \sigma, \lambda) \)-\( GR \), where \( \tau \upharpoonright V = \tau_0 \upharpoonright V, \lambda \upharpoonright l(i) = \lambda' \upharpoonright l(i) \) for \( i < k \) and \( \lambda \upharpoonright l(k) = \lambda'' \upharpoonright l(k) \).

Proof. W.l.o.g., assume that \( \tau_0 = \tau_1 = \tau_2 = \tau \). The Lemma is proven with induction on the number of global records in \( \lambda \). A stronger version of the Lemma is, however, needed to carry this induction:

Suppose

1. \( \hat{C}_0 \xrightarrow{\lambda} \cdots \xrightarrow{\lambda} \hat{C}_n \in \text{Comp}(P, \tau) \),
2. \( \hat{C}_0 \xrightarrow{\lambda} \cdots \xrightarrow{\lambda} \hat{C}_n \in \text{Comp}(P, \tau) \),
3. \( \hat{C}_0 \xrightarrow{\lambda} \cdots \xrightarrow{\lambda} \hat{C}_n \in \text{Comp}(P, \tau) \) and
4. \( A_0 \cup A_1 \subseteq \{1 \ldots n\} \) with \( A_0 \cap A_1 = \emptyset \) and \( \lambda \upharpoonright_{\downarrow g} i = \hat{\lambda} \upharpoonright_{\downarrow g} i \) if \( i \in A_0 \), \( \hat{\lambda} \upharpoonright_{\downarrow g} i = \hat{\lambda} \upharpoonright_{\downarrow g} i \) if \( i \in A_1 \).
Then there is a computation \( C_0 \xrightarrow{\lambda} \ast C_n \in \text{Comp}(P, \tau) \), with \( \lambda \mid i = \lambda \mid i \) for \( i \in A_0 \),

\[
\lambda \mid i = \lambda \mid i \text{ for } i \in A_1, \lambda \mid i = \lambda \mid i \text{ for } i \notin A_0 \cup A_1 \text{ and } C_n^i = \begin{cases} C_n^i & \text{if } i \in A_0 \\ C_n^i & \text{if } i \in A_1 \\ C_n^i & \text{otherwise} \end{cases}
\]

Note that the reformulation is symmetrical with respect to the partition of \( \{1 \ldots n\} \).

If \( \lambda \) does not contain any global records (hence, neither do \( \tilde{\lambda} \), \( \check{\lambda} \) and \( \lambda \)) the claim obviously holds, because the module computations all are independent.

Next, split \( \lambda \) as \( \lambda_0 \cdot \lambda_1 \), where \( \lambda_1 \) is the smallest suffix of \( \lambda \) containing one global record, \( (p, \varrho, \delta, \tau, s, \text{ind}) \). By symmetry, there are only two cases: (1) \( p \) and \( \tau \) are in different partitions and (2) \( p \) and \( \tau \) are in the same partition. The strengthened form of the Lemma is needed in case \( \{p, \tau\} \not\subseteq A_0 \cup A_1 \).

Case 1) W.l.o.g., let \( p \in A_0 \) and \( \tau \in A_1 \). The splitting of \( \lambda \) induces splittings of \( \lambda \) and \( \tilde{\lambda} \) into \( \lambda_0 \cdot \lambda_1 \) and \( \lambda_0 \cdot \check{\lambda}_1 \) such that for \( i = 0, 1 \): \( \lambda_i \mid_{p, j} = \lambda_i \mid_{p, j} \) if \( j \in A_0 \) and \( \check{\lambda}_i \mid_{p, j} = \check{\lambda}_i \mid_{p, j} \) if \( j \in A_1 \) and \( \check{\lambda}_1 \) and \( \lambda_1 \) are as small as possible. Define \( \tilde{m} \) to be the largest index such that \( \tilde{C}_0 \xrightarrow{\lambda_0} \ast \tilde{C}_m \xrightarrow{\lambda_1} \ast \tilde{C}_n \); likewise, define \( \check{m} \) and \( m \) w.r.t. the latter two computations. Induction gives a computation \( C_0 \xrightarrow{\lambda_0} \ast C_m \xrightarrow{\lambda_1} \ast C_n \in \text{Comp}(P, \tau) \) such that \( \lambda_0 \mid i = \lambda \mid i \) for \( i \in A_0 \), \( \lambda_0 \mid i = \tilde{\lambda}_0 \mid i \) for \( i \in A_1 \), \( \lambda_0 \mid i = \check{\lambda}_0 \mid i \) if \( i \notin A_0 \cup A_1 \), \( C_m^i = \tilde{C}_m^i \) if \( i \in A_0 \), \( C_m^i = \check{C}_m^i \) if \( i \in A_1 \) and \( C_m^i = C_m^i \) otherwise. By definition of \( C_m \) (i.e., of \( \tilde{m} \) and \( m \)) a transition to \( C_{m+1} \) is possible producing the global record in \( \lambda_1 \), where \( C_{m+1}^i = C_m^i \) for \( i \notin \{p, \tau\} \), \( C_{m+1}^p = \tilde{C}_{m+1}^p \) and \( C_{m+1}^\tau = \check{C}_{m+1}^\tau \). The remainder of the three computations (if any) are independent and hence can be grafted onto \( C_{m+1} \): first all the steps involving modules (whose indices are) in \( A_0 \) from computation 1; then the steps involving modules in \( A_1 \) from the second one; and finally the steps in the remaining modules from the third computation. This results in a computation \( C_0 \xrightarrow{\lambda_0} \ast C_m \xrightarrow{\lambda_1} \ast C_n \in \text{Comp}(P, \tau) \).

By construction, \( C_n \) and \( \lambda_1 \) (hence \( \lambda_0 \cdot \lambda_1 \)) satisfy the conditions of the Lemma.

Case 2) W.l.o.g., let \( p, \tau \in A_0 \). As in case 1, the label sequences \( \lambda \) and \( \check{\lambda} \) can be split, which also determines \( \tilde{m} \) and \( m \). Since the second computation is not involved in the last communication, I can take \( \lambda_0 = \lambda, \lambda_1 = \varepsilon \) and \( \check{m} = \tilde{m} \). Note that \( \lambda_0 \mid_{p, i} = \lambda \mid_{p, i} \) for \( i \in A_1 \). The rest of the argument is as in the first case.

The strengthened form of the Lemma together with its proof show the essential triviality of the Global Merging Lemma: a transition label sequence really determines the complete computation. An inspection of the proof also suggests that if \( A_0 \) and \( A_1 \) are singleton sets (or empty) it suffices to assume local instead of global reachability for the \( A_0 \) and \( A_1 \) computations. In fact, the following special case is worth recording:

**4-7 Lemma.** With notation as above, assume that

- \((h, \ell)\) is \((M, \tau, K, \nu, \sigma, \lambda)\)-ir,
- \((\cdot)\) is \((P, \tau, \sigma, \lambda)\)-gr and
- \(\tau \mid V_1 \equiv \tau \mid V_1 \) and \(\lambda \mid_{p, i} = \lambda \).

Then \((h, \ell)\) is \((P, \tau, \nu, \sigma, \lambda)\)-gr.

**Proof.** The proof is in fact very easy and does not require an induction. Just take the global computation, \( C_0 \xrightarrow{\lambda} \ast C_n \in \text{Comp}(P, \tau) \) and remove all transitions involving \( M_1 \). In their place,
substitute the transitions from $C_0 \xrightarrow{\lambda} C_m \in \text{Comp}(M_1, \tau, K)$. Because in both computations $M_1$ starts in the same state and because $\lambda \perp q I = \lambda$, the resulting computation is in $\text{Comp}(P, \tau)$ and shows the required global reachability. □

4-8 Observation. Local and global reachability are semi computable relations.

5 Specifications and Proofrules

5.1 Specifications and their interpretation

5.1 Definition (Specifications). Let $T$ be a $CM^\ast(\Sigma)$-statement or procedure, $M$ a module and $P$ a program; $p, q, MI \in L(\Sigma)$. Specifications are of one of the following three forms:

- $(1) (CA, WA, CO, MI \mid \{p\}_T\{q\})$ or $(2) (CA, \emptyset, CO, MI \mid \{p\}_M\{q\})$, where ($T$ stands for $T$ or $M$)
  - $CA \subseteq \{ \ell : p \mid \ell \text{ labels a closing bracket in } T, \ p \in L(\Sigma) \}$
  - $WA \subseteq \{ \ell : p \mid \ell \text{ labels the rear of a wait in } T \text{ or } T \equiv T'.\ell, \ p \in L(\Sigma) \}$
  - $CO \subseteq \{ \ell : p \mid \ell \text{ labels an opening bracket, a call or the front of a wait in } T, \ p \in L(\Sigma) \}$

- in case (2),
  - $FV(MI) \cap L_i = \emptyset$,
  - for all $\ell : p$-term: $FV(p) \cap L(\ell) \subseteq L(\ell)$ and
  - any $ic_i \in \text{Var}^{IC}$ that appears in any assertion refers to a procedure in $M_i$; i.e., has the form $ic.a_i$.

- if $T$ is a procedure with formal parameters, $u$, then $FV(p, q) \cap \{u\} = \emptyset$

- $(3) (\emptyset, \emptyset, \emptyset, \text{true} \mid \{p\}_P\{q\})$

5.2 Definition. Let $A \subseteq \{ \ell : p \mid \ell \in \text{Lab}, \ p \in L(\Sigma) \}$

- $A(\ell) = \bigwedge\{p \mid \ell : p \in A\} \in L(\Sigma)$. (Note that $\bigwedge\emptyset \equiv \text{true}$).

- $A$ is disjoint if $\neg \exists \ell \in \text{Lab}, p, q \in L(\Sigma)$ such that $\{\ell : p, \ell : q\} \subseteq A$ and $p \neq q$.

Intuitively, a formula like $\ell : p$ is true in configuration-state either if control is not at the $\ell$-labeled point, or $p$ is satisfied in the (local + global) state at such a point. The meaning of a specification for $T$ is, that on any execution sequence of $T$ that starts in a state satisfying $p$ and for which $CA$ and $WA$ are true in every configuration-state, $CO$ will be true in the last configuration state of the sequence, $MI$ will hold if it is a waiting point and $q$ will hold if $T$ terminated. So, e.g. $(\emptyset, \emptyset, \emptyset, \text{true} \mid \{p\}_S\{q\})$ is an ordinary partial correctness specification and (hence) clause 3 expresses the paper’s basic interest in partial correctness properties.

This notion of specification is essential for syntax-directedness. The communication-assumption, $CA$, waiting-assumption, $WA$, and commitment, $CO$, make explicit (1) the various assumptions about the environment’s behaviour upon which validity of the specification (i.e. $CO$ and $q$) depends and (2) the behaviour to which the statement commits, necessary to validate the environment’s assumptions. Finally, there is a module-invariant, $MI$, that characterizes the (global) module state at waiting points; hence the states in which procedures start executing.
5–3 DEFINITION (satisfaction and validity). Let $\Sigma$ be a signature and $A$ a $\Sigma$-structure; $T$ a $CM^*(\Sigma)$-statement, procedure, module or program, $p \in L(\Sigma)$, $A \subseteq \{t : p \mid t \in Lab, p \in L(\Sigma)\}$ and $\delta$ a configuration-state $\{\left(\sigma_i, \theta_{i,0}, \theta_{i,1}, \ldots, \theta_{i,n}, \sigma_i\right) \mid i = 1..m\}$.

Let $\bar{\sigma} = \bigcup_{i=1}^n \sigma_i$ and $\bar{\nu} = \bigcup \left\{ \nu_{i,a} \mid a_i \neq -1, i = 1..m \right\}$.

- $A, \delta \models p$ if and only if $A, \bar{\sigma} \cup \bar{\nu} \models p$
- $A, \delta \models A$ if $a_i \neq -1 \Rightarrow A, \sigma_i \cup \nu_i \models \rho$ for $j = 0..n_i$, $i = 1..m$
- $A, \delta \models^* A$ if $A, \sigma_i \cup \nu_i \models \rho$ for $j = 0..n_i$, $i = 1..m$
- Satisfaction can be trivially extended to conjunctions of first order formulae and formulae of the form $A$. Validity is defined in the standard way.

Let $\bar{\rho} = \{\sigma \mid A, \sigma \models p\}$ and $\bar{A} = \{C \mid A, s(C) \models A\}$,

- $A, (\delta_0 \ldots \delta_n, t) \models (CA, WA, CO, MI \mid \{p\}T\{q\})$ iff
  \[
  (\delta_0 \ldots \delta_n, t) \in [T](\bar{\rho}, CA \cup WA) \Rightarrow A, \delta_n \models CO \& \\{p\}
  \]
  \[
  a_n = -1 \Rightarrow A, \delta_n \models MI \& \\{p\}
  \]
  \[
  t = T \Rightarrow A, \delta_n \models q
  \]

- $A \models (CA, WA, CO, MI \mid \{p\}T\{q\})$ iff
  \[
  (\delta_0 \ldots \delta_n, t) \in [T](\bar{\rho}, CA \cup WA) \text{ and } A, (\delta_0 \ldots \delta_n, t) \models (CA, WA, CO, MI \mid \{p\}T\{q\})
  \]

- $A \models (CA, WA, CO, MI \mid \{p\}T\{q\})$ iff
  \[
  \forall (\delta_0 \ldots \delta_n, t) \in [T] \ A, (\delta_0 \ldots \delta_n, t) \equiv (CA, WA, CO, MI \mid \{p\}T\{q\})
  \]

The definition itself is straightforward. Note that local states can only be specified by "labeled" formulae and that satisfaction of non-labeled formulae at waiting points does not involve the local state. The meaning of satisfaction of a labeled formula interacts with the definition of configuration state: remember that in a (module) configuration state, $\delta = (a, \theta_0, \ldots, \theta_m, \sigma)$, the label part of a local state, $\theta_i$, is included only if $a = i$ or $a = -1$. In other words

\[
A, \delta \models A \iff a \neq -1 \Rightarrow \sigma \cup \nu_a \models A(\rho_0)
\]

Hence, if $a \neq -1$ then the relations $\models$ and $\models^*$ coincide, but if $a = -1$ then $A, \delta \models A$ is vacuously true, whereas for $\models^* 1$ have that

\[
a = -1 \Rightarrow A, \delta \models^* A \iff \sigma \cup \nu_i \models A(\rho_i) \text{ for } i = 1..m
\]

Note that if $a = -1$ then $\rho_i \neq \emptyset$ for $i = 1..m$.

The definition of satisfaction can be turned around — as usual — and be used to characterize sets of states, respectively, configuration states by formulae, respectively, sets of labeled formulae. In the sequel, such correspondences will be implicitly understood and I shall write, e.g., $[S](p, CA)$.

\[\text{Remember the assumption that states always have domains that contain all the appropriate variables.}\]
5–4 Observation. Satisfaction is a computable relation. Non-validity is a semi-computable relation.

5.2 The proof system, \( P_S(T) \)

Let \( T \subseteq L(\Sigma) \) for some signature \( \Sigma \). In the rules below, some of the premises are \( L(\Sigma) \)-formulae. The interpretation of such a rule — as usual — is that it can be applied provided every \( L(\Sigma) \)-formula among its premises is in \( T \).

The proof system consists of the following rules and axioms.

5.2(1.) Assignment axiom.

\[
\langle 0, 0, 0, \text{true} \mid \{ p[e/z] \} z := e(p) \rangle
\]

5.2(2.) Wait axiom.

\[
\frac{p \rightarrow q \land MI}{\langle 0, \{ \ell' : r \}, \{ \ell : q \}, MI \mid \{ p \} \ell \text{.wait.} \ell' \{ r \} \rangle}
\]

5.2(3.) Call rule.

\[
\begin{align*}
\langle 0, 0, 0, \text{true} \mid \{ p \} S_1 \{ y \neq q \} \\ \{ p, \ell : q \} \rangle & , \langle 0, 0, 0, \text{true} \mid \{ q \} S_2 \{ z \} \rangle \\
\langle \{ \ell'' : r \}, 0, \{ \ell : p, \ell' : q \}, \text{true} \mid \{ p \} \ell . (S_1 ; \ell'. \text{call} M, a_1 (\ell\#z) ; S_2 ; \ell'' .) \{ r \land s \} \rangle
\end{align*}
\]

5.2(4.) Sequential composition rule.

\[
\begin{align*}
\langle CA_1, WA_1, CO_1, MI_1 \mid \{ p_i \} S_1 \{ q_i \} \rangle & , \langle p_1 \rightarrow p_2 \rangle \\
\langle CA_1 \cup CA_2, WA_1 \cup WA_2, CO_1 \cup CO_2, MI_1 \lor MI_2 \mid \{ p_1 \} S_1 ; S_2 \{ q_2 \} \rangle & ,
\end{align*}
\]

provided that \( CA_1 \cup CA_2, WA_1 \cup WA_2 \) and \( CO_1 \cup CO_2 \) are disjoint.

5.2(5.) If rule.

\[
\begin{align*}
\langle CA_1, WA_1, CO_1, MI_1 \mid \{ p \land b \} S_1 \{ q \} \rangle & , \langle CA_2, WA_2, CO_2, MI_2 \mid \{ p \land \neg b \} S_2 \{ q \} \rangle \\
\langle CA_1 \cup CA_2, WA_1 \cup WA_2, CO_1 \cup CO_2, MI_1 \lor MI_2 \mid \{ p \} \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi} \{ q \} \rangle & ,
\end{align*}
\]

provided that \( CA_1 \cup CA_2, WA_1 \cup WA_2 \) and \( CO_1 \cup CO_2 \) are disjoint.

5.2(6.) While rule.

\[
\begin{align*}
\langle CA, WA, CO, MI \mid \{ p \land b \} S \{ p \} \rangle & , \\
\langle CA, WA, CO, MI \mid \{ p \} \text{while } b \text{ do } S \text{ od} \{ p \land \neg b \} \rangle & ,
\end{align*}
\]

5.2(7.) Initial statement rule.

\[
\begin{align*}
\langle CA, WA, CO, MI \mid \{ p \} S \{ MI \} \rangle & , \\
\langle CA, WA \cup \{ \ell : q \}, CO, MI \mid \{ p \} \mid S . \ell \{ q \} \rangle & ,
\end{align*}
\]

provided \( WA \cup \{ \ell : q \} \) is disjoint.

5.2(8.) Procedure rule.

\[
\begin{align*}
\langle 0, 0, 0, \text{true} \mid \{ p \} S_1 \{ q \} \rangle & , \langle CA, WA, CO, MI \mid \{ q \land r \} S \{ z \} \rangle \\
\langle 0, 0, 0, \text{true} \mid \{ s \} S_2 \{ t \} \rangle & , \langle CA \cup \{ \ell : r \}, WA, CO \cup \{ \ell : s \}, MI \mid \{ p \} a_1 (\ell\#\theta) \mid S_1 ; \ell . ; S_2 \{ t \} \rangle & ,
\end{align*}
\]

provided \( FV(p, t) \cap \{ \tilde{u}, \tilde{v} \} = \emptyset \) and \( CA \cup \{ \ell : r \} \) and \( CO \cup \{ \ell : s \} \) are disjoint.
5.2(9.) module rule.

Let the module \( M_i \) be declared as

\[
M_i ::= a_1(\tilde{u}_1 \# \tilde{v}_1) : [\cdot : S_1] \\
\cdots \\
a_k(\tilde{u}_k \# \tilde{v}_k) : [\cdot : S_k] \\
: [S_0 : \ell_0]
\]

Then

\[
\begin{align*}
&\langle CA_j, WA_j, CO_j, MI \mid \{MI\}a_j'(\tilde{u}_j \# \tilde{v}_j) : [S_j]\{MI\} \rangle, \quad j = 1, \ldots, k \\
&\langle CA_0, WA_0, CO_0, MI \mid \{p \land \text{IN}\}S_0\{MI \land WA_0(\ell_0)\} \rangle, \quad MI \land WA_0(\ell_0) \rightarrow q \\
&\bigcup_{j=0}^k \{CA_j, WA_j, MI, \{\tilde{u}_j, \tilde{v}_j\}, a_j'(\tilde{u}_j \# \tilde{v}_j) : [t_j, S_j] \} \quad j = 1, \ldots, k \\
&\bigcup_{j=1}^k \{CA_j, WA_j, MI, \{\tilde{u}_j, \tilde{v}_j\}, a_j'(\tilde{u}_j \# \tilde{v}_j) : [t_j, S_j] \} \quad j = 1, \ldots, k \\
&\text{for every } \ell \text{wait } \ell' \text{ in } M_i: \bigcup_{i=0}^k CO_i(\ell) \rightarrow MI \land \bigcup_{i=0}^k WA_i(\ell') \\
\end{align*}
\]

provided \( \bigcup_{j=0}^k CA_j, \bigcup_{j=0}^k WA_j \) and \( \bigcup_{j=0}^k CO_j \) are disjoint and where

- \( \text{IN} \equiv \land \{\text{ic.a}_j^i = 0 \mid j = 1 \ldots n_i\} \).

Let \( A = \{\ell: p \mid \ell \in \text{Lab}, p \in L(\Sigma)\} \). Then the expression

\[
A \neq \langle CA, WA, p, \tilde{\varnothing}, S \rangle
\]

stands for the set of interference freedom specifications

\[
\{(CA, WA(q'), CO, \text{true} \mid \{p'\}S(q')) \mid \ell : q \in A\}
\]

where

- \( q' \equiv q[\tilde{y}/\tilde{y}] \) (\( \tilde{y} \) does not appear free in \( CA, WA, p, q \) or \( S \)),
- \( p' = p \land q' \),
- \( WA(\tau) \equiv \{\ell': WA(\ell') \land \tau \land \text{ic.a}_j^i \geq N \mid \ell' \text{ labels the rear of a wait}, \)
  \( \text{in } S; \text{FLB}(a_j^i) = \text{FLB}(S) \text{ for } j > 0 \text{ and } \)
  \( \text{if } \ell \in \text{FLB}(a_j^i) \text{ then } N = 2 \text{ else } N = 1\})
- \( CO \equiv \{\ell: q' \mid \ell \text{ labels the front of a wait}\),

5.2(10.) program rule.

\[
\begin{align*}
&\langle CA_i, \emptyset, CO_i, MI_i \mid \{p_i\}M_i(q_i) \rangle, \quad i = 1, \ldots, n \\
&\text{Coop}\{\langle CA_i, \emptyset, CO_i, MI_i \mid \{p_i\}M_i(q_i) \rangle \mid i = 1, \ldots, n\}, \text{GI} \\
&\langle \emptyset, \emptyset, \emptyset, \text{true} \mid \{p_1 \land \cdots \land p_n \land \text{GI}\}M_1 \cdots M_n(q_1 \land \cdots \land q_n \land \text{GI}) \rangle
\end{align*}
\]

provided

- \( \bigcup_{i=1}^n CA_i, \bigcup_{i=0}^n WA_i \) and \( \bigcup_{i=1}^n CO_i \) are disjoint

20
- \( FV(CA_i, CO_i, MI_i, p_i, q_i) \cap V_j = \emptyset \) for \( i \neq j \)
- no variable free in \( GI \) appears on the left-hand side of any assignment outside a bracketed section in any \( M_i \) nor as a formal parameter of any call or procedure.

The expression

\[
\text{Coop}((CA_i, WA_i, CO_i, MI_i, \{p_i\} M_i, \{q_i\}) \mid i = 1, \ldots, n), GI)
\]

stands for the set of cooperation specification pairs

\[
\langle 0, 0, 0, \text{true} | \{CO_i(\ell) \land MI_j \land GI\} S_i; c_a_k := ic.a_k + 1; S'_i[\ell] \rangle \{CA_j(\ell')[\ell] \land GI\}
\]

and

\[
\langle 0, 0, 0, \text{true} | \forall \tilde{c} \in CO_i(\ell) \land CO_j(\ell)[\ell] \land \forall \tilde{c} \in GI \{S'_i[\ell] \rangle \{S_i[\ell] \rangle \{CO_i(\ell) \land GI\}
\]

\[
\text{where } [\ell] \equiv [\varepsilon, \varepsilon/\varepsilon, \varepsilon]
\]

for any matching call-procedure pair

\[
\ell.(i) S_i; \ell. \text{call } M_j.a_k(\varepsilon/\varepsilon); S_2; \ell) \text{ in } M_i \text{ and }
\]

\[
a_k^{S_2}(\varepsilon, \varepsilon) : [S'_i[\ell]]; S_i; \ell.; S'_2 \text{ in } M_j
\]

5.2(11.) consequence rule.

\[
(CA, WA, CO, MI | \{p\} S(\{q\}), p' \rightarrow p, q \rightarrow q'
\]

\[
\forall \ell \in \text{FLB}(S) : CA'(\ell) \rightarrow CA(\ell), WA'(\ell) \rightarrow WA(\ell), CO(\ell) \rightarrow CO'(\ell)
\]

\[
\text{if } S \text{ is not a statement or contains a wait: } MI \rightarrow MI'
\]

\[
(CA', WA', CO', MI' | \{p\} S(\{q\})
\]

5.2(12.) substitution rule.

\[
(CA, WA, CO, MI | \{p\} S(\{q\})
\]

\[
\text{provided } z \notin FV(CO, MI, q, S).
\]

5.2(13.) auxiliary variable rule.

Let \( AV \) be a set of variables such that \( z \in AV \) \( \Rightarrow \) \( z \) appears in \( S' \) only in assignments, \( y := t \), where \( y \in AV \) or as formal value parameter in a procedure declaration.

\( S \) is obtained from \( S' \) by deleting all assignments to variables in \( AV \) and all occurrences as value parameter in a procedure declaration together with the corresponding actual parameters in the syntactically matching calls.

\[
(CA, WA, CO, MI | \{p\} S(\{q\})
\]

\[
\text{provided } FV(q, CO, MI) \cap AV = \emptyset.
\]

5.2(14.) label (and bracketing) rule.

\[
(0, 0, 0, \text{true} | \{p\} S(\{q\})
\]

\[
(0, 0, 0, \text{true} | \{p\} S(\{q\})
\]

provided \( S \) is obtained from \( S' \) by deleting every bracket or label.

Concluding this section are the two theorems proof of which justify the paper.

\[\text{Note that if } S \text{ is a statement but does not contain a wait, so that no waiting point is reached while executing } S, \text{ one may take an arbitrary } MI'.\]
5–5 Theorem (Soundness). Let $A$ be a $\Sigma$-structure and $T \subseteq \text{Th}(A)$. $P$ is a $CM(\Sigma)$-program. Then

$$\vdash_{PS(T)} \langle 0, 0, 0, \text{true} \mid \{p\}P\{q\} \rangle \Rightarrow A \models \langle 0, 0, 0, \text{true} \mid \{p\}P\{q\} \rangle$$

5–6 Theorem (Completeness). Let $\Sigma$ be the signature of Peano-arithmetic and $N$ the standard model. Then

$$N \models \langle 0, 0, 0, \text{true} \mid \{p\}P\{q\} \rangle \Rightarrow \vdash_{PS(\text{Th}(N))} \langle 0, 0, 0, \text{true} \mid \{p\}P\{q\} \rangle$$

5.3 A substitution lemma

For the soundness of the program rule and for completeness for program specifications, some results on substitutions are needed to relate the value assignments at the start and end of procedure calls with the syntactic substitution in the cooperation specifications.

5–7 Lemma (The substitution lemma). Let $S, S'$ be $CM^*(\Sigma)$-statements not containing call or wait-statements. Let $u, v, z \subseteq \text{Var}$, $\varepsilon \subseteq \text{Term}(\Sigma)$ and $p, q \in L(\Sigma)$ be such that $|u| = |v| = |z|$, the variables in $u$ do not appear as left-hand-side of any assignments in $S$ and the variables in the lists $u, v$ and $z$ are pairwise disjoint. Additionally, assume that the following conditions hold:

$$(u \cap v = \emptyset, \text{FV}(\varepsilon) \cap z = \emptyset, \{u, v\} \cap \text{FV}(q) = \emptyset) \text{ and } (\text{FV}(S') \cup \{u, v\}) \cap (\text{FV}(\varepsilon) \cup z) = \emptyset.$$ Then

1. $A \models \langle p \mid S; S' \mid \{q\} \rangle \iff A \models \langle p \mid S; u, v := \varepsilon, z; S'\{q\} \rangle$
2. $A \models \langle p \mid S' \mid \{q\} \rangle \iff A \models \langle p \land u = \varepsilon \mid S' \mid \{q\} \rangle$, provided $\varepsilon \cap \text{FV}(p) = \emptyset$.

The proof of this Lemma is delegated to the Appendix.

6 Soundness

Validity of rules is defined as usual: validity of the premisses implies validity of the consequence. Fix some $\Sigma$-structure $A$ and $T \subseteq \text{Th}(A)$.

6–1 Lemma. The axioms 1 and 2 and the rules 3, ..., 14 are valid.

Although this paper's specifications are more detailed than partial correctness specifications, the validity proofs for some of the axioms and rules (e.g., sequential composition, if and while rules, consequence and substitution rules, the auxiliary variable rule) are completely analogous to the usual proofs; see e.g., [deBBO] and [AptB3]. For the assignment statement this is obvious, since its execution sequences do not include labels or waiting points. For, e.g., the sequential composition rule it is because assumptions and commitments cannot refer to labels in the component they are not associated with. In the proof below, only the non-trivial and new cases will be tackled.

Proof. [of Lemma 6–1] Generically, for any $S_o^0$ considered below, let $(\delta_0^0, \cdots, \delta_n^0, t) \in [S_o^0]$ where $\delta_0^0 = s(SC_{sv}(S_o^0))$.

The notational conventions of Section 3 will be heavily used. In the proofs, prefixes $(\delta_0^0, \cdots, \delta_i^0, 1)$ of the above sequence will be considered, too. Note that if the initial state is not constrained, any sequence in $[S_o^0]$ can be obtained as a suffix of a sequence as above. Usually, statements will be embedded within other statements; say, $S_0 \equiv S_1; S_2$. In such cases, the following
“aliasing” convention applies: the configuration-state sequence associated with $S_i$ is $\delta^i_0 \cdots \delta^i_n$, for $i = 0, 1, 2$. We have that $\delta^1_0 = \delta^2_0 = \delta^3_1 = \delta_{n+1}^1$, $\delta^3_{n+1} = \delta^2_{n+2} = \delta^0_{n+3} = \delta^1_{n+4}$. I shall mostly ignore the terminal states in completed computation sequences, since it is usually the last but one state that is needed in the arguments below.

6-1(1)  wait axiom: $S \equiv \ell.\text{wait}, WA \equiv \{\ell : r\}$ and $CO \equiv \{\ell : q\}$.

By Definition 3-9, $\delta_0 = (0, (\Omega, \ell), \sigma)$, $\delta_1 = (-1, (\Omega, \ell), \sigma)$, $\delta_2 = (0, (\Omega, \ell'), \sigma')$ and $\delta_3 = (0, (\Omega, 0), \sigma')$. First assume that $\delta_0 \models p$. Then $\delta_0 \models CO$ and $\delta_1 \models MI$ because $p \rightarrow CO(\ell) \land MI$. Note that $\delta_2 \models WA$ is trivially true and that $\delta_2 \models WA$ holds by assumption. Next, $\delta_2 \models WA$ implies $\delta_3 \models r$. Here, I ignored the terminating configuration, $\delta_4 = (0, \sigma')$ as explained above. Obviously, I also have that $\delta_4 \models r$.

6-1(2)  call rule: $S' \equiv \ell'.\text{call } M_i.a_i(\bar{\delta} \# \bar{\varepsilon}), S \equiv \ell.\langle S_1; S_2; \ell'' \rangle$, \( \overline{CA} \equiv \{\ell'' : r\} \)

and $CO \equiv \{\ell : p, \ell' : q\}$.

Again by Definition 3-9, $\delta^i_0 = (0, (\Omega, \ell), \sigma)$, $\delta^i_0 = (0, (\Omega, \ell'), \sigma)$, $\delta^i_1 = (0, (\Omega, \ell), \sigma)$, $\delta^i_2 = (0, (\Omega, 0), \sigma')$ and $\delta^i_3 = (0, (\Omega, \ell), \sigma')$ where $\sigma' = \sigma \{a/\bar{\varepsilon}\}$ and $a \subseteq |A|$.

Now, if $\delta^i_0 \models p$ then $\delta^i_0 \models CO$. Since $S_2$ does not contain any labels, the next point of interest is the front of $S'$. By validity of the first premiss of the call rule, $\delta_{n+1}^0 \models \forall \bar{\varepsilon} \ q$ (where I assume that $S_4$ terminates as there is nothing to prove otherwise), so that also $\delta^i_0 = \forall \bar{\varepsilon} \ q$ and, hence, $\delta^i_0 \models q$. This implies that $\delta^i_0 \models CO$, $\delta^i_1 \models CO$, $\delta^i_2 \models q$ and $\delta^i_3 \models q$ (as $\delta^i_0 = \delta^i_2$). Validity of the second premiss yields $\delta_{n+2}^0 \models s$, hence $\delta_n \models CA(\ell'') \land s$.

6-1(3)  procedure rule: Apply the reasoning of (2) thrice.

6-1(4)  initial statement rule: This is a direct consequence of the fact that every computation sequence of $\langle S.\ell \rangle$ can be obtained from one of $S$, except for the sequence “starting at $\ell$”. This case is covered by the change of $WA$.

6-1(5)  module rule: As above, assume that the premisses of the rule hold. The following stronger statement will be proved:

For any $C_0 \overset{\lambda_1}{\rightarrow} \cdots \overset{\lambda_n}{\rightarrow} C_n \in \text{Comp}(M_i, p) \bigcup_{i=0}^k \overline{CA}$ with $\delta_n = s(C_n)$:

$$\delta_n \models \bigcup_{i=0}^k (CO_i \cup WA_i), \quad a_n = -1 \Rightarrow \delta_n \models MI \quad \text{and} \quad C_n \in TC \Rightarrow \delta_n \models q.$$  

Note that $s(C_0) \models IN$ by Definition 3-7 and that $\delta_n \models CO_i \iff \delta_n \models CO_i$. Hence, I can concentrate on $WA_i$. Next, observe that $MI \land WA_0(\ell_0) \rightarrow q$. This means that $C_n \in TC \Rightarrow \delta_n \models q$ automatically follows from the first two claims and the fact that the $k+1$st premiss of the rule establishes $WA_0(\ell_0) \land MI$ as post condition.

The proof uses induction on the number, $L$, of $\delta_i$'s with $a_i = -1$ for $i < n$. The computations in $a_i^j$ will be superscripted with $j$; those in the initial statement with 0. Define $\delta_i = s(C_i)$ for $i = 1..n$ and let $t = T$ if $C_n \in TC$ and $t = \bot$ otherwise.

$L = 0$ So, the computation is in fact within the initial statement. Validity of the initial statement specification implies that $\delta_n \models CO_0$ and $\delta_n \models MI$ in case $a_n = -1$, since by assumption $\delta_0 \models p \land IN$ and $\delta_i \models CA_0$ for $0 \leq i \leq n$. Observe that $\delta_n \models (\bigcup_{i=1}^n (WA_i \cup CO_i)$ trivially holds in this case and that $\delta_n \models WA_0$ either holds trivially (in case $a_n \neq -1$) or follows from the last premiss of the module rule.

$L > 0$ Let $n$ be the largest index $< n$ such that $a_n = -1$. Let $k = a_{n-1}$ and $(l, h) = Id_{a_{n-1}+1}$.
and let \( k = a_{n+1} \) and \((l, h) = Id_{n+1,k}\). The situation is depicted in Figure 1 (when \( a_n \neq -1 \)).

Extract the \( a_h \)-computation leading to the \( \hat{k} \)th LC in \( C_n \) from this sequence:

First, inductively define a sequence \( a_0, a_1, \ldots, a_n \) that traces the sequence of LC's that evolve into \( LC_{n,k} \):

- \( a_n = \hat{k} \)
- \( 0 \leq i < n - 1: a_i = -1 \) if \( a_{i+1} = -1 \) or \( C_i \) has less than \( a_{i+1} \) LC's,
- \( = a_{i+1} + 1 \) if \( a_i \neq -1, a_{i+1} = -1, a_i < a_{i+1} \) and \( C_i \) has more LC's than \( C_{i+1} \),
- \( = a_{i+1} \) otherwise

This definition sets \( a_i = -1 \) if the correct instance of \( a_h \) still is to be created and sets \( a_i = a_{i+1} + 1 \) if some instance terminates.

Next, define a sequence \( i_0, i_1, \ldots, i_r \) inductively as follows:\footnote{\( r \) is implicitly determined by the condition that \( i_r = n \)}:

- \( i_0 \) is the index \( t \) such that \( a_0 = a_1 = \cdots = a_{t-1} = -1 \) and \( a_t \neq -1 \),
- \( 0 < j \leq r: i_j = i_{j-1} + 1 \) if \( a_{i_{j-1}} \neq -1 \) and \( a_{i_{j-1} + 1} = -1 \),
- \( = \) the smallest index \( t > i_{j-1} \) such that \( a_t = a_i \) otherwise

This sequence points to those MC's, \( C_i \), in which activity is (the second case) or just ceased to be (the first case) within the LC pointed to by \( a_i \).

The \( a_h \)-computation can be extracted in the same way, resulting in sequences \( \beta_0, \beta_1, \ldots, \beta_n \) and \( j_0, \ldots, j_r \). Note that here the sequence ends in \( C_n \).

First I prove that

\[
A: \delta_n \models^* WA_h \cup CO_h, \quad a_n = -1 \iff \delta_n \models MI.
\]

By Definition 5-3, \( \delta_n \) can be replaced by \( s(LC_{n,k}) \).

Now, define \( \check{s} \) by \( \check{s} = n - 1 \) if \( t = \hat{T} \) and by \( \check{s} = n \) otherwise. Let \( \check{s} \) be the index such that \( j_{\check{s}} = n+1 \) (note that \( \check{s} \) exists) and define a sequence of configuration states, \( \delta_{\check{s}}^h \), for \( \hat{s} \leq i < \check{s} \) by:

\[
(\delta_{\check{s}}^h, \ldots, \delta_{\hat{s}}^h, t) = (CA_0, WA_0, CO_0, MI \cup \{ p \land IN \}) S_0[MI \land WA_0(t_0)]
\]
and, if $\eta \neq 0$

$$(\delta^k_0, \ldots, \delta^k_t, t) \models (CA_h, WA_h, CO_h, MI \mid \{MI\}d_h(u \neq v) : [S_h]\{MI\})$$

(3)

Suppose that $\delta^k_0 \models WA_h$ and $\delta^k_1 \models MI$ in case $j_0 = n + 1$; i.e., in case the instance of $a^k_1$ was created in the $C_h \rightarrow C_{h+1}$-transition. Then, $A$ would follow from satisfaction of (2) and (3) and the fact that the only resumption at a waiting point in the computation is "at" $\delta^k_1 = \epsilon(LC_{n+1,k})$.

Hence, there remains to show that

1. $\delta^k_0 \models WA_h$ if $j_0 \neq n + 1$,
2. $\delta^k_1 \models MI$ if $j_0 = n + 1$.

Two cases have to be distinguished.

(I) $j_0 > i_*$ or $k = k$. Here the easy cases are collected where no interleaving takes place: If $j_0 > i_*$ this means that the instance pointed to by the $j$-sequence is created (at the moment) after the other instance terminates. If $k = k$ and $j_0 < i_*$ this means that control remains in the same instance when passing the waiting point, so that $j_0 = i_*$ for $k \leq r$. Let $\tilde{r}$ be the largest index such that $\tilde{e}_{i-1} < \tilde{n}$ and $\alpha_{\tilde{n}-1} = \tilde{n} - 1$ or 1 if such an $\tilde{r}$ does not exist. Define configuration states, $\delta^k_0$ by $\delta^k_0 = (p_{i_1}, \theta_{i_1}, \alpha_{i_1}, \sigma_{i_1})$ where $p_{i_1} = -1$ if $\alpha_{i_1} = -1$ and $p_{i_1} = 0$ otherwise, for $\tilde{r} \leq i \leq r$. Let $\tilde{r}$ be the largest index such that $\tilde{e}_{i-1} < \tilde{n}$ and $\alpha_{\tilde{n}-1} = \tilde{n} - 1$ or 1 if such an $\tilde{r}$ does not exist. Define configuration states, $\delta^k_1$ by $\delta^k_1 = (p_{i_1}, \theta_{i_1}, \alpha_{i_1}, \sigma_{i_1})$ where $p_{i_1} = -1$ if $\alpha_{i_1} = -1$ and $p_{i_1} = 0$ otherwise, for $\tilde{r} \leq i \leq r$.

(II) $j_0 \leq i_*$ and $k \neq k$. I.e., control passed a waiting point and switched to another already existing (!) $LC$. Hence, a wait was encountered or the end of the initial statement or the end of a procedure instance.

To show: $\delta^k_{i+1} \models WA_h$. For this, validity of the interference freedom specifications is needed.

So, let $\rho_{\alpha+1,k} = t$ (note that the $LC$ is $k$ and not $\tilde{k}$). Let $f$ be the cardinality of $Dom(\nu_{\alpha+1,k}) = \{x_1, \ldots, x_f\}$ and let $\{y_1, \ldots, y_{f}\}$ be a collection of variables not part of the domain of any state appearing in any $C_i$ for $0 \leq i \leq n$ and not appearing free in $\bigcup_{t=0}^{k} CA_t$, $\bigcup_{t=0}^{k} WA_t$, $\bigcup_{t=0}^{k} CO_t$, $MI$, $p$, $q$ or $M_I$. Observe that it suffices to show that

$$(0, \theta_{\alpha+1,k}, \sigma_{\alpha+1,k}(\nu_{\alpha+1,k}(x_i)/y_i \ i = 1..f)) \models q,$$
where \( q = WA_h(\ell)[y_i/x_i, i = 1..f] \).

Again, let \( \hat{r} \) be the largest index such that \( i_\hat{r} < n \) and \( a_\hat{r} = -1 \) or 1 if this is impossible. Define \( \nu = \Omega_{i_{n+1,1}(x_i)/y_i, i = 1..f} \) and define a sequence of configuration states by \( \delta^F_j = (p_j, \theta_j, a_j, c_{ij} \cup \nu) \) where \( p_j = -1 \) if \( a_j = -1 \) and \( p_j = 0 \) otherwise, for \( \hat{r} \leq j \leq r \). Then

\[
(\delta^F_j \ldots \delta^F_1, \hat{r}) \models^* (CA^\nu, WA_h(q'), CO, true | \{p' \land q'\}S_h(q'))
\]

for some \( \hat{r} \), where \( q' = WA_h(\ell)[y_i/x_i, i = 1..f] \); \( p' \equiv false \) if \( \hat{h} = 0 \) and \( p' \equiv MI \) otherwise. Observe that if \( \hat{h} = 0 \) then \( \hat{r} \neq 1 \). Induction gives that \( \delta^F_j \models^* MI \land \bigcup_{i=0}^n WA_i \); hence \( \delta^F_0 \models MI \land WA_h \land q' \).

By definition \( \delta^F_j \models CA_h (\hat{r} \leq j \leq r) \), so that I obtain

\[
(\delta^F_j \ldots \delta^F_1, \hat{r}) \in \{[S_h]\}(p' \land q', CA_h \cup WA_h(q'))
\]

Now, (7) gives \( \delta^F_0 \models CO \) and \( \delta^F_0 \models q' \) if \( \hat{r} = T \). Hence, \( \delta_{n+1} \models WA_h \).

B: \( \delta_n \models^* \bigcup_{i=1}^n WA_h \)

If \( a_n \neq -1 \) and \( a_{n-1} \neq -1 \) this holds trivially. If this is not the case, induction gives \( \delta_n \models^* \bigcup_{i=1}^n WA_i \) and every \( WA_i \) (\( i \neq h \)) must be shown interference free over the \( a_{i,h} \)-computation in \( C_n \longrightarrow C_n \).

This is fully analogous to the argument in case A–II and, hence, is omitted.

6–1 (6) Program rule:

6–2 Definition. Acceptable configuration.

- A local configuration, \( Id : (S, \theta) \), is acceptable if either
  - \( S \equiv \ell; \text{rend}_{x,ij}; S' \) or the following situations do not occur:
    - \( S \equiv S'; \ell; \text{callM}_{ij}a_j(\ell \# \ell); S'' \) and \( S'' \) does not contain an opening bracket,
    - \( S \equiv S'; \ell; \); \( S'' \) and \( S'' \) does not contain a call-statement, or
    - \( S \equiv S'; \text{end}_{x,ij} \); \( S'' \) and \( S'' \) does not contain an opening bracket.
- A module configuration is acceptable if all its active LC's are
- A program configuration is acceptable if all its MC's are.

An inactive local configuration is always acceptable, since wait-statements never appear within bracketed sections. Hence, a configuration is acceptable if no local configuration is within a bracket section.

The following statement — implying soundness of the program rule — will be proved.

Suppose there is a program \( P \equiv [M_1] \ldots [M_n] \) and there are \( CA_i, CO_i, MI_i, p_i \) and \( q_i \) (\( i = 1..n \)) such that the premises of the program rule are valid. Then for any \( C_0 \xrightarrow{\lambda_1} C_1 \xrightarrow{\lambda_m} C_m \in \text{Comp}(P, p_1 \land \cdots \land p_n \land GI) \), where \( \varepsilon_i = s(C_i) \) for \( 0 \leq i \leq m \): \( \delta_m \models \bigcup_{i=1}^n CA_i, \delta_m \models GI \) if \( C_m \) is acceptable, and \( C_m = TC_\varepsilon(P) \Rightarrow \delta_m \models q_1 \land \cdots \land q_n \land GI \).
For every $\delta_i$, I can assume that $\text{Dom}(\sigma_i^f) \cap V_k = \emptyset$ and (by definition) that $\sigma_i^f \upharpoonright V = \sigma_i^k \upharpoonright V$ if $V = \text{Dom}(\sigma_i^f) \cap \text{Dom}(\sigma_i^k)$ for $k \neq j$ and for $j, k = 1 \ldots n$. The proof is by induction on the computation length, $m$.

$m = 0$ Then $\delta_0 = \delta_m$. $C_0$ is acceptable and by assumption $\delta_0 \vDash GI$. Note that $CA_j(\rho_{m,j}, \sigma_{m,j}) \equiv true$ for $1 \leq j \leq n$. Hence $\delta_m \vDash \bigwedge_{i=1}^{n} CA_i \land GI$. Finally, $C_0$ cannot be a terminal configuration.

$m > 0$ If $C_m$ is a terminal configuration, then $\delta_m \vDash \bigcup_{i=1}^{n} CA_i$ is vacuously true. Since in this case $C_{m-1}$ is acceptable too, induction implies $\delta_{m-1} \vDash GI$, hence $\delta_m \vDash GI$ as there is no (variable) state change involved in the $C_{m-1} \rightarrow C_m$-transition. The conditions on the assertions imply that $\delta_j^k = p_k$ and, using induction, that $\delta_j^k \vDash CA_k$ for $0 \leq j < m$, $1 \leq k \leq n$. By projecting the computation sequence on the transitions of each of the individual modules, I may use validity of the module specifications to conclude that $\delta_m^k \vDash q_k$ for $1 \leq k \leq n$. Because of the conditions on the module states and the assertions, I can conclude that $\delta_m \vDash q_1 \land \cdots \land q_n \land GI$.

Now suppose that $C_m$ is not a terminal configuration. Since $\delta_m \vDash CA_k$ if $\delta_m^k \vDash CA_k$, I may assume that $C_m$ is acceptable. Next, observe that because the free variables in $GI$ may only be assigned to in bracketed sections, it is sufficient to show that $GI$ is reestablished at the end of any bracketed section.

If $C_{m-1}$ is acceptable then the result follows by induction. So assume otherwise. I.e., assume that $C_{m-1}^j$ with $a^j_{m-1} \neq -1$, has $\rho_{m,a^j_{m-1}} = \ell$ and $\ell$ labels some closing bracket (in $M_j$). This implies that if $\lambda_1 \cdots \lambda_m = \lambda$, then $\lambda = \lambda' \cdot c$ with $c \equiv (i, h, b, j, k, \text{ind})$ for some $i, j, k, h, \text{ind}$ and $b$. The computation can be split as follows:

$$C_0 \xrightarrow{\lambda'} C_q \xrightarrow{c} C_{q+1} \xrightarrow{\epsilon} C_m$$

There are two possibilities: Either $c$ witnesses the start of a synchronization period between $M_i$ and $M_j$ or $c$ witnesses the end of such a period. These cases are analyzed separately:

1. There are statements, $\ell . (\ell . \text{call } M_j.a_k(\ell \# \ell ); \ell \ell . \ell )$ in $M_i$ and $\alpha_k(\ell \# \ell ); \ell \ell . \ell )$ in $M_j$; $\alpha^i_1 = -1$, $\alpha^j_2 = -1$, $C^i_{q,a^i_1} \equiv i, h, a : (\ell . \text{call } M_j.a_k(\ell \# \ell ); T_q, \theta^i_{q,a^i_1})$, and $\rho^j_{m,a^j_{m-1}} = \ell'$.

This implies that there are configurations $C_p$ and $C_r$ such that $C_p \xrightarrow{\lambda''} \tilde{C}_p \xrightarrow{\lambda'} \tilde{C}_r \xrightarrow{\epsilon} \tilde{C}_m$.

Claim: There is a computation $\tilde{C}_0 \xrightarrow{\lambda_{1}'} \tilde{C}_q \xrightarrow{c} \tilde{C}_{q+1} \xrightarrow{\epsilon} \tilde{C}_r \xrightarrow{\epsilon} \tilde{C}_m$ such that $\tilde{C}_0 = C_0$, $\tilde{C}_m = C_m$, $\tilde{C}_p$ and $\tilde{C}_r$ are acceptable, $\tilde{C}_{p,a^j} = C^i_{p,a^j}$, $\tilde{C}_{r,a^j} = C^j_{r,a^j}$, $\tilde{C}^j_{r,a^j}$ involves only $M_j$-transitions, $\tilde{C}_{q+1} \xrightarrow{\epsilon} C_r$ only $M_j$-transitions and $\tilde{C}_r$ neither $M_i$ nor $M_j$-transitions$^6$. Hence, $\tilde{C}^j_{q,a^j} \equiv \tilde{C}^j_{q,a^j}$ and $\tilde{a}^j_q = -1$.

This claim follows trivially from the fact that $C_m$ is acceptable and that in any computation, $\tilde{C}_0 \xrightarrow{\lambda_{1}'} \tilde{C}_k \xrightarrow{\lambda_{1}'} \tilde{C}_l$, if on $C_k \xrightarrow{\lambda_{1}'} C_l$ no communications take place between two modules, $^6$Remember that $\rho^j_{m,a^j_{m-1}} = \ell'$
say M_r and M_s, any ordering of transitions from M_r and M_s is accidental and can be changed. It suffices to show the Lemma w.r.t. this sequence.

Define s(CA_j) = δ_k for 0 ≤ k ≤ m. By induction δ_p = ∪_{i=1}^n CA_i ∧ GI, which implies that δ_m = ∪_{k≠j} CA_k, hence δ_m = ∪_{k≠j} CA_k because FV(CA_k) ∩ V = ∅ for j ≠ k and CA_l(δ) = true. It remains to show that δ_m = CA_j ∧ GI and for this it suffices to show that δ_r = CA_j ∧ GI.

Now define configuration states δ_p and δ_r by

δ_p = (δ_0, (Ω, θ), ∪_{k=1}^n ρ^{δ_p}_k, p_k, a_k) and δ_r = (δ_0, (Ω, θ), ∪_{k=1}^n ρ^{δ_r}_k, p_k, a_k)

By definition, the right-hand sides are defined. Observe that by δ_p = CO_i(δ) ∧ MI_j ∧ GI = CO_i(δ) ∧ MI_j ∧ GI and δ_r = CA_j(δ) ∧ GI = δ_r = CA_j(δ) ∧ GI.

As in the soundness proof of the module rule, extract from δ_0 ··· δ_m for l = i, j the configuration state sequence δ_0 ··· δ_m, corresponding with the M_l-transitions in C_0 ⋂··· ⋂ C_p. The lth premiss of the program rule implies that (δ_0 ··· δ_m, t) ≡* (CA_i, θ, CO_i, MI_j | {p_l} M_l(q_l)). Since by assumption δ_0 = p_l and by induction δ_m = CA_i for 0 ≤ v ≤ m, conclude that δ_{k,m} = CO_i and δ_{k,m} = MI_j; hence that δ_p = CO_i(δ) ∧ MI_j ∧ GI.

Next observe that there is a sequence (δ_0 ··· δ_m, T) ∈ [S_1; ic.a_k := ic.a_k + 1; δ, δ := δ, δ; S_1'] such that δ_0 = δ_p and δ_δ = δ_r. Validity of the cooperation specifications says

|= (θ, θ, true | (CO_i(δ) ∧ MI_j ∧ GI)S_1; ic.a_k := ic.a_k + 1; S_1')

Substitution lemma 5–7(1) allows the statement-sequence in this specification to be replaced by S_1; ic.a_k := ic.a_k + 1; δ, δ := δ, δ; S_1'. Since δ_0 = CO_i(δ) ∧ MI_j ∧ GI, this implies that δ_d = CA_j(δ) ∧ GI which concludes the proof.

(2) There are statements t.(S_1; δ.call M_i, a_k(δ#δ); S_2; δ.) in M_j and a_k(δ#δ) : (δ, δ, δ', S); δ'.(S_1'); S_1', S, δ' in M_i; a_k δ ≠ δ', a_k δ ≠ δ', C_j, a_k δ ≡ i, h, a : (end_j, δ, δ); T_i, T_j, δ', a_k δ) and ρ_{m, a_k δ} = δ.

The proof continues as a complete analogon of the proof of case 1, except for the following: the cooperation specification that has to be used here, is

|= (θ, θ, true | (∀x CO_j(δ) ∧ CO_i(δ) ∧ GI)S_1'; ic.a_k := ic.a_k - 1; S_2 {CA_j(δ) ∧ GI})

Substitution lemma 5–7(2) implies

|= (θ, θ, true | (∀x CO_j(δ) ∧ CO_i(δ) ∧ GI ∧ δ = δ)S_1'; δ := δ; ic.a_k := ic.a_k - 1; S_2

Let C_p be the configuration such that C_p δ ≡ (i, h, a : (δ, δ, δ, end_j, δ, δ)). Then, additionally, I must argue that v_{p, a_k δ}(δ = ρ_{p, a_k δ}(δ). This, however, is easy because no M_l-transition occurred since the procedure instance was created and no M_i-transition will change the valuation of the variables in δ.
6-3 Theorem (Soundness theorem). Let \((CA, WA, CO, M! \mid \{p\}U\{q\})\) be any specification and \(T \subseteq \text{Th}(A)\). Then
\[ \vdash_{PS(T)} (CA, WA, CO, M! \mid \{p\}U\{q\}) \Rightarrow A \models (CA, WA, CO, M! \mid \{p\}U\{q\}) \]

Proof. The standard argument, using induction on the length of the proof and using Lemma 6-1. \(\square\)

7 Relative completeness

From now on, \(\Sigma\) denotes the signature of Peano-arithmetic (or some countable extension) and \(N\) is a standard model. Validity is relative to \(N\) and the proof system used is \(PS(Th(N))\).

7.1 Some definability results

7-1 Theorem ([Wei87, Corollary 2.9.12]. Definability of recursive sets). Let \(R \subseteq \omega\) be a recursive set. Then, there is a \(\phi_R \in L(\Sigma)\) such that \(\forall n \in \omega \, n \in R \equiv \phi_R(n)\), where \(n\) is the term representing the integer \(n\) (i.e., \(n \equiv \text{succ}^n(0)\)) and \(\phi_R(n)\) is a sentence.

7-2 Lemma. If \(R \subseteq St_n\) is computable, then there is a \(\phi_R \in L(\Sigma)\) such that \(\forall \sigma \in St_n \sigma \in R \Rightarrow N, \sigma \models \phi_R\).

Proof. For any \(\sigma \in St_n\), define \(\bar{\sigma} \in (\omega^2)^n \equiv \omega^{2n}\) as the \(n\)-tuple of pairs \((a_0, \ldots, a_{n-1})\) for \(0 \leq i \leq n:\)
\[ a_i = \begin{cases} (i, v) & \text{if } z_i \in \text{Dom}(\sigma) \text{ and } v = \sigma(z_i) \\ (n, 0) & \text{otherwise} \end{cases} \]

Observe that \(\tau : St_n \rightarrow \omega^{2n}\) is an injection. Let \(\langle \cdots \rangle_k\) be the \((k-1)\) iterated pairing function \(\omega^k \rightarrow \omega\) and let \(\pi_k^i (0 \leq i < k)\) be the associated projections.

Define \(R \subseteq \omega\) by \((a_0, \ldots, a_{n-1}) \in R \equiv \langle a_0 \rangle, \ldots, \langle a_{n-1} \rangle \in R\). Since \(R\) is recursive (Church's thesis), there is a \(\phi_R \in L(\Sigma)\) that defines it. Let \(\pi_k^i\) stand for the \(L(\Sigma)\)-formula that defines it. It is easy to show that the wanted formula, \(\phi_R\), is given by \(\exists y \left[ \phi_R(y) \rightarrow \bigwedge_{i=0}^{n-1} (\pi_k^i(\pi_k^i(y)) \neq \bar{\sigma}(\tau(z_i))) \right]\), where \(y \not\in \text{Var}_n\). \(\square\)

7-3 Corollary. Let \(Ms_n\) be the set of module specification states such that the set of variables that is valuated, is contained in \(\text{Var}_n\). If \(R \subseteq Ms_n\) is a computable set, then there is a \(\phi_R \in L(\Sigma)\) such that \(\forall \delta \in Ms_n : \delta \in R \iff \exists \delta' \in Ms_n \sigma(\nu) = \delta'(\nu) \& N, \delta' \models \phi_R\).

Proof. Evident. \(\square\)

Note that in these lemma's, "recursive" can be replaced by "arithmetical" ([Wei87]). Definability as in Lemma 7-2 and Corollary 7-3 will be called representability. The representing formula for a module configuration state, \((\alpha, \theta_1, \ldots, \theta_n, \sigma)\), does neither fix \(\alpha\), nor \(\rho_1, \ldots, \rho_m\), nor the partition of \(\text{Var}_n\) into \(\text{Dom}(\sigma)\) and \(\text{Dom}(\nu_i)\) for \(i = 1..n\).
Facts:

1. For every specification, \( \langle CA, WA, CO, MI \mid \{p\}T\{q\} \rangle \), there is an \( n \in \omega \) such that the variables and formal parameters appearing (free) in the specification, is contained in \( \text{Var}_n \).

2. If \( \models \langle CA, WA, CO, MI \mid \{p\}T\{q\} \rangle \), then there is an a-priori upperbound, \( \bar{n} \), such that there is a proof of this specification for which any specification appearing in this proof has its free variables in \( \text{Var}_{\bar{n}} \).

Of course, this last fact is a consequence of the completeness proof. Inspection of this proof yields that a safe bound is \( 2m + n + \sum_{i=1}^{n} k_i \), where \( m \) bounds the cardinality of the set of variables in the specification to be proved, \( n \) is the number of modules and \( k_i \) is the number of procedures in the \( i \)-th module.

7–4 Lemma. Let \( \models \langle CA, WA, CO, MI \mid \{p\}T\{q\} \rangle \) and let all free variables be contained in \( \text{Var}_m \). If \( \delta \) is a configuration state, let \( \delta' \) be the configuration state obtained by restricting every valuation of \( \delta \) to \( \text{Var}_m \). Then \( \models \langle CA, WA, CO, MI \mid \{p\}T\{q\} \rangle \) is equivalent with

\[
\forall(\delta_0 \cdots \delta_n, t) \in [T] \quad (\delta_0 \cdots \delta_n, t) \models \langle CA, WA, CO, MI \mid \{p\}T\{q\} \rangle
\]

Proof. Evident.

The consequence of all this is, that given a valid specification, in its proof one can restrict oneself a priori to a finite set of variables. This allows the "completeness-assertions" to be defined and allows Lemma 7–3 to be used. In the representability definitions in the proof below it will be up to the reader to check that the sets to be represented are in fact arithmetical. These results are always straightforward consequences of the observations 3–13, 4.2 and 5–4.

From now on, every \( CM^\ast(\Sigma) \)-statement, procedure, module or program and every specification is assumed only to contain variables from the a priori fixed set \( \text{Var}_N \). Also, since \( < \cdots >_k \) is definable as a term ([Wei87]), it will be freely used in \( CM^\ast(\Sigma) \)-statements. Concatenation (i.e., pairing) of two sequences is denoted by \( h \cdot h' \) instead of \( < h, h' >_2 \). Finally, I shall write \( < \cdots > \) instead of \( < \cdots >_k \) and leave the determination of \( k \) to the context.

7.2 Completeness for statement and procedure specifications

7–5 Definition (weakest precondition).

If \( S \) is a statement or procedure, \( WP(CA, WA, CO, MI, q, S) \) represents the set

\[
\{ \delta \mid \delta_0 = s(SC_{en}(T)) \& \forall \delta_1, \ldots, \delta_n : (\delta_0 \cdots \delta_n, t) \in [T](CA \cup WA) \Rightarrow \\
\delta_n \models CO \& a_n = -1 \Rightarrow \delta_n \models MI \& t = T \Rightarrow \delta_n \models q \}
\]

7–6 Lemma.

1. \( N \models \langle CA, WA, CO, MI \mid \{ WP(CA, WA, CO, MI, q, S) \}S\{q\} \rangle \)

2. \( N \models \langle CA, WA, CO, MI \mid \{p\}S\{q\} \rangle \iff N \models p \rightarrow WP(CA, WA, CO, MI, q, S) \)

Proof. Directly from the definitions.
7-7 Lemma. For any CM*(Σ)-statement, procedure, module or program, T, and for any specification, \((CA, WA, CO, MI \mid \{p\}T(q))\):

\[ \vdash_{PS(Th(\Sigma))} (CA, WA, CO, MI \mid \{p\}T(q)) \Rightarrow \vdash_{PS(Th(\Sigma))} (CA, WA, CO, MI \mid \{p\}T(q)), \]

where

- \(\overline{CA}, \overline{WA}\) and \(\overline{CO}\) are obtained from \(CA, WA\) and \(CO\) by removing all labeled formulas, \(t : p\), for which the label \(t\) does not appear in \(T\), and

- \(\overline{MI} = \{true\ \ \text{if no waiting point occurs in } T\ \ \text{otherwise}\)

Proof. This is a trivial application of the consequence rule. □

Specifications as on the left-hand-side of Lemma 7-7 are called minimal.

7-8 Theorem (completeness).

If \(S\) is a CM*(Σ)-statement or procedure, then

\[ \vdash_{PS(Th(\Sigma))} (CA, WA, CO, MI \mid \{p\}S(q)) \]

Proof. Structural induction on \(S\). Here too, only the representative cases will be treated. Because of Lemma 7-7, I can assume the specification to be minimal.

7-8(1) \(S \equiv \exists z := e\):

Observe that \(WP(0, WA, CO, MI, q, S)\) represents the set \(\{(0, (\Omega, \ell), \sigma) \mid \sigma \in CO(\ell) \wedge MI\}\). Hence, Lemma 7-6(2) implies \(\vdash (0, WA, CO, MI \mid \{p\}t.wait.\ell'\{q\})).

7-8(2) \(S \equiv \ell.wait.\ell'\):

\[ WP(0, WA, CO, MI, q, S) \]

represents the set \(\{(0, (\Omega, \ell), \sigma) \mid \sigma \in CO(\ell) \wedge MI\}\). Hence, Lemma 7-6(2) implies \(\vdash (0, WA, CO, MI \mid \{p\}t.wait.\ell'\{q\})).

7-8(3) \(S \equiv \ell.(S_1; \ell'.callM,a_i(\#\#); S_2; S')\):

Define \(q' \equiv CA(\ell') \rightarrow q\) and \(q'\) is the weakest such formula. Define \(r \equiv WP(0, 0, 0, true, q', S_2)\) and \(CO' \equiv \{\ell : p, \ell : r\}.\) To show (1) \(\vdash (0, 0, 0, true \mid \{CO'(\ell)\}S_2(q')\})\) and (2) \(\vdash (0, 0, 0, true \mid \{p\}S_1(\forall \neq CO'(\ell))).\)

Validity of (1) follows from the definitions of \(r\) and \(CO'(\ell')\). For validity of (2), first note that a call acts (locally) as a random assignment, \(\emptyset := ?,\) so that \(\vdash \forall \neq CO'(\ell') \rightarrow WP(0, 0, 0, true, CO'(\ell'), callM,a_i(\#\#)).\) Then validity of the specification and the definition of \(q'\) (hence of \(CO'(\ell')\)) gives validity of (2). Induction yields proofs of (1) and (2) (in \(PS(Th(\Sigma))\)). The call rule yields \(\vdash (CA, 0, CO', true \mid \{p\}S(q' \wedge CA(\ell')))).\) By definition \(\vdash CO'(\ell') \rightarrow CO(\ell), CO'(\ell') \rightarrow CO(\ell')\) and \(\vdash q' \wedge CA(\ell') \rightarrow q.\) So a final application of the consequence rule suffices.

7-8(4) \(S \equiv a(a#\# \mid \{S_1; \ell; S'; \ell'\{S_2\})\):

Define \(\overline{CA}, \overline{WA}\) and \(\overline{CO}\) in the obvious way as those parts of the assumption and commitment pertaining to \(S\). Since the formal parameters \(u\) and \(\overline{v}\) do not appear outside \(S, FV(p, q) \cap \{u, \overline{v}\} = \emptyset\) may be assumed. Let \(r \equiv WP(0, 0, 0, true, q \wedge MI, S_2) \wedge CO(\ell')\) and \(s \equiv WP(CA, WA, CO, MI, r, S),\)

Clearly \(\vdash CO(\ell') \rightarrow r, \vdash q \wedge MI \rightarrow MI, \vdash (0, 0, 0, true \mid \{r\}S_2(q \wedge MI))\) and \(\vdash (CA, WA, CO, MI \mid \{r\}S_2(r)).\) Next, define \(s' \equiv CA(\ell') \rightarrow s.\) Then \(\vdash s' \wedge CA(\ell') \rightarrow s\) and \(\vdash (0, 0, 0, true \mid \{s\}S_2(r)).\)
\{p \land WA(\ell')S_i\{s'\}\} by the same reasoning as in (3). Hence, induction allows the procedure rule to be used:

\[ \Gamma \vdash (CA, WA, CO, MI \mid \{p\}S\{q\}) \]

\[ \square \]

### 7.3 Restricted completeness for module specification

To obtain completeness results in this case, auxiliary variables are needed that record the computation history. The reason for this is the necessity of using the local merging lemma in order to show validity of the interference freedom specifications. For the same reason, I cannot show general completeness: for that I would need to be able to distinguish between coexisting instances, which is not always possible. Hence, each procedure, \(a_j\), obtains two extra read-only parameters, \(m_j^l\) and \(i_j^l\), which are intended to be initiated in each instance so as to uniquely identify this instance. Specifically, they are instantiated to the indices of the calling module and procedure. This makes the local states of coexisting instances unique, provided the communication assumption, \(CA\), expresses the above intention.

Below, I assume that to every label, \(\ell\), a unique non-zero integer is associated that is also denoted by \(l\); associate 0 with "no label". Also, assume that "in" and "te" stand for some arbitrary but different values.

### 7-9 Definition

Let \(M_i\) be some \(CM^*(\Sigma)\)-module. Introduce variables, generically denoted by \(m_j^l, i_j^l, h_l, \bar{h}_l\), such that \(m_j^l, i_j^l, h_l, \bar{h}_l \notin V_i\) for \(j = 1 \ldots m_i\). Let \(\{\bar{y}\} = V_i\) and let \(\{\bar{w}_j\} = L_{l,i,j} \cup \{m_j^l, i_j^l\}\). The module \(M'_i\) is defined as \(M_i\) except that

- any wait-statement, \(\ell\).wait.\(\ell'\) in any procedure \(a_j\) (\(j \geq 0\)) is replaced by
  \[ h_l := \bar{h}_l^\sim < l, j, m_j^l, < \bar{w}_j >, \ell, < \bar{y} >; \ell\).wait.\(\ell'\) \]
- any call-statement, \(\ell'.(S_i; \ell'.callM.a_s(\ell\#\bar{y}); S_2; \ell''.)\), in any \(a_j\) (\(j \geq 0\)) is replaced by
  \[ \ell'.(S_i; h_l := h_l^\sim < l, j, l, j, \bar{e}, \bar{e} >, r, s, in >; \bar{h}_l := <>; \ell'.callM.a_s(l, j, \bar{e}\#\bar{y}); h_l := h_l^\sim < r, s, < \bar{e} >, l, j, te >; S_2; \ell''.) \]
- any procedure \(a_j^l(u\#\bar{v})\): \([S_0; \ell]; S; \ell'.(S_2)\] is replaced by
  \[ a_j^l(m_j^l, i_j^l, u\#\bar{v}) : [h_l := h_l^\sim < m_j^l, i_j^l, < m_j^l, i_j^l, u, \theta >, l, j, in >; \bar{h}_l := <>; S_i; \ell]; S; \ell'.(S_2; h_l := < h_l^\sim < l, j, < \theta >, m_j^l, i_j^l, te > < l, j, m_j^l, < >, 0, < \bar{y} >; \bar{h}_l := <>)] \]
- the initial statement: \([S_0; \ell_0]\) is replaced by \(S_0; h_l := h_l^\sim < 0, 0, < >, \ell_0, < \bar{y} > > \ell_0. \)

### 7-10 Definition (local coding, lc)

Let \(l, m \in \omega\) and let \(\lambda_1 \cdots \lambda_n\) be a transition label sequence. Then

- \(m\) locally codes (lc) \(\lambda_1 \cdots \lambda_n \Downarrow l\) iff \(m = << \lambda_1, >, \ldots, < \lambda_n >>\), where \(\lambda_1 \cdots \lambda_n \Downarrow l\) and where \(< \lambda_i >\) stands for the code of \(\lambda_i\) as shown above.

Note that there is a 1-1 correspondence between a label sequence and its code.
7-11 DEFINITION. Let \( M_i \) be a CM*\( (\Sigma) \)-module, \( p \in L(\Sigma) \) and \( A \subseteq \{ \ell : p | \ell \in FLB(M_i) \} \), \( p \in L(\Sigma) \).

- \( SMI(A, p, M_i) \) represents
  \[
  \left\{ (-1, \Omega, \sigma_n) \mid \exists C_0 \xrightarrow{\lambda_1} C_1 \cdots \xrightarrow{\lambda_n} C_n \in \text{Comp}(M_i, p, A) \text{ with } C_0 \in SC \right\}
  \]
  such that \( \alpha_n = -1 \) and \( \sigma_n(h_i, \bar{h}_i) \in \lambda_1 \cdots \lambda_n \)

- \( COMP(A, p, M_i, \ell) \) represents
  \[
  \left\{ (\alpha_n, \theta_{n, \alpha_n}, \sigma_n) \mid \exists C_0 \xrightarrow{\lambda_1} C_1 \cdots \xrightarrow{\lambda_n} C_n \in \text{Comp}(M_i, p, A) \text{ with } C_0 \in SC \right\}
  \]
  such that \( \alpha_n \neq -1 \), \( \rho_n, \alpha_n = \ell \) and \( \sigma_n(h_i, \bar{h}_i) \in \lambda_1 \cdots \lambda_n \)

- Let \( \ell \in FLB(a^i_j) \). Then
  - if \( j = 0 \) then \( U(\ell) = \text{true} \),
  - if \( j > 0 \) then
    \[
    U(\ell) = \exists g, g' \left( \text{active}_{h}(h_i, g, g', m^i_j, z^i_j) \land \neg \exists g'', \text{active}_{g''}(g, g', m^i_j, z^i_j) \land \right.
    \]
    \[
    \text{ic.a}^i_j > 1 \implies \exists g', g'', m, p \text{ active}_{g''}(g', g'', m, p) \land \text{active}_{g''}(g, g', g'', m, p) \right)
    \]
    where \( \text{active}_{h}(h, f, g, m, p) = \exists c h = f' < m, p, c, i, j, m' \text{ and } \right) \)
    \[
    \neg \exists g', g'', c' | g' = g'' < l, j, c', m, p, \text{ te } > g''
    \]

The predicate \( \text{active}_{h}(h, f, g, m, p) \) expresses that the (code of the) history \( h \) witnesses an active instance of procedure \( a^i_j \), which is being called by procedure \( a^m_i \) (in \( M_m \)); moreover \( f \), respectively, \( g \) codes the history before, respectively, after this instance becomes active. Then, the assertion \( U(\ell) \) first claims that as long as an instance of \( a^i_j \) with particular values for \( m^i_j \) and \( a^i_j \) is active, there can be no second instance with the same values for \( m^i_j \) and \( a^i_j \). This implies that these values uniquely identify procedure instances. \( U(\ell) \) also claims that if \( ic.a^i_j > 1 \) holds, there must be another active instance of \( a^i_j \).

7-12 LEMMA. Given a specification \( \langle CA, WA, CO, MI | \{ p \} M_i \{ q \} \rangle \), define
- \( p \equiv \emptyset \land h_i = < > \land \bar{h}_i = < > \land \emptyset = \bar{\emptyset} \), with \( \{ \emptyset \} = V_i \) and \( \{ \bar{\emptyset} \} \cap V_i = \emptyset \)
- \( SWA \equiv \{ \ell : \text{COMP}(CA, p, M_i, \ell) \land U(\ell) | \ell \text{ labels the rear of a wait in } M_i \text{ or the rear of its initial statement} \} \)
- \( SCO \equiv \{ \ell : \text{COMP}(CA, p, M_i, \ell) \land U(\ell) | \ell \text{ labels the front of a wait, a call or an opening bracket in } M_i \} \)
- \( SMI \equiv SMI(CA, p, M_i) \)
- \( SP \equiv \text{COMP}(CA, p, M_i, \ell) \), where \( \ell \) labels the rear of the initial statement.
- Assume that \( CA \) is such that
  - if \( \ell \) labels the first (closing) bracket in some \( a^i_j \) (\( j > 0 \)) then \( \vdash CA(\ell) \rightarrow U(\ell) \)
  - \( CA \) is LC-independent

(These two constraints are compatible.)

Then \( \vdash_{PS(\tau(h(A)))} \langle CA, \emptyset, SCO, SMI | \{ p \} M_i \{ SP \} \rangle \)

33
Proof. Observe that $\models SWA(\ell) \rightarrow SMI$ for any rear wait or rear initial statement label, $\ell$.

For any $A \subseteq \{ \ell : p \mid \ell \in Lab, p \in L(\Sigma) \}$, let $A_j = \{ \ell : p \mid p \in A, \ell \in FLB(a^j_i) \}$. Again, Lemma 7-7 allows the specification to be taken minimal. By Theorem 7-8, validity of the corresponding premisses of the module rule has to be shown. Only validity of the interference freedom specifications will be treated in detail.

7-12(1) $\models (CA_k, SWA_k, SCO_k, SMI \mid \{ SMI \} a_k^i (u \# v) : [S_k] \{ SMI \})$, $k > 0$:

The following fact follows directly from the definition of $SWA, SCO$ and $SMI$, and its proof is left to the reader:

Supose that $(\delta_0 \ldots \delta_m, i) \models^* (CA_k, SWA_k, SCO_k, SMI \mid \{ SMI \} a_k^i (u \# v) : [S_k] \{ SMI \})$

$\delta_0 \models SMI$ and $\delta_i \models CA_k$ for $i = 0..n$. Then there is a computation $C_0 \vdash \lambda_i \vdash C_1 \vdash \ldots \vdash C_n \in Comp(M'_i, p, CA)$, with $\delta_i = s(C_i)$ for $i = 0..n$ and $3i_0, i_1, \ldots, i_m$ such that $\forall 0 \leq j \leq m \: \exists j \in (V_i \cup \{ \text{ic}a_1^j \}) = \sigma_{ij} \in (V_i \cup \{ \text{ic}a_1^j \})$, $\delta_j = a_{ij}$, if $a_{ij} = -1$ then $\delta_j = a_{ij}, a_{ij} = a_{i_{j-1}}$ else $\delta_j = a_{ij}, a_{ij} = a_{i_{j-1}}$, and $\sigma_m(h_i = h_i)$ le $\lambda_1 \ldots \lambda_n$.

From this, validity is easily established.

7-12(2) $\models (CA_0, SWA_0, SCO_0, SMI \mid \{ p \} S_0 (SMI \land SWA_0 (\ell_0)))$: Analagous to 1, since a similar fact holds.

7-12(3) $\models SMI \land SWA_0 (\ell_0) \rightarrow SP$: Evident.

7-12(4) for every $\ell$, wait.$\ell'$ in $M'_i$: $\models SCO(\ell) \rightarrow SWA(\ell')$: Immediately from the definitions.

7-12(5) $SWA \# (CA_k, SWA_k, SMI, \{ u_k, v_k \}, a_k^i (u_k \# v_k) : [S_k])$ for $k > 0$:

So, for every $\ell : q \in SWA$, I must prove that

$\models < CA_k, SWA_k (q'), CO, true \mid \{ SMI \land q' \} a_k^i (u_k \# v_k) : [S_k] \{ q' \}>$

Let $\ell \in FLB(a^i_k)$. Then, I must show that

$$\forall C_0 \vdash \lambda \vdash C_n \in Comp(a^i_k (u_k \# v_k) : [S_k], SMI \land q', CA \land SWA_k (q'))$$

$s(C_n) = CO \land C_n \in TC \Rightarrow s(C_n) = q'$.

Fix a computation and let $C_i = \langle A \mid a_i \mid LC_i, \sigma_{i} \rangle$, $LC_i = Id_i : (S_i, \theta_i)$ and $\delta_i = s(C_i)$ for $i \leq n$.

If $C_n \notin TC$ or $\rho_0$ does not label the front of a wait, there is nothing to prove. So, assume otherwise. In either case, I must show that $\delta_n \models q'$. Obviously, it suffices to prove this on sequences for which $a_i \neq -1$ for $i \leq n$. There are two cases; namely, $\rho_0 \neq 0$ and $\rho_0 = 0$. In the first case, the computation restarts at a block-statement. The latter case corresponds with a computation starting in the procedure's initial configuration.

Case 1: $\rho_0 \neq 0)$ Then $\delta_0 \models SWA(\rho_0) \land q'$, with $q' \equiv SWA(\ell)[\bar{y}/\bar{u}_k, \bar{v}_k]$. Let $\nu_0(< m_i^1, i_k^1 >) = h$ and $\sigma_0(< m_i^1, i_k^1 >) = h'$, where $m_i^1$ and $i_k^1$ are the fresh variables substituted for $m_k^1$ and $i_k^1$. Note that $h \neq h'$. By interpreting the assertions, I obtain (as explained below) that

1. $\delta_0 \models SWA(\rho_0) \Rightarrow (h, \rho_0) = (M'_i, p, CA, \nu_0, \sigma, \lambda)-Ir, (h' \sim h) \in \lambda$ and $\sigma \cup \nu \models U(\rho_0),$

2. $\delta_0 \models q' \Rightarrow (h', \ell) = (M'_i, p, CA, \nu, \sigma, \lambda)-Ir, (h' \sim h) \in \lambda$ and $\sigma \cup \nu \models U(\ell)$,

where $\sigma = \sigma_0 \land V_i$ and $\nu = \Omega(\sigma_0(\bar{y}/\bar{u}_k, \bar{v}_k))$.

These implications depend on the fact that $(L_k \cup L_k) \cap V_i = \emptyset$. This determines the splitting of the variable valuations (in $\delta_0$) into $\nu_0, \sigma$ and $\nu, \sigma$, as this must be the case in any configuration of
The only moot point is why I can choose \( h \) in (1) and \( h' \) in (2): Since \( \sigma \cup \nu_0 = U(\rho_0) \) there can be no other active instance of \( a_L \) with the same values for \( m_L \) and \( i_L \). This means that there is a computation in which the LC leading to \((\nu_0, \rho_0)\) is identified by \((l, k, h)\). If \( k = k' \) then \( \delta_{l(h-c.a_l)} > 1 \) so that \( U(\rho_0) \) implies the existence of another active instance. As \( h \neq h' \), the same argument gives, both in the case that \( k = k' \) and in the case that \( k \neq k' \), the reachability of \((\nu, l)\) by an LC with identification \((l, k', h')\).

As \( CA \) is LC-independent, the LML applies, so that \((h, \rho_0, h', l)\) is \((M'_1, p, CA, \nu_0, \nu, \sigma, \lambda)-br\). Note that \( \lambda = \lambda' \) because they are both locally coded by the same integer.

Hence, there is a computation,

\[
\tilde{C}_0 \xrightarrow{\lambda} \tilde{C}_m \in \text{Comp}(M'_1, p, CA),
\]

with \( \delta_i = s(C_i) \) for \( i \leq m \) and such that \( \exists r, s \bar{d}_{m,r} = (\nu, \ell), \bar{d}_{m,s} = (l, k', h'), \bar{d}_{m,s} = \theta_0, \bar{d}_{m,s} = (l, k, h), \bar{d}_m = \sigma \) and \( \sigma(h_1, h) \mid \ell \mid \lambda \). Also, I can assume that \( \bar{d}_m = -1 \): by definition of \( SWA(\rho_0) \) and of \( q' \) the last record in \( \lambda \) is a local one and the global state must be attainable at a waiting point.

Next, extend (9) with \( \tilde{C}_m \xrightarrow{\lambda} \tilde{C}'_{m+n} \) as follows:

- \( LC_{m+i,j} \equiv LC_{m,j} \) for \( j \neq s \),
- \( LC_{m+i,s} \equiv LC_i \),
- \( \sigma_{m+i} = \sigma_i \mid V_i \) and
- \( \bar{d}_{m+i} = s \)

This extension is well-defined because I can assume that \( \bar{d}_m = (l, k, h) \). Clearly, \( \sigma_{m+n} \cup \nu_{m+n,r} \mid SWA(\ell) \) holds. This means that \( \sigma \mid V_i \mid SWA(\ell) \), hence \( \sigma \mid q' \) and, finally, \( \delta_0 \mid q' \).

Case 2: \( \rho_0 = \emptyset \) The argument is analogous to Case 1, except that the LML is not needed.

This is completely analogous to case 1 above. The second case, \( \rho_0 = \emptyset \), does not apply because the pre condition is false, reflecting the fact that on no computation sequence of \( M_l \) a procedure instance can be active before the initial statement has reached a waiting point.

7.4 Completeness for program specifications

Here, too, auxiliary variables are needed; this time in order to apply the global merging lemma.

7-13 Definition. Let \( P \) be a \( CM(\Sigma) \)-program \([M_1] \cdots [M_n]\). Then

- \( P \) is a \( CM'(\Sigma) \)-program \([\tilde{M}_1] \cdots [\tilde{M}_n]\) such that \( P \) can be re-obtained by removing every bracket or label from \( P \)
- \( P' \) is the \( CM'(\Sigma) \)-program, \([M'_1] \cdots [M'_n]\), obtained from \( P \) by modifying every module \( M_l \) as in Definition 7-9.

7-14 Definition (global coding, gc). Let \( i, m \in \omega \) and let \( \lambda_1 \cdot \cdots \cdot \lambda_m \) be a transition label sequence. \( m \) globally codes \((gc) \lambda_1 \cdot \cdots \cdot \lambda_m \downarrow_{g} i \) iff \( m = \langle \lambda_{i_1} >, \ldots, < \lambda_{i_k} > \rangle \) where \( \lambda_{i_1} \cdot \cdots \cdot \lambda_{i_k} = \lambda_1 \cdot \cdots \cdot \lambda_m \downarrow_g i \)
7-15 DEFINITION. Let \( P \) be a \( CM(\Sigma) \)-program \([M_1 \| \cdots \| M_n] \), \( p \in L(\Sigma) \), \( \ell \in FLB(M_k) \) and \( A \subseteq \{ \ell : p \mid \ell \in Lab, \ p \in L(\Sigma) \} \).

- \( \text{COMP}_p(V, \ell) \) represents
  \[
  \left\{ (a, (v, \ell), \sigma) \mid \exists C_0 \xrightarrow{\lambda}^* C_m \in \text{Comp}(P', p) \text{ such that } a_k^* \neq -1, \ \theta_k^* a_k^* = (v, \ell), \right. \\
  \text{ and } \sigma_k^* | V = \sigma | V \text{ and } \sigma(h_k \sim \bar{h}_k) \text{ lc } \lambda \downarrow k \right\}
  \]

7-16 LEMMA. Given a \( CM(\Sigma) \)-program, \( P \equiv [M_1 \| \cdots \| M_n] \), and \( p \in L(\Sigma) \), let \( \{z_i\} = V_i \) for \( i = 1..n \) and let \( \bar{z}_1, \ldots, \bar{z}_n \) be pairwise disjoint sequences of variables such that \( \bigcup_{i=1}^{n} \bar{z}_i \cap \bigcup_{i=1}^{n} (V_i \cup L_\ell) \cup FV(p) = \emptyset \) and \( |z_i| = |\bar{z}_i| \) for \( i = 1..n \).

Define for \( i = 1..n \)

- \( p_i \equiv \bar{p}[\bar{z}_j/\bar{z}_j, \ j = 1..n, \ j \neq i] \land \bar{z}_i = z_i \land h_i = < > \land \bar{h}_i = < > \) and \( p \equiv \bigwedge_{i=1}^{n} p_i \)
- \( \text{SCA}_i \equiv \{ \ell : \text{COMP}_i(V_i \cup \{z_i\}, \ell) \land U(\ell) \mid \ell \text{ labels a closing bracket in } M_i' \} \)
- \( \text{SGI} \) represents
  \[
  \left\{ (a, \theta, \sigma) \mid \exists C_0 \xrightarrow{\lambda}^* C_m \in \text{Comp}(P', p) \text{ with } \delta_n = s(C_n) \text{ such that } C_n \right. \\
  \text{ is acceptable, } \sigma(h_i) gc \lambda \text{ and } \sigma_n(h_i) = \sigma(h_i) \text{ for } i = 1 \ldots n \right\}
  \]

Then, every specification in the set of cooperation specifications,

\[
\text{Coop} \left( \{ \text{SCA}_i, \emptyset, \text{SCO}_i, \text{SMI}_i, \text{SGI} \mid \{ p_i \} M_i' \{ SP_i \} \mid i = 1..n \} \right),
\]

is valid.

Here, \( \text{SCO}_i, \text{SMI}_i \) and \( \text{SP}_i \) for \( i = 1..n \) are defined as in Lemma 7-15 (w.r.t. \( \text{SCA}_i \) and \( p_i \)). The indices \( i \) denote the modules to which they belong. Note that \( \text{SCA} \) is LC-independent because \( \text{COMP}_k \) is.

This proof is somewhat less detailed than that for module specifications, because the constructions of the computations are similar. It can be short because much of the work has been done in the proof of the GML.

PROOF. Take any matching call procedure pair

\[
\ell.(S_1; \ell.\text{callM}_j.\alpha_k(\ell\#\bar{z}); S_2; \ell.) \quad \text{(in } M_i) \quad \text{and}
\]

\[
\alpha_k(\#\bar{v}) : [S'_1; \ell']; S'_2; \ell'); S'_2 \quad \text{(in } M_j) 
\]

Since the assumptions, commitment and monitor invariant are trivial in the cooperation specifications, they will not be mentioned explicitly below.

\[
7-16(1) \models \{ \text{SCO}_i(\ell) \land \text{SMI}_j \land \text{SGI} \} S_1; \ i.c.\alpha_k^* := i.c.\alpha_k^* + 1; S'_1; [\text{SCA}_j(\ell')]; \text{SGI}
\]

Substitution lemma 5-7(1) yields equivalence with

\[
\models \{ \text{SCO}_i(\ell) \land \text{SMI}_j \land \text{SGI} \} S_1; \ i.c.\alpha_k^* := i.c.\alpha_k^* + 1; \bar{u}, \bar{v} := \bar{z}, \bar{z}; S'_1; \{ \text{SCA}_j(\ell') \land \text{SGI} \} \quad (10)
\]

Denote the statement sequence in (10) by \( T \). Choose any configuration state, \( \delta \) such that \( \delta \models \text{SCO}_i(\ell) \land \text{SMI}_j \land \text{SGI} \) and let the configuration state, \( \bar{\delta} \), be determined by the condition that \( (\delta \cdots \bar{\delta}, \bar{T}) \in [\bar{T}] \). If there is no such \( \bar{\delta} \), (10) holds trivially. So assume otherwise. To show:
\( \delta \models SCA_j(\ell') \land SGI \). W.l.o.g., assume that \( \text{Dom}(\delta) = V'_i \cup V''_j \cup L \) and that \( \delta \models V'_k = \sigma_k \) for \( k = i, j \) and that \( \delta \models L_i = \nu \).

By Lemma 4–7, \( \delta \models SCO_i(\ell) \land SGI \) implies that for some \( h, (h, \ell) \) is \( (P', p, \nu, \sigma, \lambda) \)-gr and \( \sigma_i(h, \lambda) = \lambda \mid i \), whence \( \sigma_i(h, \lambda) = \lambda \mid i \). Similarly, \( \delta \models SMI_j \land SGI \) implies that for some \( h, (h, \ell) \) is \( (P', p, v, \sigma, \lambda) \)-gr and \( \sigma_j(h, \lambda) = \lambda \mid j \), whence \( \sigma_j(h, \lambda) = \lambda \mid j \). Since \( (P', \ell' \in \lambda \} \) is some arbitrary local state in the final configuration of the computation implied by \( SMI_j \) (note that at least one such state exists). The foregoing proof is a long one indeed. The programming language is, however, quite powerful, embodying both Ada-type task interactions as well as shared variable concurrency. Previously published proofs in this area have dealt with simple shared variable concurrency (cf. [Apt81]) and with CSP-type concurrency (cf. [Apt83]) separately. This paper has to deal with both and also with their interaction. Moreover, a conscious attempt was made to obtain a notion of specification and a proof system that comes close to being compositional. This contrasts with [Apt81, Apt83] which rather consider global verification algorithms (in the sense of the introduction).

8 Conclusions

The foregoing proof is a long one indeed. The programming language is, however, quite powerful, embodying both Ada-type task interactions as well as shared variable concurrency. Previously published proofs in this area have dealt with simple shared variable concurrency (cf. [Apt81]) and with CSP-type concurrency (cf. [Apt83]) separately. This paper has to deal with both and also with their interaction. Moreover, a conscious attempt was made to obtain a notion of specification and a proof system that comes close to being compositional. This contrasts with [Apt81, Apt83] which rather consider global verification algorithms (in the sense of the introduction).
A second difference is in the formulation of the two merging lemmas of Section 4. Here, they are statements about the computations of modules and programs whose truth is independent from whether or not computations of procedures and modules are representable within the assertion language. Such a separation of concern is absent in earlier proofs and gives some conceptual clarity.

The proof system ultimately deals with partial correctness properties of $CM(\Sigma)$-programs. In fact, the axiomatisation is powerful enough to support the proof of arbitrary safety properties, as may be gleaned from the completeness proofs. However, to extend the notion of validity to program specification with non-trivial commitments is not so easy. The problem originates in the use of the global invariant, $GI$, and its bracketed sections. The $GI$ only holds in acceptable program configurations (in which no module is within a bracketed section), so that acceptability would have to be included in the validity definition. This added complication did not seem worth the trouble.

The proof system is not complete for module specifications. This points to a difference between the current proof system and a truly compositional one. The local merging lemma shows the need to distinguish procedure instances other than through the local and global states—which, indeed, may not show a difference. Here, this has been done by extending the local states of procedures. It might have been done differently. For instance, by introducing logical variables, $id_j$, that are initiated in each instance to the third component of the identifier of the corresponding LC. In either way, the existence of multiple instances can be postulated in the interference freedom specifications. However, whatever one does, such assumptions must be discharged by associating such instances with outstanding calls in other modules, for which one needs the communication histories. In the traditional compositional proof systems (see [Zwi89] for the state-of-the-art) these histories are directly accessible in the specification language, so that the connection between the coexistence of procedure instances and the communication histories can be expressed as axioms.

This paper’s proof system must introduce auxiliary variables first, before anything can be expressed about communication histories (or rather, about their encodings). The restricted form of module completeness is a direct consequence of this.

Acknowledgements

I thank Frank de Boer for discovering the invalidity of a previous version of the local merging lemma which pointed to the incompleteness of an earlier form of the proof system.

The first version of the paper has been troffed by Mrs. E van Thiel.

A Appendix: the substitution lemma

Here the substitution lemma 5-7 is proved.

A-1 Definition. Let $R$ be a binary relation on states, $R \subseteq St \times St$; $u, z, v \in Var$ and $e \in Tm(\Sigma) \setminus Var$. Then

- $R_u^z$ is defined by $\sigma\{u/v\} R \tau(\tilde{a}/u) \Leftrightarrow \sigma\{u/z\} R_u^z \tau(\tilde{a}/z)$
- $R_u^e$ is defined by $\sigma(\{e/u\} R \tau \Leftrightarrow \sigma R_u^e \tau$.

If $z \in Var$ and $t \in Tm(\Sigma)$, let $z := t$ stand for the relation $\{(\sigma, \sigma(\{t/y\})) \mid \sigma \in St\}$. 

38
Lemma. Use the same notation as in Definition A-1

If (a) \( \sigma R \tau \Rightarrow \sigma(z) = \tau(z) \) and (b) \( \sigma R \tau \Rightarrow \sigma(\beta/z) \) \( R \tau \) then
1. \( \sigma R^*_\tau \Leftrightarrow \sigma(\nu := e \circ \sigma) R (\circ \) is the usual composition of relations \( ) \)
2. \( \sigma R^*_\tau \Leftrightarrow \sigma(\nu := z \circ R \circ z := v) \tau(\tau(x)/v) \)

Proof. (1) is trivial, so concentrate on (2).

For a state \( \nu \), if \( \nu = \nu(\beta/z), \beta \neq y \) then \( \nu(\nu(x)/y) = \nu(\nu(x)/z) \). Hence, if \( \sigma = \sigma(\alpha/x) \) and \( \tau = \tau(\alpha/x) \), then \( \sigma R^*_\tau \) is equivalent with \( \sigma(\sigma(x)/v) \tau(\sigma(x)/v) \). By (a), \( \sigma(x) = \tau(x) \) and so by (b), \( \sigma R^*_\tau \Leftrightarrow \sigma(\sigma(x)/v) \tau(\tau(x)/v) \sigma(x)/z \). The result follows since \( \sigma(\sigma(x)/v) z := v(\tau) \) implies \( \tau(\tau(x)/v) \).

For the rest of the section, let \( S \) denote \( CM\epsilon(\Sigma)-\)statements not containing any call or wait statements. In other words, consider statements for which state-changes - involving a variable \( z \), say - can only be effected through assignments \( z := e \). For statements, \( S \), the usual partial correctness semantics \( \langle S \rangle \), is given by

\[
\{ (u, T) \mid u = U_0 U_{\ldots} U_n, T = T_0 \cdot T_n \}
\]

Facts:

- \( \langle \{x := e\} \rangle = x := e \) (i.e., as a relation)
- \( \langle S[x/v] \rangle = \langle S \rangle^e \) provided \( z \not\in \forall V(S) \)
- \( \langle S[e/u] \rangle = \langle S \rangle^u \) provided \( FV(e) \cap FV(S) = \emptyset \) and \( u \) does not appear as the left-hand-side of any assignment in \( S \).

These facts are sufficiently obvious so as not to warrant a proof. Note that the condition in the second fact implies assumptions (a) and (b) of Lemma A-2. Partial correctness “triples” in this paper’s formalism are notated as \( \{p\} S \{q\} \). In the rest of this section the (trivial) assumptions and commitments will be ignored and I shall write \( \{p\} S \{q\} \).

Lemma. Let \( u, v, x \in \text{Var}, e \in Tm(\Sigma) \) and \( p, q \in L(\Sigma) \) be such that \( x \not\in \forall V(S), FV(e) \cap \forall V(S) = \emptyset \) and \( u \) does not appear as the left-hand-side of an assignment in \( S \).

- \( A \models \{p\} S[z/v] \{q\} \Leftrightarrow A \models \{p\} x := v, S[z := v] \{q\} \), provided \( v \not\in \forall V(S) \)
- \( A \models \{p\} S[e/u] \{q\} \Leftrightarrow A \models \{p\} u := e, S \{q\} \).


The next step is the extension to multiple substitutions.

Lemma (\cite{deB80}). Let \( e, t \in Tm(\Sigma), \tau \in Tm(\Sigma) \cup L(\Sigma), x, y \in \text{Var} \) and suppose \( x \neq y \) and \( x \not\in \forall V(t) \). Then \( \tau[e/x][t/y] \equiv \tau[t/y][e[t/y]/x] \).

Corollary. With the same notations and conditions as in Lemma 5.3.4, if additionally \( y \not\in \forall V(e) \), then \( \tau[e/x][t/y] = \tau[t/y][e/x] \).

So under these conditions, simultaneous substitution acquires meaning. Obviously, these substitution-results hold for statements, too (provided the usual assumptions about non-variable substitutions hold). In the following (and in fact in the rest of the paper) simultaneous assignments, \( u_1, \ldots, u_n := e_1, \ldots, e_n \) or \( \bar{u} := \bar{e} \) (with \( u_i \neq u_j \) if \( i \neq j \)), will be used, with obvious meaning.

Now, Lemma 5-7 can be proven.

39
LEMMATA (The substitution lemma). Let $S, S'$ be $CM^e(\Sigma)$-statements not containing call or wait-statements. Let $u, v, \bar{z} \subseteq \text{Var}$, $\bar{e} \subseteq \text{TM}(\Sigma)$ and $p, q \in L(\Sigma)$ be such that $|u| = |\bar{e}|, |v| = |\bar{z}|$, the variables in $\bar{u}$ do not appear as left-hand-side of any assignments in $S$ and the variables in the lists $u, v$ and $f$ are pairwise disjoint. Additionally, assume that the following conditions hold:

$|u| \cap \bar{v} = \emptyset, FV(\bar{e}) \cap \bar{z} = \emptyset, \{\bar{u}, \bar{v}\} \cap FV(q) = \emptyset$ and $(FV(S') \cup \{\bar{u}, \bar{v}\}) \cap (FV(\bar{e}) \cup \bar{z}) = \emptyset$. Then

1. $A \models \{p\} S; S'\{q[\cdot]\}, S; \bar{u}; \bar{v} := \bar{e}, \bar{z}; S'; \bar{u} := \emptyset, \bar{v}; S\{q\}$. Provided $\bar{z} \cap FV(p) = \emptyset$.

2. $A \models \{p; \bar{u} := e\} S'/f := \bar{e}; S; \bar{u} := \emptyset, \bar{v}; S\{q\}$. Provided $\bar{z} \cap FV(p) = \emptyset$.

**Proof.** Define $WP(S, p) = \{\sigma | (\sigma, \tau) \in \langle S \rangle \Rightarrow \tau \models p\}$ and $SP(S, p) = \{\sigma | (\sigma, \tau) \in \langle S \rangle \& \sigma \models p\}$. Extend the signature $I: \Sigma = I: \cup \{sp(S, p), wp(S, p) | \bar{S} \models \text{a } CM^e\text{-statement}, p \in L(\Sigma)\}$ and let $A$ be the extension of $A$ in which the new predicate symbols get their intended meaning:

$sp(S, p) = SP(S, p)$ and $wp(S, p) = WP(S, p)$.

(1) By Lemma 5.3.3 (and corollary 5.3.5): $A \models \{sp(S, p)\} u, v := \bar{e}, \bar{z}; S'; \bar{e} := v\{q[\cdot]\}$. Denote $sp(S, p)$ by $p$. From [deB80] obtain that $A \models wp(z := e, p) \leftrightarrow p[e/z]$ and hence equivalence with

$A \models \{p; \bar{u} := e\} S'/f := \bar{e}; S; \bar{u} := \emptyset, \bar{v}; S\{q\}$. Since the variables in $\bar{u}$ are never assigned to and $FV(S') \cup \{\bar{u}, \bar{v}\} \cap FV(\bar{e}) = \emptyset$, I have

$A \models \{true\} u, v := \bar{e}, \bar{z}; S'; \bar{e} := \emptyset, \bar{v}; S\{q\}$. This implies equivalence of the left-hand side of (A) with $A \models \{p; \bar{u} := e\} S'/f := \bar{e}; S; \bar{u} := \emptyset, \bar{v}; S\{q\}$.

(2) Since I can assume that $\bar{e} \cap wp(S, q) = \emptyset$, Lemma 5.3.3 yields equivalence with $A \models \{p[\cdot]\} u, v := \bar{e}, \bar{z}; S'; \bar{v} := \emptyset\{wp(S, q)\}$. Denote $wp(S, q)$ by $\bar{q}$. Again, [deB80] learns that $A \models sp(z := e, p) \leftrightarrow \exists y(p[y/z] \wedge z = e[y/z]), y \notin FV(z := e, p)$.

As $FV(p[\cdot]) \cap \{\bar{u}, \bar{v}\} = \emptyset$, the left-hand side of (B) is equivalent with $A \models \{p \wedge \bar{u} := \bar{e}\} S'/\{\bar{v}[\bar{e}/\bar{z}]\}$. Next observe that $(FV(p, \bar{e}, S', \bar{q}[\bar{v}/\bar{z}] \cup \bar{u}) \cap \bar{z} = \emptyset$. This implies equivalence with $A \models \{p \wedge \bar{u} := \bar{e}\} S'; \bar{e} := \emptyset; \bar{q}\}$. Provided $\bar{z} \cap FV(p) = \emptyset$.

References


