A Proof Theory of Asynchronously Communicating Sequential Processes

F.S. de Boer, N. Francez, M. van Hulst, and F.A. Stomp

UU-CS-1996-05
January 1996

Universiteit Utrecht

ISSN: 0924–3275

A proof theory of asynchronously communicating sequential processes*

F. S. de Boer[†] N. Francez[‡] M. van Hulst[†] F. A. Stomp[§]

Abstract

We present compositional Hoare logics for distributed systems in which communication is asynchronous via FIFO channels. The logics are proved sound and relative complete.

One of our main results is that the non-compositional proof method for (synchronous) CSP of Apt, Francez, and de Roever applied to asynchronous communication allows a compositional formulation in which the so-called cooperation test can be fully incorporated in the local verification of the (sequential) components of the system. Thus, the resulting method enables (verified) local specifications of the components, which may be augmented with auxiliary variables, to be combined into a global specification without any test involving the way those local specifications have been derived.

We argue that the approach presented in the current paper is preferable over other compositional proof methods presented in the literature in that it allows more flexibility in correctness proofs. The applicability of our approach is demonstrated by a correctness proof of a distributed program for leader election.

*This paper is the culmination of two independent developments, one by Francez and Stomp (preliminary versions [11, 12]) and the other by de Boer and van Hulst (preliminary versions [4, 6, 7]). As the two approaches had much in common, they were joined together upon the advice of the editor of TCS, in sequel to the submission of the first for publication to TCS. The research of the authors F.S. de Boer and M. van Hulst has been partially supported by the Human Captial and Mobility Network 'EXPRESS'.

[†]Utrecht University, Department of Computer Science, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.

[‡]The Technion –Israel Institute of Technology, Department of Computer Science, Haifa 32000, Israel.

[§]AT&T, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, USA.

1 Introduction

Hoare logics have been developed for and applied successfully to a variety of sequential programming constructs (see for an overview [2]). For the parallel programming language CSP a Hoare logic is presented in [3]. The proof method in [3] is based on the proof-theoretical ideas first introduced in [15] for a parallel programming language based on communication via shared variables. Characteristic of this method is its two-leveled structure: In the first level the (sequential) components of a system are verified using *assumptions* about the communication statements; the second level consists of checking whether the assumptions introduced in the first level for the various components are mutually consistent. This test is called the *cooperation test*. Since this test involves information of the internal structure of the local proofs of the correctness of the sequential components, this proof method is not compositional.

The results

In this paper we develop *compositional* Hoare logics for distributed systems composed of processes which communicate asynchronously via FIFO (First In First Out) channels.

One of our main results is that the non-compositional proof method of [3] applied to ACSP (Asynchronously Communicating Sequential Processes) allows a formulation in which the cooperation test can be fully incorporated in the verification of the (sequential) components of a system. As a consequence the resulting proof method mainly consists of the verification of the parallel components in a slight extension of the usual Hoare logic for sequential programs. The verified local specifications then can be logically combined into a global specification, without any additional test involving the way these local specifications are derived (as usually codified in proof outlines). As such the resulting proof method is compositional. Moreover, we prove that the method is sound and (relative) complete. The applicability of the proof method is demonstrated by proving correctness of a distributed leader election algorithm.

To obtain a (relative) complete proof method for ACSP *auxiliary variables* are needed to describe and reason about dynamic control properties. These variables do not affect the control of the original program. We show that when restricting sequential processes to *deterministic* control structures, these variables are no longer needed.

Comparison with related work

The first proof method for ACSP was proposed in [19] and [20]. That method is an adaptation of the system presented in [14] for synchronously CSP. These systems are not compositional because of the so-called tests for *satisfaction* and *interference-freedom*. A proof method in the same spirit has been proposed by Camp, Kearns, and Ahuja [9] to reason about communication though *flush-channels*, which generalize conventional asynchronous channels.

A compositional Hoare logic for CSP has been developed in [22]. This Hoare logic has been applied to ACSP in [17]. Other similar compositional proof systems for reasoning about asynchronous communication have been presented in [13, 21, 18]. Francez [10] gives on overview of many existing proof methods for sequential and parallel/distributed programs. Compositionality in the above-mentioned systems is achieved by the incorporation of certain logical variables which range over *histories*, i.e. sequences of communication events. In [4] a completeness proof for the method of [3] (for CSP) is given which is based on a compositional semantics, and which, as a consequence, encodes in a precise manner the compositional proof method of [22]. Similarly we show how to "extract" the compositional proof method of [17] from the completeness proof of our proof method for ACSP. The completeness proof thus shows that the compositional reasoning pattern based on histories can be mimicked in our proof method. However in general histories complicate the reasoning process while in many real-life examples simple boolean "flags" as auxiliary variables already suffice. Therefore compositional proof methods, which allow auxiliary variables of any

kind, like the one presented in this paper, are more flexible and thus more useful in practice.

The plan of the paper

In the next section we introduce our programming language of asynchronously communicating sequential processes. We present our proof system for reasoning about partial correctness properties of programs in Section 3. Soundness and (relative) completeness of this proof system is proved in the Sections 4 and 5 respectively. In Section 6, we prove correctness of a distributed leader election program. A proof system for reasoning about non-terminating programs is presented in Section 7. As shown in Section 8 the proof system can be simplified when we restrict ourselves to so-called deterministic processes. Finally, Section 9 draws some conclusions.

2 Programming language

In this section we define the syntax and the semantics of the programming language used throughout this paper. The semantics describes the behaviour of asynchronously communicating processes. Processes interact only via directed communication channels which are implemented by unbounded FIFO-buffers. A process can send a value along a channel or it can receive a value along a channel. The value sent along a channel is appended to the buffer which implements that channel, whereas receiving a value along a channel consists of retrieving the first element from its corresponding buffer. Thus values are received along a channel in the same order as they have been sent along that channel. A process is suspended when it tries to receive a value along an empty channel. Since buffers are assumed to be unbounded sending values can always take place.

2.1 Syntax

We assume given a set *Pvar* of (program) variables, and a set C of channel names with typical element $c, \ldots$. The sets *Pvar* and C are assumed to be disjoint.

The syntax of a statement, which can be executed by a (sequential) process, is defined by the following grammar:

$$
\begin{array}{rcl}
S & ::= & \mathsf{skip} \\
& | & x := e \\
& | & c!!e \\
& | & c??x \\
& | & S_1 ; S_2 \\
& | & [\!]_{i \in A} [g_i \rightarrow S_i] \\
& | & \star[\!]_{i \in A} [g_i \rightarrow S_i]
\end{array}
$$

In the above, e denotes an expression, and A denotes some fixed and finite set (of indices.) (The syntactic structures of expressions and of indices is left unspecified.)

The statements skip and $x := e$ have their usual meaning. Sending the value of expression e along channel c is described by the *send-statement* $c!!e$, whereas receiving a value along channel c and

recording that value in variable x is described by the *receive-statement* $c??x$. We sometimes refer to a receive- or a send-statement as an IO statement. Sequential composition is denoted by ";". The statement $[]_{i\in A}[g_i \to S_i]$ is called a *guarded conditional.* It consists of a (finite number of) guarded statements $g_i \to S_i$ $(i \in A)$. Every *guard* is either a boolean expression b or a sequential composition $b; c??x$, for some boolean condition b and some receive-statement $c??x$. (The syntactic structure of boolean expressions is left unspecified.) Guard b is enabled in some state, if it evaluates to true in that state. If enabled, execution of such a guard is equivalent to the execution of skip. Guard $b; c??x$ is enabled in some state, if in that state b evaluates to true and channel c is not empty. If enabled, execution of such a guard is equivalent to the execution of $c??x$. The execution of a guarded conditional $[]_{i\in A}[g_i \to S_i]$ consists of the execution of some enabled guard g_i and the execution of S_i thereafter. If there exists no enabled guard, then the execution of the guarded conditional suspends. The *guarded iteration* $\star[]_{i\in A}[g_i \to S_i]$ consists of repeatedly executing the guarded selection $[]_{i\in A}[g_i \to S_i]$ until all the boolean parts of the guards are false (and the iteration terminates). Note that we do not have send-statements in guards. However this does not reduce the expressive power as in CSP ([8]) because the execution of a send-statement does not depend on the environment.

In order to reason about programs, we expand them by (assignments to) *auxiliary* variables. These variables are not allowed to affect the flow of control of the original program. They are used only to reason about dynamic control properties. Related with the notion of auxiliary variables is that of a *bracketed section.* Bracketed sections are used to render several (typically two) sequentially composed statements *atomic*, so that it is irrelevant whether an *invariant* holds in between.

Definition 2.1 The syntax of a bracketed statement S which describes the behaviour of a sequential process, is defined by

$$
\begin{array}{rcl}
S & ::= & \mathsf{skip} \\
& | & x := e \\
& | & \langle c!!e; y := e' \rangle \\
& | & \langle c??x; y := e \rangle \\
& | & S_1; S_2 \\
& | & []_{i\in A}[bg_i \to S_i] \\
& | & \star[]_{i\in A}[bg_i \to S_i]
\end{array}
$$

Here bg_i is either a simple boolean guard as before, or it is a bracketed guard of the form $\langle b_i; c??x_i; y_i := e_i \rangle$. The symbols "$\langle$" and "$\rangle$" mark the region of a bracketed section, which typically consists of an IO statement and an assignment (in case of a guarded conditional or iteration the boolean guard is also included). The auxiliary variables of a bracketed statement are those variables which occur at the lefthandside of an assignment in a bracketed section. We require that the auxiliary variables do not occur outside bracketed sections and that they do not occur in the boolean expressions.

Non-bracketed IO-statements can be viewed as special cases of bracketed statements. For example, $c??x$ can be identified with $\langle c??x; y := y \rangle$, for some auxiliary variable y. Hence we will only consider statements in the sequel in which every receive-statement and every send-statement is contained within a bracketed section.

Definition 2.2 A parallel program is of the form $[S_1 \parallel ... \parallel S_n]$ with the following restrictions: The statements S_i do not share program variables, channels are unidirectional and connect exactly one sender and one receiver. Formally we assume a partitioning of the set $Pvar$ of program variables into sets $Pvar_i$ (here i ranges over the natural numbers.) Also, we assume a partitioning of the set $IOvar = \{c??, c!! \mid c \in C\}$ of *input/output* variables into sets $IOvar_i$. Let $Pvar(S)$ be the set of program variables of S and let $IOvar(S)$ be the set of those input/output variables for which there occurs a corresponding input/output statement in S, e.g. $IOvar(c??x) = \{c??\}$. We then impose on any program $P = [S_1 \parallel ... \parallel S_n]$ the syntactic restriction that $Pvar(S_i) \subseteq PVar_i$ and $IOvar(S_i) \subseteq IOvar_i$.

2.2 Semantics

We first define the notion of a state:

Definition 2.3 A state assigns values to all program variables and all input/output variables. Define

$$\Sigma = (Pvar \cup IOvar) \to (Val \cup Val^*),$$

where Val denotes some domain of values and Val^* denotes the set of all finite sequences over Val. We impose the restriction that program variables are mapped to values; and that input/output variables are mapped to sequences of values. Elements of Σ are called states and denoted by $\sigma, \ldots$.

In the sequel $\sigma(e)$ and $\sigma(b)$ denote the value of the expressions e and b in state σ, respectively. In particular, $\sigma(x)$, for $x \in Pvar$, denotes the value of the program variable x in σ, and similarly, $\sigma(c??)$ and $\sigma(c!!)$, $c \in C$, denote the value of the input/output variables $c??$ and $c!!$ in σ. The idea is that $\sigma(c??)$ denotes the sequence of values received along channel c, whereas $\sigma(c!!)$ denotes the sequence of values sent along channel c. The state which results from assigning the value v to the program variable x in σ is denoted by $\sigma\{v/x\}$. Similarly, the state which results from assigning the sequence s of values to the input/output variables $c??$, $c!!$ is denoted by $\sigma\{s/c??\}$ and $\sigma\{s/c!!\}$, respectively. As usual, we denote by $\sigma\{v_1/y_1, \ldots, v_n/y_n\}$ the result of the simultaneous assignment of v_i to y_i in σ $(i = 1, \ldots, n)$.

We will use the following operations on sequences: $f(s)$, which is only defined if sequence s is not empty, denotes the first element of s; $s \cdot v$ denotes the result of appending the value v to s (similarly, $v \cdot s$ denotes the result of prefixing v to s); and $s - s'$, which is only defined when s' is a prefix of s, denotes the suffix of s determined by s'. The empty sequence is denoted by ϵ. For technical convenience we introduce $\sigma(c)$ as an abbreviation of $\sigma(c!!) - \sigma(c??)$, i.e. the values sent, but not yet received along channel c.

To formally define the semantics of our programming language, we introduce a fictitious statement E to denote termination.

Definition 2.4 The relation $\Longrightarrow \subseteq Stat \times \Sigma \times \Sigma \times Stat$ is defined as the smallest relation satisfying the following, where we denote $(S, \sigma, \sigma', S') \in \Longrightarrow$ by $S \overset{\langle \sigma, \sigma' \rangle}{\Longrightarrow} S'$:

1. $\mathsf{skip} \overset{\langle \sigma, \sigma \rangle}{\Longrightarrow} E$
2. $x := e \overset{\langle \sigma, \sigma' \rangle}{\Longrightarrow} E$,
   where $\sigma' = \sigma\{\sigma(e)/x\}$.

3. $\langle c!!e; y := e' \rangle \stackrel{\langle \sigma, \sigma' \rangle}{\Longrightarrow} E$,
   where $\sigma' = \sigma\{\sigma(c!!) \cdot \sigma(e)/c!!, \sigma(e')/y\}$.

4. $\langle c??x; y := e \rangle \stackrel{\langle \sigma, \sigma' \rangle}{\Longrightarrow} E$,
   provided $\sigma(c) \neq \epsilon$. Here $\sigma' = \sigma\{\sigma(c??) \cdot v/c??, v/x, \sigma(e)/y\}$, where $v = f(\sigma(c))$.

5. If $S_1 \stackrel{\langle \sigma, \sigma' \rangle}{\Longrightarrow} S'$ then $S_1; S_2 \stackrel{\langle \sigma, \sigma' \rangle}{\Longrightarrow} S'; S_2$.
   We identify $E; S$ with S.

6. $[\!]_{i \in A}[bg_i \rightarrow S_i] \stackrel{\langle \sigma, \sigma' \rangle}{\Longrightarrow} S_i$,
   provided $bg_i = b_i$, $\sigma(b_i) = \mathit{true}$ and $\sigma = \sigma'$; or $bg_i = \langle b_i; c_i??x_i; y_i := e_i \rangle$, $\sigma(b_i) = \mathit{true}$, $\sigma(c_i) \neq \epsilon$, and $\sigma' = \sigma\{\sigma(c_i??) \cdot v/c_i??, v/x_i, \sigma(e_i)/y_i\}$, where $v = f(\sigma(c_i))$.

7. $\star[\!]_{i \in A}[bg_i \rightarrow S_i] \stackrel{\langle \sigma, \sigma' \rangle}{\Longrightarrow} S_i; \star[\!]_{i \in I}[bg_i \rightarrow S_i]$,
   provided $bg_i = b_i$, $\sigma(b_i) = \mathit{true}$ and $\sigma = \sigma'$; or $bg_i = \langle b_i; c_i??x_i; y_i := e_i \rangle$, $\sigma(b_i) = \mathit{true}$, $\sigma(c_i) \neq \epsilon$, and $\sigma' = \sigma\{\sigma(c_i??) \cdot v/c_i??, v/x_i, \sigma(e_i)/y_i\}$, where $v = f(\sigma(c_i))$.

8. $\star[\!]_{i \in A}[bg_i \rightarrow S_i] \stackrel{\langle \sigma, \sigma \rangle}{\Longrightarrow} E$,
   provided for the boolean part b_i of bg_i, $\sigma(b_i) = \mathit{false}$ for all $i \in A$.

We define the semantics of a statement S as the set of (finite) sequences, so-called *reactive* sequences, of elements of $\Sigma \times \Sigma$ generated by the above transition relation (such sequences are introduced in [5] as a general model for asynchronous communication).

Definition 2.5 For $w = \langle \sigma_1, \sigma_1' \rangle \cdots \langle \sigma_n, \sigma_n' \rangle \in (\Sigma \times \Sigma)^*$, $S \stackrel{w}{\Longrightarrow} E$ holds if there exist statements $S_1, \ldots, S_n$ such that $S_1 = S$, $S_n = E$ and, for $1 \leq i < n$, $S_i \stackrel{\langle \sigma_i, \sigma_i' \rangle}{\Longrightarrow} S_{i+1}$. For any statement S with $Pvar(S) \subseteq Pvar_m$ and $IOvar(S) \subseteq IOvar_m$, the semantics $\mathcal{M}(S) \subseteq (\Sigma \times \Sigma)^*$ is defined by

$$\mathcal{M}(S) = \{w \mid S \stackrel{w}{\Longrightarrow} E \text{ and } w \text{ is } m\text{-connected}\}$$

where sequence $\langle \sigma_1, \sigma_1' \rangle \cdots \langle \sigma_n, \sigma_n' \rangle$ is m-connected if for all $1 \leq i < n$, σ_{i+1} and σ_i' agree on all the variables in $Pvar_m$ and $IOvar_m$. By $\mathcal{M}(S)(\sigma)$ we denote the set of sequences $\langle \sigma_1, \sigma_1' \rangle \cdots \langle \sigma_n, \sigma_n' \rangle$ with $\sigma = \sigma_1$. As a special case we define $\mathcal{M}(E) = \{\epsilon\}$.

A sequence in set $\mathcal{M}(S)$ represents a computation of S in some environment. The environmental steps in a sequence $\langle \sigma_1, \sigma_1' \rangle \cdots \langle \sigma_n, \sigma_n' \rangle$ are represented by the "gaps" $\langle \sigma_i', \sigma_{i+1} \rangle$. The requirement of m-connectedness guarantees that the environment does not affect the (program and input/output) variables controlled by S.

Definition 2.6 For any program $P = [S_1 \parallel \ldots \parallel S_n]$, the semantics $\mathcal{M}(P) \subseteq \Sigma \times \Sigma$ of P is defined by

$$\mathcal{M}(P) = \alpha_P(\mathcal{M}(S_1) \parallel \ldots \parallel \mathcal{M}(S_n)),$$

where the abstraction operator α_P returns the set of pairs $\langle \sigma, \sigma' \rangle$ of initial and final states of all *connected* sequences, such that for all channels c in the program $\sigma(c??) \leq \sigma(c!!)$ holds. An element $\langle \sigma_1, \sigma_1' \rangle \cdots \langle \sigma_n, \sigma_n' \rangle$ of $(\Sigma \times \Sigma)^*$ is called connected if $\sigma_i' = \sigma_{i+1}$, for $1 \leq i < n$. The semantic operator $\parallel$ denotes the usual operation of interleaving. By $\mathcal{M}(P)(\sigma)$ we denote the set of states σ' such that $\langle \sigma, \sigma' \rangle \in \mathcal{M}(P)$. (Thus, if $\sigma(c??)$ is not a prefix of $\sigma(c!!)$ for some channel c in program P, then $M(P)(\sigma) = \emptyset$ holds.)

3 Proof system

In this section we present axioms and proof rules for formally reasoning about programs.

3.1 Correctness formulas

We assume given some logic to reason about properties of states. The vocabulary of the logic includes the set of program variables and input/output variables, and function/predicate symbols describing operations/relations on the domain *Val* of values and the domain Val^* of sequences (of values). Assertions of the underlying logic are denoted by $p, q, \ldots$. By $\sigma \models p$ we denote that assertion p holds in σ. Validity of an assertion p is denoted by $\models p$.

Local correctness formulas are of the form

$$I : \{p\}S\{q\},$$

for assertions p, q, I and statement S. Here, I is an assertion which describes certain invariant properties of computations of S starting in a state satisfying p. W.r.t. a local correctness formula $I : \{p\}S\{q\}$, it is implicitly assumed that $PVar(S,p,q) \subseteq Pvar_n$ and $IOvar(S,p,q) \subseteq IOvar_n$, for some n. Invariant I may refer to arbitrary input/output variables, but is allowed to refer only to those program variables of S which occur as auxiliary variables. Informally, $I : \{p\}S\{q\}$ is valid iff

> every terminating computation of S in a parallel environment, which guarantees I at each interleaving point, preserves I and results in a state satisfying postcondition q, if initially p holds.

Global correctness formulas are of the form

$$\{p\}P\{q\},$$

for assertions p, q and program P. The correctness formula $\{p\}P\{q\}$ holds if every terminating computation of P in a state satisfying precondition p results in a state satisfying postcondition q.

We now turn to the formal semantics of correctness formulas:

Definition 3.1 Let $\mathcal{M}_{\leq}(S)$ denote the prefix closure of $\mathcal{M}(S)$, i.e. it consists of $M(S)$ plus all the prefixes of sequences of $M(S)$. We define $\models I : \{p\}S\{q\}$ iff for every sequence $w = \langle\sigma_1, \sigma_1'\rangle \cdots \langle\sigma_n, \sigma_n'\rangle \in \mathcal{M}_{\leq}(S)$, such that $\sigma_1 \models p$ and $\sigma_i \models I$, for $1 \leq i \leq n$, we have that $\sigma_i' \models I$, for $1 \leq i \leq n$; and if $w \in M(S)$ then $\sigma_n' \models q$.

Invariant I in local correctness formula $I : \{p\}S\{q\}$ thus provides for a kind of *assumption/commitment* style specification.

Definition 3.2 For global correctness formula $\{p\}P\{q\}$, $\models \{p\}P\{q\}$ holds iff for all $\langle\sigma, \sigma'\rangle \in \mathcal{M}(P)$ with $\sigma \models p$ we have that $\sigma' \models q$ is satisfied.

This is the standard partial correctness interpretation of correctness formulas in Hoare logics.

3.2 Axioms and rules

Next we discuss the axioms and rules of the proof system. We have the following axioms for the skip statement and assignments:

Axiom 1 *(skip)* $I : \{p\}\mathsf{skip}\{p\}$

Axiom 2 *(assignment)* $I : \{p[e/x]\}x := e\{q\}$

Here $p[e/x]$ denotes the result of substituting all free occurrences of x in p by e. Hereafter, simultaneous substitution of $e_1, \ldots, e_n$ for all free variables $x_1, \ldots, x_n$ in p will be denoted by $p[e_1/x_1, \ldots, e_n/x_n]$. Since the invariant I does not refer to program variables (except for the auxiliary variables, but these occur only in bracketed sections for which we have different rules, see below), I is on purely syntactic grounds preserved by the skip statement and assignments.

The following axiomatization of the input/output statements closely mirrors their semantics. (The sequence term c stands for the term $c!! - c??$, i.e. it represents the contents of the channel c. The term $f(c)$ denotes the first element of c.)

Rule 1 *(output)* $\dfrac{(I \wedge p) \to (I \wedge q)[c!! \cdot e/c!!, e'/x]}{I : \{p\}\langle c!!e; x := e'\rangle\{q\}}$

Rule 2 *(input)* $\dfrac{(I \wedge p \wedge c \neq \epsilon) \to (I \wedge q)[f(c)/x, c?? \cdot f(c)/c??, e/y]}{I : \{p\}\langle c??x; y := e\rangle\{q\}}$

The following rules for sequential composition, the guarded statement and the guarded iteration are the usual ones, except for the addition of invariant I.

Rule 3 *(sequential composition)* $\dfrac{I : \{p\}S_1\{r\},\ I : \{r\}S_2\{q\}}{I : \{p\}S_1; S_2\{q\}}$

In the following rule we assume that $bg_i = b_i$ for $i \in A_1$; and $bg_i = \langle b_i; c_i??x_i; y_i := e_i\rangle$ for $i \in A - A_1$.

Rule 4 *(guarded conditional)*
$$\frac{I : \{p \wedge b_i\}S_i\{q\}\ (i \in A_1),\ I : \{p \wedge b_i\}\langle c_i??x_i; y_i := e_i\rangle; S_i\{q\}\ (i \in A - A_1)}{I : \{p\}[]_{i\in A}[bg_i \to S_i]\{q\}}$$

Rule 5 *(guarded iteration)* $\dfrac{I : \{p\}[]_{i\in A}[bg_i \to S_i]\{p\}}{I : \{p\} \star []_{i\in A}[bg_i \to S_i]\{p \wedge \bigwedge_{i\in A} \neg b_i\}}$

Moreover we have the following consequence rule.

Rule 6 *(Local consequence rule)* $\dfrac{(I \wedge p) \to p',\ I : \{p'\}S\{q'\},\ (I \wedge q') \to q}{I : \{p\}S\{q\}}$

Note that the assumption/commitment style of specification inherent in the interpretation of the invariant does not allow a weakening of the invariant in the above consequence rule.

We have the following rule for parallel composition:

Rule 7 *(parallel composition)* $\dfrac{I : \{p_i\}S_i\{q_i\}\ i = 1, ..., n}{\{I \wedge \bigwedge_{i=1}^{n} p_i\}[S_1 \parallel ... \parallel S_n]\{I \wedge \bigwedge_{i=1}^{n} q_i\}}$

We conclude the exposition of the proof system with the global consequence rule, the auxiliary variables rule, the FIFO rule and the elimination rule.

Rule 8 *(global consequence rule)* $\dfrac{p \rightarrow p',\ \{p'\}P\{q'\},\ q' \rightarrow q}{\{p\}P\{q\}}$

Rule 9 *(auxiliary variables rule)* $\dfrac{\{p\}P'\{q\}}{\{p\}P\{q\}}$

where P is obtained from P' by deleting all assignments to the auxiliary variables. It is assumed that the postcondition q does not refer to the auxiliary variables of P'.

Rule 10 *(FIFO rule)*

$$\frac{\{p \wedge c?? \leq c!!\}P\{q\}}{\{p\}P\{q\}}$$

where channel c occurs in program P, and "$\leq$" denotes the prefix relation on sequences.

Rule 11 *(elimination rule)* $\dfrac{\{p\}P\{q\}}{\{\exists x.p\}P\{q\}}$

provided x is a variable not occurring in P or q.

By $\vdash I : \{p\}S\{q\}$ and $\vdash \{p\}P\{q\}$ we denote the derivability of the correctness formulas $I : \{p\}S\{q\}$ and $\{p\}P\{q\}$, respectively, in the above proof system, using as additional axioms all valid assertions of the underlying logic.

4 Soundness

In this section we show validity of every derivable correctness formula.

We start with local correctness formulas:

Theorem 4.1 *For every local correctness formula $I : \{p\}S\{q\}$ we have that*

$$\vdash I : \{p\}S\{q\} \text{ implies } \models I : \{p\}S\{q\}.$$

Proof By straightforward induction on the length of the derivation. The basis of induction (for axioms) should be obvious. For the induction step, we treat the following cases:

$S = \langle c??x; y := e\rangle$: Suppose that the last step in the proof had been the application of the input rule. We are given that $(I \wedge p \wedge c \neq \epsilon) \rightarrow (I \wedge q)[f(c)/x, c?? \cdot f(c)/c??, e/y]$ holds. Let $\langle \sigma, \sigma' \rangle \in M(S)$ be such that $\sigma \models p$ and $\sigma \models I$. By definition of the transition relation

it follows that $\sigma(c) \neq \epsilon$ and $\sigma' = \sigma\{\sigma(c??) \cdot v/c??, v/x, \sigma(e)/y\}$ with $v = f(\sigma(c))$. So we have that $\sigma \models I \wedge p \wedge c \neq \epsilon$, and thus by the validity of the above implication, $\sigma \models (I \wedge q)[f(c)/x, c?? \cdot f(c)/c??, e/y]$. The latter is equivalent to $\sigma' \models I \wedge q$ (assuming the substitution lemma of the underlying logic), which had to be proved.

$S = S_1; S_2$: Suppose that the last step in the proof had been the application of the sequential composition rule. Then, $\vdash I : \{p\}S_1\{r\}$ and $\vdash I : \{r\}S_2\{q\}$ both hold, for some assertion r. ¿From the induction hypothesis, we obtain that $\models I : \{p\}S_1\{r\}$ and $\models I : \{r\}S_2\{q\}$ are both satisfied. Choose $w = \langle\sigma_1, \sigma_1'\rangle \cdots \langle\sigma_n, \sigma_n'\rangle \in M_{\leq}(S_1; S_2)$ such that $\sigma_1 \models p$ and $\sigma_i \models I$, for $1 \leq i \leq n$. Then either $w \in M_{\leq}(S_1)$ and so, by the validity of $I : \{p\}S_1\{r\}$, $\sigma_i' \models I$, for $1 \leq i \leq n$; or $w = w_1 w_2$, with $w_1 \in M(S_1)$ and $w_2 \in M_{\leq}(S_2)$. Let k be the length of w_1. By the validity of $I : \{p\}S_1\{r\}$ we have that $\sigma_i' \models I$, for $1 \leq i \leq k$, and $\sigma_k' \models r$. By the syntactic restrictions on local correctness formulas and the definition of the semantics $\mathcal{M}$ we have that σ_k' and σ_{k+1} agree on all the program variables and the input/output variables of r. So we have that $\sigma_{k+1} \models r$. The validity of $I : \{r\}S_2\{q\}$ then gives us $\sigma_i' \models I$, for $k < i \leq n$, and if $w_2 \in M(S_2)$ then $\sigma_n' \models q$.

□

Next we show the validity of every derivable global correctness formula.

Theorem 4.2 *For every global correctness formula $\{p\}P\{q\}$ we have that*

$\vdash \{p\}P\{q\}$ implies $\models \{p\}P\{q\}$.

Proof By induction on the length of the derivation. We concentrate on application of the parallel composition rule. (The other rules are standard.)

Let $P = [S_1 \parallel ... \parallel S_n]$. Assume that the last step in the proof of $\{p\}P\{q\}$ has been the application of the parallel composition rule. So $\{p\}P\{q\}$ equals $\{I \wedge \bigwedge_{i=1}^n p_i\}[S_1 \parallel ... \parallel S_n]\{I \wedge \bigwedge_{i=1}^n q_i\}$, for some assertions $p_1, \ldots, p_n, q_1, \ldots, q_n$ and I such that $\vdash I : \{p_i\}S_i\{q_i\}$, $1 \leq i \leq n$. By the soundness of the local proof system we derive that $\models I : \{p_i\}S_i\{q_i\}$, $1 \leq i \leq n$, from which we have to prove that $\models \{I \wedge \bigwedge_{i=1}^n p_i\}[S_1 \parallel ... \parallel S_n]\{I \wedge \bigwedge_{i=1}^n q_i\}$ holds. Choose states σ, σ' such that $\sigma \models I \wedge \bigwedge_{i=1}^n p_i$ and $\langle\sigma, \sigma'\rangle \in \mathcal{M}(S_1 \parallel ... \parallel S_n)$. We have to show $\sigma' \models I \wedge \bigwedge_{i=1}^n q_i$. By the semantics of $[S_1 \parallel ... \parallel S_n]$ there exist $w_i \in \mathcal{M}(S_i)$, for $i = 1, ..., n$, and a (connected) sequence $w = \langle\sigma_1, \sigma_2\rangle ... \langle\sigma_{n-1}, \sigma_n\rangle$ such that $\sigma_1 = \sigma, \sigma_n = \sigma'$ and w is an interleaving of $w_1, ..., w_n$. Since $\sigma \models p_i$ and the computation steps of the other components do not affect the variables of p_i we have that p_i holds also in the initial state of w_i. If, for example, the first computation step of w, $\langle\sigma_1, \sigma_2\rangle$, is executed by S_j, $j \neq i$, then, since p_i does not refer to the (program and input/output) variables of S_j, we have that $\sigma_2 \models p_i$. Now let w' be a prefix of w such that I holds in every state of w', say $w' = \langle\sigma_1, \sigma_2\rangle \cdots \langle\sigma_{k-1}, \sigma_k\rangle$. Let $\langle\sigma_k, \sigma_{k+1}\rangle$ be a computation step of S_i and w_i' be the prefix of w_i corresponding to the computation steps of S_i in $w' \cdot \langle\sigma_k, \sigma_{k+1}\rangle$. It then follows by the validity of $\models I : \{p_i\}S_i\{q_i\}$ that $\sigma_{k+1} \models I$. Thus by induction we have that $\sigma' \models I$. It remains to show that $\sigma' \models \bigwedge_{i=1}^n q_i$. Take any of the q_i. From the validity of $\models I : \{p_i\}S_i\{q_i\}$, and the fact that I holds in all states of w_i (as shown above), it then follows that q_i holds in the last state of w_i. Because q_i only refers to the variables of $Pvar_i$ and $IOvar_i$, which are not affected by the subsequent (if any) computation steps of the other components, we infer that q_i holds in the last state of w, i.e. $\sigma' \models q_i$. Because this holds for any q_i we conclude $\sigma' \models \bigwedge_{i=1}^n q_i$, which completes the proof. □

5 Completeness

We prove (relative) completeness of our proof system in the sense that all valid *global* correctness formulas about unbracketed programs can be derived using the axioms and rules. That is, for every P, p and q, with P unbracketed: $\models \{p\}P\{q\}$ implies $\vdash \{p\}P\{q\}$. Let us thus fix some unbracketed program $P = [S_1 \parallel ... \parallel S_n]$. We extend P so that every input and output statement in S_i is put in a bracketed section together with a corresponding assignment to a fresh auxiliary variable h_i. This variable h_i ranges over sequences of the *names* of the input/output variables local to S_i. In order to distinguish in the underlying logic between the input/output variables $c??$ and $c!!$, which denote sequences of values, and their "names", we introduce $\underline{c}??, \underline{c}!!$ to denote the names of the variables $c??, c!!$. Formally, we define the extension of S_i' by induction on S_i:

$$
\begin{array}{lcl}
(\mathsf{skip})' & = & \mathsf{skip} \\
(x := e)' & = & x := e \\
(c!!e)' & = & \langle c!!e; h_i := h_i \cdot \underline{c}!! \rangle \\
(c??x)' & = & \langle c??x; h_i := h_i \cdot \underline{c}?? \rangle \\
(S_1; S_2)' & = & S_1'; S_2' \\
(\llbracket_{i\in A}[g_i \to S_i])' & = & \llbracket_{i\in A}[bg_i \to S_i'] \\
(\star\llbracket_{i\in A}[g_i \to S_i])' & = & \star\llbracket_{i\in A}[bg_i \to S_i'],
\end{array}
$$

where bg_i is the same as g_i if g_i is a pure boolean guard, and where bg_i is the same as $\langle b_i; c_i??; h_i := h_i \cdot \underline{c_i}?? \rangle$ if g_i is $b_i; c_i??$. In the above, the assignment $h_i := h_i \cdot \underline{c_i}??$ consists of appending the name $\underline{c_i}??$ to the sequence h_i. Note that h_i abstracts from the values sent or received (these are already recorded by the input/output variables); it only records *when* an input/output along a channel occurred.

The extended program we denote by $P' = [S_1' \parallel ... \parallel S_n']$.

Definition 5.1 The (history) variables $h_1, ..., h_n$ are *compatible* in a state σ if there exists an interleaving h of the $\sigma(h_i)$ such that in all prefixes of h, the number of inputs on every channel in program P is less than or equal to the number of outputs on that channel. The compatibility of $h_1, ..., h_n$ is denoted by $Compat(h_1, ..., h_n)$.

Now consider the following invariant used in the rest of the proof:

$$
I = Compat(h_1, ..., h_n) \wedge \bigwedge_{\substack{1\le i\le n \\ c??\in I(S_i)}} |h_i|_{c??} = |c??| \wedge \bigwedge_{\substack{1\le i\le n \\ c!!\in O(S_i)}} |h_i|_{c!!} = |c!!|
$$

Here $|h_i|_{c??}$ gives the number of occurrences of $\underline{c}??$ in h_i, similarly for $|.|_{c!!}$. Furthermore, $I(S)$ and $O(S)$ denote the set of input and output variables of S.

The invariant I thus requires that the number of occurrences of $\underline{c}??$ ($\underline{c}!!$) in a history h_i is the same as the number of values received (sent) along channel c.

We now give a semantic characterization of the strongest postcondition given a statement S and precondition p:

Definition 5.2 Let S be such that its (program and input/output) variables are among $Pvar_i$ and $IOvar_i$. We define $\sigma \models SP(p, S)$ iff there exists $\langle \sigma_1, \sigma_1' \rangle ... \langle \sigma_n, \sigma_n' \rangle \in \mathcal{M}(S)$ such that $\sigma_1 \models p$ and σ, σ_n' agree on the (program and input/output) variables of $Pvar_i$ and $IOvar_i$.

The program variables and input/output variables of $SP(p,S)$ are among those of $Pvar_i$ and $IOvar_i$, because for any pair of states σ and σ' such that σ and σ' agree on the variables of $Pvar_i$ and $IOvar_i$, we have that $\sigma \models SP(p,S)$ implies $\sigma' \models SP(p,S)$. We assume the underlying logic to be sufficiently expressive to encode the above notion of strongest postcondition.

The following lemma is needed to establish completeness of our proof system:

Lemma 5.3 *For any bracketed substatement S of S'_i, $1 \leq i \leq n$, and assertion p, we have*

$$\vdash I : \{p\}S\{SP(p,S)\}.$$

Proof: The proof proceeds by induction on S and consists of a slight generalization of the completeness proof of the standard Hoare logic for sequential programs. We treat one case only:

$S = \langle c??x; h_i := h_i \cdot \underline{c}?? \rangle$: To show the derivability of $I : \{p\}S\{SP(p,S)\}$, for any p, it suffices to prove the validity of $(I \wedge p \wedge c \neq \epsilon) \rightarrow (I \wedge SP(p,S))[f(c)/x, c?? \cdot f(c)/c??, h_i \cdot \underline{c}??/h_i]$. To this end let $\sigma \models I \wedge p \wedge c \neq \epsilon$. It follows that $\langle \sigma, \sigma' \rangle \in \mathcal{M}(S)$, for $\sigma' = \sigma\{v/x, \sigma(c??) \cdot v/c??, \sigma(h_i) \cdot \underline{c}??/h_i\}$ with $v = f(\sigma(c))$. By the definition of $SP(p,S)$, it follows that $\sigma' \models SP(p,S)$. Moreover from $\sigma \models I$, the definition of I, and the semantics of S it follows in a straightforward manner that $\sigma' \models I$. (The main point is that $\sigma \models I \wedge c \neq \epsilon$ implies that $|\sigma(h_i)|_{c??} < |\sigma(h_j)|_{\underline{c}!!}$, assuming the jth component controls $c!!$; and this implies that in the compatible interleaving h of the histories $\sigma(h_1), \ldots, \sigma(h_n)$ there occurs an unmatched $\underline{c}!!$, and so $h \cdot \underline{c}??$ is a compatible interleaving of $\sigma(h_1), \ldots, \sigma(h_i) \cdot \underline{c}??, \ldots, \sigma(h_n)$.) Therefore, by the substitution lemma of the underlying logic, $\sigma \models (I \wedge SP(p,S))[f(c)/x, c?? \cdot f(c)/c??, h_i \cdot \underline{c}??/h_i]$, which had to be proved.

□

Now suppose that $\models \{p\}[S_1 \parallel \ldots \parallel S_n]\{q\}$ holds. We may assume that p and q do not contain occurrences of the auxiliary variables h_i. We have to show that $\vdash \{p\}[S_1 \parallel \ldots \parallel S_n]\{q\}$ is true. By $\bar{u}_i$ we denote the (list of) variables (both program variables and input/output variables) of S'_i, for $i \in \{1, \ldots, n\}$. In order to "split" the given global precondition p into local preconditions p_i, we introduce for each i a list $\bar{v}_i$ of fresh "freeze" variables corresponding to those of $\bar{u}_i$. Given the assertion $\bar{p} = p[\bar{v}_1/\bar{u}_1, \ldots, \bar{v}_n/\bar{u}_n]$ we can split p into the local assertions $p_i = \bar{p} \wedge \bar{u}_i = \bar{v}_i$.

Using lemma 5.3, we have for all $i = 1, \ldots, n$

$$\vdash I : \{p_i\}S'_i\{SP(p_i, S'_i)\}.$$

Hence, using the rule of parallel composition, we derive

$$\vdash \{I \wedge \bigwedge_{i=1}^{n} p_i\}[S'_1 \parallel \ldots \parallel S'_n]\{I \wedge \bigwedge_{i=1}^{n} SP(p_i, S'_i)\}$$

It is not difficult, but slightly tedious, to prove that $\models (I \wedge \bigwedge_{i=1}^{n} SP(p_i, S'_i)) \rightarrow q$. The main idea is the following: $\sigma \models SP(p_i, S'_i)$, for $1 \leq i \leq n$, implies the existence of $w_i \in \mathcal{M}(S'_i)$, such that the initial state of w_i satisfies p_i and σ and the final state of w_i agree on the variables of $Pvar_i$ and $IOvar_i$. Moreover $\sigma \models I$ implies the existence of a compatible interleaving h of the local histories $\sigma(h_i)$. On the basis of h one then can define a connected interleaving of the local computations w_i with final state σ and an initial state which satisfies $\bigwedge_{i=1}^{n} p_i$. Here we have to make use of that fact that for any w, if w agrees with some $w' \in \mathcal{M}(S'_i)$ w.r.t. the variables of $Pvar_i$ and $IOvar_i$ then also $w \in \mathcal{M}(S'_i)$, so we can vary arbitrarily in w_i the values of the variables not controlled by S'_i. Moreover the freeze variables $\bar{v}_i$, $i = 1, \ldots, n$, fix the initial state of the interleaving of the

computations w_i. The connected interleaving of the local computations w_i thus corresponds with a computation of P' which starts from a state satisfying $\bigwedge_{i=1}^{n} p_i$. Now since $\bigwedge_{i=1}^{n} p_i$ clearly implies p and we are given the validity of $\{p\}P\{q\}$ (from which follows the validity of $\{p\}P'\{q\}$), we thus obtain that $\sigma \models q$.

Applying the consequence rule yields

$$\vdash \{I \wedge \bigwedge_{i=1}^{n} p_i\}[S'_1 \parallel \ldots \parallel S'_n]\{q\}$$

Next we apply the auxiliary variables rule to obtain

$$\vdash \{I \wedge \bigwedge_{i=1}^{n} p_i\}[S_1 \parallel \ldots \parallel S_n]\{q\}$$

(Postcondition q does not refer to the auxiliary variables.)

Let *fifo* be the conjunction of the assertions $c?? \leq c!!$, where c is a channel occurring in p, q or the program P. Furthermore, let $\bar{h}$ be a list of the variables $h_1, \ldots, h_n$ and $\bar{v}$ be a list of the variables of $\bar{v}_i$, $1 \leq i \leq n$. It is easy to check that $p \wedge \mathit{fifo}$ implies $\exists \bar{h}, \bar{v}.I \wedge \bigwedge_i p_i$. Thus an application of the elimination rule, the consequence rule and the FIFO rule, in that order, yields

$$\vdash \{p\}P\{q\}$$

5.1 Discussion

The compositional proof system of [17] can be "extracted" from the above completeness proof simply by fixing the history variables as the auxiliary variables in the proof system and by fixing the above invariant I. The input/output statements then are axiomatized as multiple assignments including an update to the (local) history. For example, consider a statement of the form $\langle c??x; h_i := h_i \cdot \underline{c}?? \rangle$. To derive a correctness formula

$$I : \{p\}\langle c??x; h_i := h_i \cdot \underline{c}?? \rangle \{q\}$$

we have to establish the validity of

$$(I \wedge p \wedge c \neq \epsilon) \rightarrow (I \wedge q)[f(c)/x, c?? \cdot f(c)/c??, h_i \cdot \underline{c}??/h_i]. \tag{1}$$

This implication is equivalent with

$$p \rightarrow \forall v.q[v/x, c?? \cdot v/c??, h_i \cdot \underline{c}??/h_i] \tag{2}$$

This can be proved as follows: First we assume the validity of (1). Let $\sigma \models p$. Since p is local, i.e. it refers only to the (program and input/output) variables of Pvar_i and IOvar_i, for some i, and since I refers only to the length of $c!!$, it follows that for any value v, there exists a state σ' such that σ, σ' agree on the variables of Pvar_i and IOvar_i, $f(\sigma'(c)) = v$, and $\sigma' \models I \wedge c \neq \epsilon \wedge p$. ¿From the validity of (1) it then follows that $\sigma' \models (I \wedge q)[f(c)/x, c?? \cdot f(c)/c??, h_i \cdot \underline{c}??/h_i]$, and thus $\sigma' \models q[v/x, c?? \cdot v/c??, h_i \cdot \underline{c}??/h_i]$. The assertion q is assumed also to refer only to the variables of Pvar_i and IOvar_i, and so, since v is arbitrary, we may conclude that $\sigma \models \forall v.q[v/x, c?? \cdot v/c??, h_i \cdot \underline{c}??/h_i]$.

Conversely, let $\sigma \models I \wedge c \neq \epsilon \wedge p$, assuming the validity of (2). By definition of I and the validity of (2) it then immediately follows that $\sigma \models (I \wedge q)[f(c)/x, c?? \cdot f(c)/c??, h_i \cdot \underline{c}??/h_i]$.

The implication (2) corresponds with the axiom for the input statement in the compositional proof system of [17]. The invariant I then can be removed from the local correctness formulas, which thus will reduce to the standard Hoare triples of the form $\{p\}S\{q\}$. The rule for parallel composition then boils down to

$$\frac{\{p_i\}S_i\{q_i\},\ 1 \leq i \leq n}{\{\bigwedge_{i=1}^{n} p_i\}[S_1 \parallel \cdots \parallel S_n]\{\bigwedge_{i=1}^{n} q_i\}}$$

where I denotes the invariant used in the above completeness proof.

The above observation is analogous to the results of [4], where, for example, a completeness proof is given for the Apt, Francez and de Roever proof method [3], for systems based on synchronous communication, which incorporates the compositional proof system of [22]. The advantage of the proof system presented in this paper is that it allows more flexible reasoning patterns using auxiliary variables other than history variables.

6 Example

We now demonstrate the applicability of our proof system by proving correct a *distributed program for leader election.*

Consider program P=$[S_1 \parallel \ldots \parallel S_n]$ with

S_i=$z_i := 0$; y_i :=0; $c_i!!(x_i, 0)$;
*[$y_i \neq \infty$;$c_{i-1}??(y_i, hop_i)$; [x_i=$y_i \rightarrow z_i := 1$;$y_i := \infty$; $c_i!!(\infty, hop_i + 1)$
□
$x_i > y_i \rightarrow skip$
□
$x_i < y_i \rightarrow c_i!!(y_i, hop_i + 1)$
]
]

Figure 1: Distributed Leader Election

In this program, there are n+1 processes, $n > 1$ arranged in a ring, with channel c_i connecting S_i to S_{i+1}. Addition and subtraction involving such indices i is done modulo n+1. Each process S_i has a local variable x_i (all indices are explicit, for readability). Distinct variables x_i and x_j represent distinct integer values. Here "∞" denotes a value larger than all integers. In addition, each process has a local variable z_i whose initial value is irrelevant. Upon termination, it is required that there exists *exactly one* index i_0, such that $z_{i_0} = 1$, and $z_j = 0$, for all $j \neq i_0$. The idea is to choose as the "leader" S_{i_0} such that x_{i_0}=$max\{x_k \mid 1 \leq k \leq n\}$, i.e., the index of the process with the largest value of x_i.

Each process sends its value x_i to the right, and propagates to the right any received value x larger than its own x_i. The maximal value x_{i_0} is the only one to traverse the whole ring. When it arrives back to its origin, a "∞" message is sent around the ring to terminate all other processes. The

second component *hop* in message (x, hop) records the number of hops, i.e., the distance that x has traversed in the ring. (This component as well as the variables hop_i, $i = 1, \ldots, n$, have been added only to simplify the correctness proof. Strictly speaking, they are not needed.) It is assumed that for i=0, $\cdots$, n, $x_i < \infty$ holds, and that ∞ and pairs such as (x, hop) can be encoded as natural numbers to conform to the syntax of programs.

The correctness assertion we want to establish is:

$$(*) \vdash \begin{array}{l} \{\bigwedge_{i=0}^{n}(c_i = \epsilon \wedge x_i = X_i \neq \infty \wedge \bigwedge_{j=0}^{n}(j \neq i \Rightarrow X_j \neq X_i))\} \\ \qquad P \\ \{\bigvee_{i=1}^{n}(z_i = 1 \wedge \bigwedge_{j=0}^{n}(j \neq i \Rightarrow (z_j = 0 \wedge X_i > X_j)))\}. \end{array}$$

For this example, there is no need to introduce bracketed sections.

Let p_i=$c_{i-1}??$=$c_i!!$=$\epsilon \wedge x_i$=$X_i \neq \infty \wedge \bigwedge_{j=0}^{n}(j \neq i \Rightarrow X_j \neq X_i)$, for $i = 1, \ldots, n$; and let I=$\bigwedge_{i=1}^{n} c_i?? \leq c_i!!$ We construct local correctness assertions for each S_i. We have:

$$(1) \vdash I: \begin{array}{l} \{c_{i-1}?? = c_i!! = \epsilon \wedge x_i = X_i \neq \infty \wedge \bigwedge_{j=0}^{n}(j \neq i \Rightarrow X_j \neq X_i)\} \\ z_i := 0 \\ \{z_i = 0 \wedge c_{i-1}?? = c_i!! = \epsilon \wedge x_i = X_i \neq \infty \wedge \bigwedge_{j=0}^{n}(j \neq i \Rightarrow X_j \neq X_i)\} \end{array}$$

as a consequence of (*assignment*), and

$$(2) \vdash I: \begin{array}{l} \{z_i = 0 \wedge c_{i-1}?? = c_i!! = \epsilon \wedge x_i = X_i \neq \infty \wedge \bigwedge_{j=0}^{n}(j \neq i \Rightarrow X_j \neq X_i)\} \\ y_i := 0 \\ \{y_i = 0 \wedge z_i = 0 \wedge c_{i-1}?? = c_i!! = \epsilon \wedge x_i = X_i \neq \infty \wedge \bigwedge_{j=0}^{n}(j \neq i \Rightarrow X_j \neq X_i)\} \end{array}$$

as a consequence of (*assignment*), and

$$(3) \vdash I: \begin{array}{l} \{y_i = 0 \wedge z_i = 0 \wedge c_{i-1}?? = c_i!! = \epsilon \wedge x_i = X_i \neq \infty \wedge \bigwedge_{j=0}^{n}(j \neq i \Rightarrow X_j \neq X_i)\} \\ c_i!!(x_i, 0) \\ \{y_i = 0 \wedge z_i = 0 \wedge c_{i-1}?? = \epsilon \wedge c_i!! = <(x_i, 0)> \wedge x_i = X_i \neq \infty \wedge \bigwedge_{j=0}^{n}(j \neq i \Rightarrow X_j \neq X_i)\} \end{array}$$

as a consequence of (*output*).

To proceed we, obviously, need a loop invariant L_i. It comprises a number of conjuncts formulated below. (We have named these conjuncts for convenience.) For a sequence h and value x, $x \in h$ denotes that x occurs in h; $h[i]$ denotes the i^{th} element of h, provided that $1 \leq i \leq |\ h\ |$ holds; and $last(h)$ denotes $h[|\ h\ |]$.

Define L_i by the conjunction of

(A) x_i=$X_i \neq \infty \wedge (z_i$=$0 \vee z_i$=$1) \wedge \bigwedge_{j=0}^{n}(j \neq i \Rightarrow X_j \neq X_i)$.

(B) z_i=$1 \Rightarrow y_i$=∞.

(C) $(\exists hop.\ (x_i, hop) \in c_{i-1}??) \Leftrightarrow z_i$=$1$.

(D) $\forall x, hop.\ ((x, hop) \in c_i!! \Rightarrow x \geq x_i)$.

(E) $\forall hop.((\infty, hop) \in c_i!! \Rightarrow \quad (hop \neq 0 \wedge ((\infty, hop-1) \in c_{i-1}?? \vee (x_i, hop-1) \in c_{i-1}??)))$.

(F) $\forall x, hop.\ ((x, hop) \in c_i!! \Rightarrow \quad (x$=$x_i \wedge hop$=$0) \vee (x$=$\infty \wedge hop$>$0) \vee ((x, hop-1) \in c_{i-1}?? \wedge hop$>$0))$.

(G) $y_i = \infty \Rightarrow \exists hop \geq 0.\ ((x_i, hop) \in c_{i-1}?? \vee (\infty, hop) \in c_{i-1}??)$.

The loop invariant is established indeed:

$$(4)\quad \begin{array}{l}(\quad y_i{=}0 \wedge z_i{=}0 \wedge c_{i-1}?? = \epsilon \wedge c_i!! =< (x_i, 0) > \\ \quad \wedge\ x_i{=}X_i{\neq}\infty \wedge \bigwedge_{j=0}^{n}(j \neq i \Rightarrow X_i{\neq}X_j)) \\ \Rightarrow L_i.\end{array}$$

We show in some detail that L_i serves as an invariant for the loop in S_i. First, we introduce assertion ALT_i, holding prior to the branching points in S_i. It also comprises several numbered conjuncts. ALT_i is the conjunction of (A), (D), and

(B′) $z_i{=}0$,

(C′) $\forall hop, k.\ (1{\leq}\ k <|\ c_{i-1}??\ | \Rightarrow (x_i,\ hop) \neq c_{i-1}??[k])$,

$$(E')\ \forall hop.\quad \begin{array}{l}((\infty,\ hop){\in}\ c_i!! \\ \Rightarrow (\quad hop{\neq}0 \\ \quad \wedge\ \exists k.(1{\leq}k{<}|c_{i-1}??| \wedge ((\infty, hop{-}1){=}c_{i-1}??[k]{\vee}(x_i,\ hop{-}1){=}c_{i-1}??[k]))),\end{array}$$

$$(F')\ \forall x, hop.\ (x,\ hop){\in}\ c_i!! \Rightarrow \begin{array}{l}(x{=}x_i \wedge hop{=}0) \\ \vee\ (x{=}\infty \wedge hop{>}0) \\ \vee\ \exists k.\ (1{\leq}k{<}|c_{i-1}??| \wedge (x, hop{-}1){=}c_{i-1}??[k] \wedge hop{>}0),\end{array}$$

$$(G')\ y_i = \infty \Rightarrow \exists hop \geq 0.\exists k.(\quad \begin{array}{l}1 \leq k <|\ c_{i-1}??\ | \\ \wedge\ ((x_i, hop){=}c_{i-1}??[k] \vee (\infty, hop){=}c_{i-1}??[k])),\end{array}$$

(H′) $c_{i-1}?? \neq \epsilon \wedge last(c_{i-1}??){=}(y_i,\ hop_i) \wedge hop_i{\geq}0$.

We note that

$$(5)\ \vdash \{L_i \wedge y_i \neq \infty\}\ c_{i-1}??(y_i, hop_i)\ \{ALT_i\}$$

is obtained via (*input*). We next consider the separate directions.
For the first direction we reason as follows:

$$(6)\ \vdash \begin{array}{l}\{ALT_i \wedge x_i = y_i\} \\ z_i := 1 \\ \{(A) \wedge z_i = 1 \wedge (C') \wedge (D) \wedge (E') \wedge (F') \wedge (G') \wedge (H') \wedge x_i = y_i\}.\end{array}$$

$$(7)\ \vdash \begin{array}{l}\{(A) \wedge z_i = 1 \wedge (C') \wedge (D) \wedge (E') \wedge (F') \wedge (G') \wedge (H') \wedge x_i = y_i\} \\ y_i := \infty \\ \{(A){\wedge}z_i{=}\ 1{\wedge}y_i{=}\infty{\wedge}(C'){\wedge}(D){\wedge}(E'){\wedge}(F'){\wedge}c_{i-1}??{\neq}\epsilon{\wedge}last(c_{i-1}??){=}(x_i, hop_i){\wedge}hop_i{\geq}0\}.\end{array}$$

$$(8)\ \vdash \begin{array}{l}\{(A){\wedge}z_i{=}\ 1{\wedge}y_i{=}\infty{\wedge}(C'){\wedge}(D){\wedge}(E'){\wedge}(F'){\wedge}c_{i-1}??{\neq}\epsilon{\wedge}last(c_{i-1}??){=}(x_i, hop_i){\wedge}hop_i{\geq}0\} \\ c_i!!(\infty, hop_i + 1) \\ \{L_i\}.\end{array}$$

Thus, the first direction re-establishes L_i. The second direction also re-establishes L_i, since we have

$$(9)\ \vdash \{ALT_i \wedge x_i{>}y_i\}\ skip\ \{L_i\}.$$

Finally, we have

(10) $\{ALT_i \wedge x_i{<}y_i\}\ c_i!!(y_i,\ hop_i{+}1)\ \{L_i\}$.

Therefore, the third direction re-establishes L_i, too.

By combining the above local correctness assertions, we obtain that $\vdash I : \{p_i\}S_i\{q_i\}$ holds for $q_i = (L_i \wedge y_i = \infty)$, $i = 1, \ldots, n$.

Application of the parallel composition rule then yields $\vdash \{I \wedge \bigwedge_{i=1}^{n} p_i\}\ P\ \{I \wedge \bigwedge_{i=1}^{n} q_i\}$. By repeatedly applying the *FIFO* rule, we obtain that $\vdash \{\bigwedge_{i=1}^{n} p_i\}\ P\ \{I \wedge \bigwedge_{i=1}^{n} q_i\}$ holds.

We now derive correctness formula (*) to be established for the leader election program by application of the global consequence rule. It suffices to prove that $I \wedge \bigwedge_{i=1}^{n} q_i$ implies $\{\bigvee_{i=1}^{n}(z_i = 1 \wedge \bigwedge_{j=1}^{n}(j \neq i \Rightarrow (z_j = 0 \wedge X_i > X_j)))\}$.

To do so, assume $I \wedge \bigwedge_{i=1}^{n} q_i$ holds. Since q_i has been defined as $L_i \wedge y_i = \infty$ and L_i has been defined as the conjunction of (A), $\cdots$, (G), for all i=1, $\cdots$, n, we get from clause (G) that

(i) for all $i = 1, \cdots, n$, there exists some $hop{\geq}0$ with $(x_i, hop) \in c_{i-1}?? \vee (\infty, hop) \in c_{i-1}??$.

¿From this we derive by *reductio ad absurdum* that

(ii) for some $j = 1, \cdots, n$, and for some $hop{\geq}0$, $(x_j, hop) \in c_{j-1}??$ holds.

Suppose that (ii) is not the case. Then, for all $j = 1, \cdots, n$, and for all $hop{\geq}0$, $(x_j, hop) \notin c_{j-1}??$ holds. Consider an arbitrary $k \in \{1, \cdots, n\}$. From (i) we obtain that there exists some $hop_0{\geq}0$, such that $(\infty, hop_0) \in c_{k-1}??$. From I, $(\infty, hop_0) \in c_{k-1}!!$ is obtained. Then (E) implies that for some hop_1 with $hop_0 > hop_1 \geq 0$, $(x_{k-1}, hop_1) \in c_{k-2}?? \vee (\infty, hop_1) \in c_{k-2}??$ is true. Because (ii) is assumed to be false, $(\infty, hop_1) \in c_{k-2}??$ follows. By repeating this process ad infinitum we find an infinite decreasing sequence $hop_0 > hop_1 > \cdots$ of natural numbers, which is impossible. We conclude that (ii) is satisfied.

Now, consider an arbitrary $i_0 \in \{1, \cdots, n\}$ such that for some $hop{\geq}0$, $(x_{i_0}, hop) \in c_{i_0-1}??$ holds. From (C) we obtain that z_{i_0}=1 is true; I implies that for some $hop{\geq}0$, $(x_{i_0}, hop) \in c_{i_0-1}!!$ holds. Then, $x_{i_0} \geq x_{i_0-1}$ follows from (D). From (A) and (F) we obtain that for some $hop{\geq}0$, $(x_{i_0}, hop) \in c_{i_0-2}??$. Continuing this way, we get $(x_{i_0}, hop) \in c_{i_0-1}?? \Rightarrow \bigwedge_{1 \leq j \leq n} x_{i_0} \geq x_j$. Together with (A) this proves uniqueness of i_0; and hence the righthandside of the implication we wished to prove.

Consequently, correctness of the distributed leader election program follows from rule (*Consequence*). This concludes our proof, and hence this section.

7 Reasoning about non-terminating processes

The proof system presented in Section 3 allows for the verification of partial correctness properties of programs, i.e. properties which hold upon termination. In this section we discuss briefly how to extend the proof system in order to reason about non-terminating processes. The main idea is to extend the global correctness formulas with an invariant which describes properties which hold throughout any computation, i.e. we have global correctness formulas of the form $I : \{p\}P\{q\}$, where I is an assertion which is allowed to refer only to the input/output variables and the auxiliary variables of P. Intuitively, $I : \{p\}P\{q\}$ is valid iff

> I holds throughout any computation of P which starts in a state satisfying p and I, and upon termination q holds.

In order to define formally the semantics of these extended global correctness formulas, we introduce the following semantics of a statement S:

Definition 7.1 Let, for $w = \langle\sigma_1, \sigma_1'\rangle \cdots \langle\sigma_n, \sigma_n'\rangle$, $S \stackrel{w}{\Rightarrow}$ hold if there exist $S_1, \ldots, S_n$ such that $S_i \stackrel{\langle\sigma_i, \sigma_i'\rangle}{\Longrightarrow} S_{i+1}$, for $1 \leq i < n$, and $S{=}S_1$. For S with $Pvar(S) \subseteq Pvar_m$ and $IOvar(S) \subseteq IOvar_m$, we define

$$\mathcal{M}^\infty(S) = \{w \mid S \stackrel{w}{\Rightarrow} \text{ and } w \text{ is } m\text{-connected}\}$$

Thus $\mathcal{M}^\infty(S)$ includes the prefixes of non-terminating computations of S. The semantics $\mathcal{M}^\infty(P) \subseteq (\Sigma \times \Sigma)^*$ of program P is defined as follows:

Definition 7.2 For any program $P = [S_1 \parallel \ldots \parallel S_n]$ we define

$$\mathcal{M}^\infty(P) = \beta(\mathcal{M}^\infty(S_1) \parallel \ldots \parallel \mathcal{M}^\infty(S_n))$$

where the abstraction operation β simply selects all the *connected* sequences which start in a state σ such that $\sigma(c??)$ is a prefix of $\sigma(c!!)$, for every channel c in program P.

For states σ, $\mathcal{M}^\infty(S)(\sigma)$ and $\mathcal{M}^\infty(P)(\sigma)$ are defined in the obvious way.

The semantics of local correctness formulas $I : \{p\}S\{q\}$ are defined as before, but now w.r.t. $\mathcal{M}^\infty(S)$.

We next define the semantics of global correctness formulas.

Definition 7.3 For any program P, the correctness formula $I : \{p\}P\{q\}$ is valid iff for any $w = \langle\sigma_1, \sigma_1'\rangle \cdots \langle\sigma_n, \sigma_n'\rangle \in \mathcal{M}^\infty(P)$ we have that $\sigma_1 \models p \wedge I$ implies $\sigma_i' \models I$, for $1 \leq i \leq n$, and if $\langle\sigma_1, \sigma_n'\rangle \in \mathcal{M}(P)$ then $\sigma_n' \models q$.

Note that in the above definition the condition $\sigma_i' \models I$, for $1 \leq i \leq n$, amounts to requiring that I holds in every state of $w = \langle\sigma_1, \sigma_1'\rangle \cdots \langle\sigma_n, \sigma_n'\rangle$, since w is connected (that is, $\sigma_i' = \sigma_{i+1}$, for $1 \leq i < n$).

The local proof system remains as before. The rule for parallel composition however now becomes:

Rule 12 *(parallel composition)* $$\frac{I : \{p_i\}S_i\{q_i\},\ i = 1, \ldots, n}{I : \{\bigwedge_i p_i\}[S_1 \parallel \ldots \parallel S_n]\{\bigwedge_i q_i\}}$$

Furthermore we have the following versions of the global consequence rule, the auxiliary variables rule, and the FIFO rule.

Rule 13 *(global consequence rule)* $$\frac{I \to I',\ (I' \wedge p') \to \exists\bar{x}(I \wedge p),\ I : \{p\}P\{q\},\ (I \wedge q) \to q'}{I' : \{p'\}P\{q'\}}$$

In the above rule $\bar{x}$ is a list of variables which do not occur in the assertions I', p', q' or the program P. Note that the above consequence rule additionally allows for the elimination of variables in the invariant and the precondition.

Rule 14 *(auxiliary variables rule)* $\dfrac{I : \{p\}P'\{q\}}{I : \{p\}P\{q\}}$

where P is obtained from P' by deleting all assignments to the auxiliary variables. It is assumed that the postcondition q and the invariant I do not refer to the auxiliary variables of P'.

Rule 15 *(FIFO rule)*

$$\frac{I : \{p \wedge c?? \leq c!!\}P\{q\}}{I : \{p\}P\{q\}}$$

where channel c occurs in program P. As before, in the above rule "$\leq$" denotes the prefix relation on sequences.

Soundness of the above proof system follows by a straightforward induction on the length of the proof. We sketch the completeness proof: We first introduce the following notion of a strongest (local) invariant.

Definition 7.4 Let S be a (bracketed) statement and p be some local assertion. Let $I(p, S)$ be such that $\sigma \models I(p, S)$ iff $\sigma \models p$ or there exists a computation $\langle \sigma_1, \sigma_1' \rangle \cdots \langle \sigma_n, \sigma_n' \rangle \in \mathcal{M}^\infty(S)$, such that $\sigma_1 \models p$; and σ and σ_n' agree on the input/output variables and the auxiliary variables of S.

Roughly, $I(p, S)$ characterizes the set of all reachable states of computations of S starting in a state satisfying p. Note that the variables of $I(p, S)$ are among the input/output variables and the auxiliary variables of S since for any states σ, σ', such that σ and σ' agree on those variables of S, we have that $\sigma \models I(p, S)$ implies $\sigma' \models I(p, S)$.

We have the following lemma, corresponding to lemma 5.3:

Lemma 7.5 *For any statement S and local assertion p, we have*

$$\vdash I(p, S) : \{p\}S\{SP(p, S)\}$$

where $SP(p, S)$ is defined as in the completeness proof of the proof system for partial correctness.

Proof The proof proceeds by induction on the complexity of S. We treat the following cases:

$S = \langle c??x; y := e \rangle$: It suffices to show the validity of $(I(p, S) \wedge p \wedge c \neq \epsilon) \rightarrow (I(p, S) \wedge SP(p, S))[f(c)/x, c?? \cdot f(c)/c??, e/y]$. Nota that since p implies $I(p, S)$ the above implication is equivalent to $((p \wedge c \neq \epsilon) \rightarrow (I(p, S) \wedge SP(p, S))[f(c)/x, c?? \cdot f(c)/c??, e/y]$, the validity of which is easy to verify.

$S = S_1; S_2$: By the induction hypothesis we have that

$$\vdash I(p, S_1) : \{p\}S_1\{SP(p, S_1)\}$$

and

$$\vdash I(SP(p, S_1), S_2) : \{SP(p, S_1)\}S_2\{SP(SP(p, S_1), S_2)\}.$$

To obtain the desired result we need the following derived rule: For any statement S', if $\vdash I : \{p\}S'\{q\}$, where I refers only to the (program and input/output) variables of S', then $\vdash I' : \{p \wedge I\}S'\{q\}$, for any I' such that $\models I \to I'$. The proof of this rule follows by a straightforward induction on S'. Moreover, it is easy to verify the validity of the implications $I(p, S_1) \to I(p, S_1; S_2)$ and $I(SP(p, S_1), S_2) \to I(p, S_1; S_2)$. So we derive that

$$\vdash I(p, S_1; S_2) : \{p \wedge I(p, S_1)\}S_1\{SP(p, S_1)\}$$

and

$$\vdash I(p, S_1; S_2) : \{SP(p, S_1) \wedge I(SP(p, S_1), S_2)\}S_2\{SP(SP(p, S_1), S_2)\}.$$

Next we observe that for any statement S' and local assertion q we have that q implies $I(q, S')$. So by the local consequence rule and the rule for sequential composition we, finally, derive

$$\vdash I(p, S_1; S_2) : \{p\}S_1; S_2\{SP(p, S_1; S_2)\}$$

$S = \star\|_i[\langle b_i; c_i??x_i; y_i := e_i\rangle \to R_i]$: Let $\bar{x}$ be a list of the variables of S and $\bar{z}$ some fresh corresponding variables. Let R be the statement obtained from S by replacing the booleans b_i by $b_i \wedge \neg(\bar{x} = \bar{z})$. For any given p let $p' = \exists\bar{z}SP(p, R)$. By the induction hypothesis we derive that $\vdash I(p' \wedge b_i, R_i') : \{p' \wedge b_i\}R_i'\{SP(p' \wedge b_i, R_i')\}$, where $R_i' = c_i??x_i; y_i := e_i; R_i$. Since $I(p' \wedge b_i, R_i')$ implies $I(p, S)$, it follows as described in the case above that $\vdash I(p, S) : \{p' \wedge b_i\}R_i'\{SP(p' \wedge b_i, R_i')\}$. Now it is not difficult to verify that $SP(p' \wedge b_i, R_i')$ implies p'. Thus we have shown that p' is an invariant of S, so we derive that $\vdash I(p, S) : \{p'\}S\{p' \wedge \bigwedge_i \neg b_i\}$. Applying the consequence rule, using the validity of the implications $p \to p'$ and $p' \wedge \bigwedge_i \neg b_i \to SP(p, S)$, we conclude that $\vdash I(p, S) : \{p\}S\{SP(p, S)\}$.

□

Now let $P = [S_1 \parallel \cdots \parallel S_n]$ be a program without bracketed sections, for which $I : \{p\}P\{q\}$ is a valid correctness formula. We extend P with history variables, which we assume not to occur in P, p or q. This extended program we denote by $P' = [S_1' \parallel \cdots \parallel S_n']$. We now split the conjunction of the given precondition p and invariant I into the following local assertions p_i': Let $\bar{u}_i$ be the (program and input/output) variables of S_i' $(i = 1, \ldots, n)$: We define $p_i' = (p \wedge I)[\bar{v}/\bar{u}] \wedge \bar{v}_i = \bar{u}_i$ ($\bar{v}$ denotes the union of $\bar{v}_i$, $i = 1, \ldots, n$, similarly for $\bar{u}$).

By the above lemma we have

$$\vdash I(p_i', S_i') : \{p_i'\}S_i'\{SP(p_i', S_i')\},\ i = 1, \ldots, n.$$

In order to proceed we need the following derived rule: If $\vdash I : \{p\}S\{q\}$ and $\vdash I' : \{p'\}S\{q'\}$ then $\vdash I \wedge I' : \{p \wedge p'\}S\{q \wedge q'\}$. This rule follows by a straightforward induction on the complexity of S. Moreover, it is also easy to verify that, for any S and local p, $\vdash I : \{p\}S\{SP(p, S)\}$, where I does not contain variables of S.

So for $I' = (\bigwedge_{i=1}^n I(p_i', S_i')) \wedge I''$, where I'' denotes the invariant defined in the completeness proof for partial correctness, we obtain

$$I' : \{p_i'\}S_i'\{SP(p_i', S_i')\},\ \text{for } i = 1, \ldots, n$$

Next we apply the parallel composition rule

$$I' : \{\bigwedge_{i=1}^n p_i'\}[S_1' \parallel \cdots \parallel S_n']\{\bigwedge_{i=1}^n SP(p_i', S_i')\}$$

Since we are given the validity of $I : \{p\}P\{q\}$, it follows in a similar manner as in the completeness proof for partial correctness that

$$\models (I'' \wedge \bigwedge_{i=1}^{n} SP(p_i', S_i')) \rightarrow q \tag{3}$$

Next we argue that

$$\models I' \rightarrow I \tag{4}$$

Choose state σ such that $\sigma \models I'$ holds. Therefore, for $1 \leq i \leq n$, $\sigma \models p_i'$ or there exists a computation $w_i \in \mathcal{M}^\infty(S_i')$, such that the initial state of w_i satisfies p_i' and σ and the final state of w_i agree on the input/output variables and the auxiliary variables of S_i'. Moreover, since $\sigma \models I''$, we can define a connected interleaving of the local computations w_i with final state σ and an initial state which satisfies $\bigwedge_{i=1}^{n} p_i'$ (again using that we can vary the values of the variables not controlled by S_i'). Thus, since clearly $\bigwedge_{i=1}^{n} p_i'$ implies $p \wedge I$, we have that σ is an intermediate state of a computation of P' which starts in a state satisfying p and I. And so, by the validity of $I : \{p\}P\{q\}$ (which implies the validity of $I : \{p\}P'\{q\}$), we have that $\sigma \models I$.

Moreover it is not difficult to check that

$$\models (I \wedge p \wedge \mathit{fifo}) \rightarrow \exists \bar{x}(I' \wedge \bigwedge_{i=1}^{n} p_i') \tag{5}$$

where $\bar{x}$ is a list of the newly introduced freeze variables and history variables and *fifo* is a conjunction of assertion $c?? \leq c!!$, for channels c occurring in P. So by an application of the consequence rule using (3), (4) and (5) above, and an application of the *FIFO* rule we obtain

$$\vdash I : \{p\}P'\{q\}$$

¿From which we derive by an application of the auxiliary variables rule our desired result

$$\vdash I : \{p\}P\{q\}$$

We conclude this section with the observation that the above completeness proof embodies a compositional proof method for reasoning about non-terminating processes. As in the case of partial correctness, this proof method can be extracted from the above completeness proof by fixing the auxiliary variables as the history variables and axiomatizing the input/output statements as multiple assignments including an update to a history variable. Moreover when restricting to history variables as auxiliary variables the above completeness proof shows that we can restrict to local correctness formulas $I : \{p\}S\{q\}$, where I is a local assertion, i.e. I refers only to the input/output variables and the auxiliary variables of S. The corresponding parallel composition rule

$$\frac{I_i : \{p_i\}S_i\{q_i\} \quad i = 1, \ldots, n}{\bigwedge_{i=1}^{n} I_i : \{\bigwedge_{i=1}^{n} p_i\}[S_1 \parallel \cdots \parallel S_n]\{\bigwedge_{i=1}^{n} q_i\}}$$

as shown above can be derived in the original proof system.

8 Reasoning about deterministic processes

So far, we have seen how to reason about terminating and non-terminating processes by means of invariants. These invariants in general express properties of both input/output variables and auxiliary variables (which include history variables).

In this section, we show how we can limit the expressiveness of the invariants to properties of merely the input/output variables, provided we restrict our programming language to *deterministic processes* only. Since auxiliary variables are no longer needed, we can discard the notion of bracketed sections as well.

To define deterministic processes, we replace the guarded conditional and the guarded iteration in definition 2.1 by if b then S_1 else S_1 fi and while b do S od respectively. The resulting language is capable of expressing non-trivial programs. For example, the distributed algorithm for leader election discussed in Section 6 can be formulated as a parallel composition of deterministic processes, in fact its correctness proof did not require the introduction of auxiliary variables. Intuitively, the successful execution of a deterministic process in a parallel environment depends heavily –even more so than for nondeterministic processes– on the behaviour of its environment. For instance a process waiting on some input can only check one input channel at a time, and as long as this channel remains empty, it is idle while waiting. Therefore we enhance the programming language with three new constructs which allow for more flexibility. Let us give the formal definition of the language:

Definition 8.1 The syntax of a deterministic (sequential) statement S in our programming language is defined by

$$
\begin{array}{rcl}
S & ::= & \mathsf{skip} \\
 & | & x := e \\
 & | & c??x \mid c!!e \\
 & | & S_1 ; S_2 \\
 & | & \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi} \\
 & | & \mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od} \\
 & | & \mathsf{if}\ c??x\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi} \\
 & | & \mathsf{while}\ c??x\ \mathsf{do}\ S\ \mathsf{od} \\
 & | & \mathsf{repeat}\ S\ \mathsf{until}\ c??x
\end{array}
$$

The execution of a statement if $c??x$ then S_1 else S_2 fi consists of receiving a value from channel c, in case its corresponding buffer is non-empty, recording it in x and proceeding with S_1. In case the buffer is empty control moves on to S_2. The execution of a statement while $c??x$ do S od consists of alternatingly reading a value from channel c and executing S until the corresponding buffer is empty. Finally repeat S until $c??x$ models a form of busy waiting: repeat S for as long as no value can be read from channel c. Note that, intuitively speaking, $c??x$ is equivalent to repeat skip until $c??x$; this corresponds to the "idle waiting" inherent in $c??x$ as discussed above (however semantically speaking, there *is* a difference as discussed later in this section.)

¿From a semantic point of view, we can proceed in a similar way to that of definition 2.4. However in order to reason compositionally about parallel processes it is also necessary to record tests that "fail", i.e. tests within a compound construct (the last three constructs in our syntax) which are executed when the corresponding buffer is empty. We represent such a test on an empty buffer by the special value $\perp \notin Val$, which will be appended to the input variable under consideration. For example, the sequence $\langle 1, 2, 3, \perp, 4, 5 \rangle$ representing the values received from a channel indicates that after $1, 2$ and 3 have been received the process tested the contents of the channel when it was empty. Subsequent receive operations on the channel resulted in the values 4 and 5.

As a consequence, our set of states now is defined by $\Sigma = (PVar \cup IOVar) \to (Val \cup Val_\perp^*)$, where $Val_\perp^*$ denotes the set of finite sequences over $Val_\perp = Val \cup \{\perp\}$. As before, it is required that programming variables are mapped into Val, and that input/output variables are mapped into $Val_\perp^*$. Furthermore, given a sequence $s \in Val_\perp^*$, the subsequence of s consisting of elements of Val only—s with all appearances of $\perp$ removed—we denote by $r_\perp(s)$. Then, we have that the value $\sigma(c)$ of the buffer corresponding to a channel c in a state σ, that is, the sequence of values sent along c but not yet received, is given by $\sigma(c!!) - r_\perp(\sigma(c??))$. For example, if $\sigma(c!!) = \langle 1,2,3\rangle$ and $\sigma(c??) = \langle 1,\perp,2\rangle$ then $\sigma(c) = \langle 3\rangle$.

As an example, we state the following two semantic clauses for the conditional input statement:

if $c??x$ then S_1 else S_2 fi $\stackrel{\langle\sigma,\sigma'\rangle}{\Longrightarrow} S_1$,
provided $\sigma(c) \neq \epsilon$. Here $\sigma' = \sigma\{\sigma(c??) \cdot v/c??, v/x\}\rangle$ and $v = f(\sigma(c)))$.

if $c??x$ then S_1 else S_2 fi $\stackrel{\langle\sigma,\sigma'\rangle}{\Longrightarrow} S_2$,
provided $\sigma(c) = \epsilon$. Here $\sigma' = \sigma\{\sigma(c??) \cdot \perp/c??\}\rangle$.

The axioms and rules are for the most part simplifications of the earlier axioms and rules, leaving out the references to the auxiliary variables. For instance, the rule for the input statement now becomes:

Rule 16 *(input)*

$$\frac{(I \wedge p \wedge c \neq \epsilon) \to (I \wedge q)[f(c)/x, c?? \cdot f(c)/c??]}{I : \{p\}c??x\{q\}}$$

With respect to the three novel language constructs, we have the following rules:

Rule 17 *(input conditional)*

$$\frac{I : \{p\}c??x; S_1\{q\},\ (I \wedge p \wedge c = \epsilon) \to (I \wedge r)[c?? \cdot \perp/c??],\ I : \{r\}S_2\{q\}}{I : \{p\}\text{if } c??x \text{ then } S_1 \text{ else } S_2 \text{ fi}\{q\}}$$

Rule 18 *(input while loop)*

$$\frac{I : \{p\}c??x; S\{p\},\ (I \wedge p \wedge c = \epsilon) \to (I \wedge q)[c?? \cdot \perp/c??]}{I : \{p\}\text{while } c??x \text{ do } S \text{ od}\{q\}}$$

Rule 19 *(repeat)*

$$\frac{I : \{p\}S\{r\},\ (I \wedge r \wedge c = \epsilon) \to (I \wedge p)[c?? \cdot \perp/c??],\ I : \{r\}c??x\{q\}}{I : \{p\}\text{repeat } S \text{ until } c??x\{q\}}$$

Soundness of the above proof system follows by a straightforward induction on the length of the proof. We sketch the completeness proof which follows closely the completeness proof of [1]. Let $\{p\}P\{q\}$ be a valid correctness formula, where $P = [S_1 \parallel \cdots \parallel S_n]$. Now we introduce local correctness formulas based on the following *reachability* predicates as defined in [1].

Definition 8.2 A state σ is called *reachable* if there exists a computation of P starting in a state satisfying p which passes through a state σ' such that the control of each component is at a location outside a bracketed section, and σ and σ' agree on the input/output variables of P. Let I' be an assertion such that $\sigma \models I'$ iff there exists reachable state σ' such that σ and σ' agree on the input/output variables of P.

Definition 8.3 For any substatement S of S_i, let $pre(S)$ and $post(S)$ be a (local) assertion such that $\sigma \models pre(S)$ ($\sigma \models post(S)$) iff there exists a computation of P starting in a state satisfying p which passes through a state σ' such that the the ith component of P is just about to execute S (has just terminated S) and σ and σ' agree with respect to the variables of S_i.

We have the following key lemma:

Lemma 8.4 *The local correctness formulas* $I' : \{pre(S_i)\}S_i\{post(S_i)\}$ *are derivable.*

Proof The proof proceeds by induction on the complexity of S_i. The main case of the proof is that of a statement $S = c??x$. The validity of the implication $(I' \wedge pre(S) \wedge c \neq \epsilon) \rightarrow (I' \wedge post(S))[f(c)/x, c?? \cdot f(c)/c??]$ follows from the *merging* lemma. This lemma states that the local run of a component in a computation of P can be replaced by another local run of that component as long as they behave the same w.r.t. the input/output variables. The merging lemma itself follows from the fact that the input/output variables completely determine the behaviour of a deterministic process (since they record failed tests). □

Application of the parallel composition rule then gives us the derivability of

$$\{I' \wedge \bigwedge_{i=1}^{n} pre(S_i)\}P\{I' \wedge \bigwedge_{i=1}^{n} post(S_i)\}.$$

It is easy to verify, using the above mentioned merging lemma, the validity of the assertion $(I' \wedge \bigwedge_i post(S_i)) \rightarrow q$. Moreover, it also easily follows that $\models (p \wedge \bigwedge_{c \in IOVar(P)} c?? < c!!) \rightarrow I'$. Thus by the consequence rule and the *FIFO* rule we obtain the derivability of

$$\vdash \{p\}P\{q\}.$$

It is worthwhile to mention the difference between the above completeness proof and the ones given before. In the completeness proof of the system for proving partial correctness properties we introduced so-called *generic* local correctness formulas of the components of a program, i.e. these formulas are defined independently of the particular given program. As such these local correctness can be used in any program. This is to be contrasted with the latter completeness proof where the local correctness formulas are defined in terms of the given program.

9 Conclusion

We have studied compositional Hoare logics for distributed systems in which communication is asynchronous. In particular, we have presented three sound and (relative) complete proof systems: one for reasoning about partial correctness properties of programs, one for reasoning about non-terminating programs, and one for reasoning about programs in which only deterministic processes are allowed. We have applied the first proof system to a distributed leader election program, in order to demonstrate the applicability of our method. It has been argued that our method is preferable over other methods presented in the literature for reasoning about asynchronously communicating systems. In the future we intend to apply our method to larger programs, using interactive proof checkers like [16].

References

[1] K.R. Apt. *Formal Justification of a Proof System for Communicating Sequential Processes.* Journal of the ACM, Vol. 30, pp. 197–216, 1983.

[2] K.R. Apt. *Ten Years of Hoare's Logic: A Survey-Part I.* Journal of the ACM, Vol. 3, No. 4., pp. 431–483.

[3] Apt K. R., Francez N., and de Roever W.-P., *A proof system for communicating sequential processes.* Journal of the ACM, Vol.2, No. 3, pp. 359–385 (1980).

[4] F.S. de Boer. *Compositionality and Completeness of the Inductive Assertion Method for Concurrent Systems.* Proceedings of IFIP Working Conference on Programming Concepts, Methods and Calculi, San Miniato, Italy, 1994.

[5] F.S. de Boer, J.N. Kok, C. Palamidessi, and J.J.M.M. Rutten. *The failure of failures: Towards a paradigm for asynchronous communication.* Proceedings of Concur '91, LNCS 527, pp. 111–126, Amsterdam, The Netherlands, 1991.

[6] F.S. de Boer and M. van Hulst. *Partial correctness of asynchronously communicating processes.* Proceedings of MFCS'94, 1994.

[7] F.S. de Boer and M. van Hulst. A compositional proof system for asynchronously communicating processes. Proceedings of Mathematics of Program Construction, (MPC), 1995.

[8] L. Bougé. *On the existence of symmetric algorithms to find leaders in networks of communicating sequential processes.* Acta Informatica, Vol. 25, pp. 179–201, 1988.

[9] T. Camp, P. Kearns, and M. Ahuja. *Proof rules for Flush Channels.* IEEE Trans. on Softw. Eng., Vol. 19, No. 4, pp. 366–378, 1993.

[10] N. Francez. *Program Verification.* Addison Wesley, 1992.

[11] N. Francez and F. A. Stomp. *A proof system for asynchronously communicating processes.* Technical Report 722, The Technion, Department of Computer Science, 1993.

[12] N. Francez and F. A. Stomp. *A proof system for asynchronously communicating processes.* Technical Memorandum, AT&T Bell Laboratories, December 1994.

[13] B. Jonsson B. *A model and proof system for asynchronous networks.* Proceedings of the 4^{th} ACM Annual Symp. on Principles of Distr. Comp., pp. 49–58, 1985.

[14] G. Levin and D. Gries. *Proof techniques for communicating sequential processes.* Acta Informatica 15, pp. 281–302, 1981.

[15] S. Owicki and D. Gries. *An axiomatic proof technique for parallel programs.* Acta Informatica 6, pp. 319–340, 1976.

[16] S. Owre, J. Rushby and N. Shankar. *PVS: A Prototype Verification System.* Proceedings of the 1th Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 607, Springer-Verlag, pp. 748–752, 1992.

[17] P.K. Pandya. *Compositional Verification of Distributed Programs.* PhD thesis, Tata Institute of Fundamental Research, Homi Bhabha Road, Bombay 400 005, INDIA, 1988.

[18] P.K. Pandya and M. Joseph. *P-A logic - A Compositional Proof System for Distributed Programs.* Distributed Computing 5, pp. 37–54, 1991.

[19] F.B. Schneider. *Proof Rules for Message Passing, Logics and models of concurrent systems.* NATO ASI Series, Vol. F 13, Springer-Verlag, 1985. LNCS 190, pp. 234–254, 1982.

[20] R.D. Schlichting and F.B. Schneider. *Using message passing for distributed programming, Proof rules and disciplines.* Journal of the ACM, Vol. 6, No. 3, pp. 402–431, 1984.

[21] P. Zhou and J. Hooman. *A proof theory for asynchronously communicating real-time systems.* Proceedings of the 13^{th} Real-Time Systems Symp., 1992.

[22] J. Zwiers. *Compositionality, Concurrency and Partial Correctness.* LNCS 321, 1989.