




A Compositional Proof System for Asynchronously Communicating Processes



F.S. de Boer and M. van Hulst



UU-CS-1996-06
January 1996



Universiteit Utrecht



A Compositional Proof System for Asynchronously Communicating Processes

F.S. de Boer and M. van Hulst *
Utrecht University
Dept. of Comp. Sc.
P.O. Box 80089
3508 TB Utrecht, The Netherlands
email: {frankb,marten}@cs.ruu.nl

Abstract

We present a Hoare logic for distributed systems composed of processes which communicate asynchronously via (unbounded) FIFO buffers. The calculus is based on an assertion language which allows the specification of the communication interface of a process at a high level of abstraction. As such our formalism serves well as a basis for refinement and top-down development of distributed systems composed of asynchronously communicating processes. Moreover, we show that the first-order logic underlying the interface-specification language is decidable, which makes (semi-) automated verification more feasible.

1 Introduction

Hoare logics have been used successfully for reasoning about correctness of a variety of distributed systems [AdB94, AFdR80, HdR86, OG76, Pan88, Zwi88]. In general, proof systems for distributed systems based on some kind of Hoare logic formalize reasoning about communication and synchronization in terms of sequences of communication events called *histories*.

Distributed systems based on synchronous communication allow an elegant compositional proof theory [Zwi88] essentially because there exists a simple logical formulation of the *compatibility* of the local histories of the processes of a system: The local histories are compatible, that is, they can be combined into a global history of the entire system, if they can be obtained as the projection of some sequence of communication events.

On the other hand distributed systems based on asynchronous communication do not allow such a simple criterion: to check the compatibility of the local histories one has in general to consider all possible interleavings [Pan88]. As such its logical formulation involves quantification over histories, and this will obviously complicate the reasoning process.

The recent book on program correctness by Francez [Fra92] contains a section on *non-deterministic* processes which communicate asynchronously via FIFO buffers, featuring a proof system that uses a logic based on *input/output variables* instead of histories. A buffer is logically represented by an input variable which records the sequence of values read from the buffer and by an output variable which records the sequence of values sent to the buffer. The difference between input/output

*This research has been partially supported by the Human Capital and Mobility Network 'EXPRESS'. It has appeared in the proceedings of MPC'95, LNCS 947.

variables and histories is that in the former information of the relative ordering of communication events on different buffers is lost. However, it can be shown that this logic is incomplete for non-deterministic processes; in general, the information expressible by input/output variables only is insufficient to obtain a complete specification of an entire system by composing the local specifications of its constituent processes. As a consequence not all valid correctness formulas are derivable.

In [dBvH94] however, we showed that, provided the programming language is restricted to deterministic processes, a complete compositional proof system can be based on input/output variables only.

So naturally the question arises what is the minimal information we need to add to the input/output variables to obtain a compositional proof system for asynchronously communicating non-deterministic processes. Here the notion of minimal information is defined with respect to the semantic interpretation of a program as a state-transformation, a state being an assignment of values to the program variables.

From a semantical point of view the determination of the minimal information needed to characterize a notion of observables compositionally gives rise to a fully abstract semantics. For asynchronously communicating processes it is well-known that histories are not fully abstract ([Jon89]); for example, the order between inputs (outputs) on different channels is irrelevant with respect to observing the final values of the program variables. Thus the abstraction level of a proof system for asynchronously communicating processes based on histories does not coincide with the appropriate abstraction level of the programming language. With respect to program refinement and top-down program development however it is crucial to be able to specify a program at the right level of abstraction, such that the specification itself does not give rise to irrelevant design choices.

In this paper we introduce a Hoare logic based on an assertion language which allows the specification of the communication interface of a process at the right level of abstraction in the sense that no irrelevant information is included. An interface specification will consist of two separate parts; a *data* component and a *control* component. The data component specifies the FIFO buffers, that is, the sequences of values read from a channel and sent to a channel, respectively. These FIFO buffers are described in terms of input/output variables, as explained above. The control component specifies the relevant order of the communication events in terms of a *abstract history*. An abstract history describes the reactive behavior of a process in terms of the number of inputs (for each channel) it needs in order to be able to produce a certain number of outputs, abstracting from the values communicated. Mathematically, such an abstract history can be formalized by a function which takes as input a *multiset* of channel names and outputs again a multiset of channel names. The number of occurrences of a channel name in the argument then corresponds to the number of required input actions on that channel, whereas the number of occurrences of a channel name in the resulting multiset corresponds to the number of outputs generated on that channel. The distinction between the data and control component of an interface specification introduces a nice separation of concerns which will in general simplify the reasoning process. Moreover, we show that the logic underlying the control component can be formalized in a first-order logic of multisets, and we show that this logic is decidable, which makes (semi-) automated verification more feasible.

This paper is organized as follows: In the following section we introduce the programming language. The proof system is discussed in section 3, and in section 4 we discuss the first-order logic underlying the control component of an interface, and show its decidability. Finally, section 5 contains some conclusions and perspectives.

2 The programming language

In this section, we define the syntax of the programming language. The language describes the behaviour of asynchronously communicating processes. Processes interact only via communication channels which are implemented by (unbounded) FIFO-buffers. A process can send a value along a channel or it can input a value from a channel. The value sent will be appended to the buffer, whereas reading a value from a buffer consists of retrieving its first element. Thus the values will be read in the order in which they have been sent. A process will be suspended when it tries to read a value from an empty buffer. Since buffers are assumed to be unbounded sending values can always take place.

We assume given a set of program variables Var , with typical elements x, y, \dots . Channels are denoted by c, c_1, c_2, \dots ; the set of channels is denoted C .

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

$$\begin{aligned} S ::= & x := e \\ & | c!!e \\ & | S_1; S_2 \\ & | \llbracket_i [b_i; c_i??x_i \rightarrow S_i] \\ & | \star \llbracket_i [b_i; c_i??x_i \rightarrow S_i] \end{aligned}$$

In the assignment statement $x := e$ we restrict for technical convenience to arithmetical expressions e . Sending a value of an (arithmetical) expression e along channel c is described by $c!!e$, whereas storing a value read from a channel c in a variable x is described by $c??x$. Sequential composition is denoted by $;$. The execution of a guarded command $\llbracket_i [b_i; c_i??x_i \rightarrow S_i]$ consists of the selection of a non-empty buffer c_i for which the corresponding boolean guard b_i is true, subsequently a value is read from the buffer and stored in x_i , and control is passed on to the statement S_i . If there exists no non-empty buffer c_i for which the corresponding boolean b_i is true the execution of the guarded command suspends; in case all the boolean guards are false the execution fails. The iterative construct $\star \llbracket_i [b_i; c_i??x_i \rightarrow S_i]$ consists of repeatedly executing the guarded command $\llbracket_i [b_i; c_i??x_i \rightarrow S_i]$ until all the boolean guards b_i are false. In the following some examples will be given using boolean guards $b \rightarrow S$; it is easy to extend the syntax and the semantics of the programming language to include such boolean guards.

Definition 2.2 A parallel program P is of the form $[S_1 \parallel \dots \parallel S_n]$, where we assume the following restrictions: The statements S_i do not share program variables, channels are unidirectional and connect exactly one sender and one receiver.

3 The proof system

3.1 Assertion Language

A compositional proof system for asynchronously communicating processes can be obtained by specifying the interface of a process in terms of a local history which consists of a sequence of

communication records. A communication record $c!!v$ indicates that the value v has been sent along channel c , and $c??v$, on the other hand, indicates that the value v has been read from channel c . The local history of a process itself can be described as the projection of a global history which satisfies the requirement that at any point the number of inputs on a channel does not exceed the number of preceding outputs on the same channel and that the values are read in the order in which they are sent.

Let us illustrate the above informally described proof methodology.

Example 3.1 Consider the sequential processes

$$S_1 = [c??x \rightarrow c??y \parallel true \rightarrow c??y]; d!!0$$

and

$$S_2 = c!!0; [d??z \rightarrow c!!1].$$

After termination of $[S_1 \parallel S_2]$ we have that $y = 0$, since the selection of the branch $c??x \rightarrow c??y$ leads to deadlock. This can be proved by deriving locally the postcondition

$$p_1 = \forall u, v [s_1 = \langle c??u, c??v, d!!0 \rangle \rightarrow y = v] \vee \forall u [s_1 = \langle c??u, d!!0 \rangle \rightarrow y = u]$$

for S_1 and the postcondition

$$p_2 = \exists u [s_2 = \langle c!!0, d??u, c!!1 \rangle]$$

for S_2 . Here s_1 and s_2 are history variables which denote sequences of communication events. Let $compat(s_1, s_2)$ be the compatibility predicate which expresses that there exists an interleaving of s_1 and s_2 such that at any point the number of inputs on a channel does not exceed the number of preceding outputs on the same channel and that the values are read in the order in which they are sent. Then we have that $p_1 \wedge p_2 \wedge compat(s_1, s_2)$ implies $y = 0$. This follows from the fact that there exists no compatible interleaving of the sequences $\langle c??u, c??v, d!!0 \rangle$ and $\langle c!!0, d??u, c!!1 \rangle$.

Note that a history as defined above records the order of the communications on different channels. But we are primarily interested in observing the final values of the program variables, and it can be shown that history variables encode too detailed information about the flow of control of a process. For example, with respect to the final values of the program variables, the order between outputs on different channels is irrelevant. This can be formalized by a fully abstract semantics, which introduces certain abstractions from the order of communications (see also [Jon89, Jos92]): inputs and outputs on different channels can be swapped, and the order between an output followed by an input can be reversed.

A necessary requirement of top-down program development is a specification language which allows the description of a program at the right level of abstraction, such that the specification itself does not give rise to irrelevant design choices. Therefore the above proof methodology, since it is based on the description of the communication interface in terms of sequences of communication records, is not appropriate as a basis for program refinement.

As an example of this observation, consider an implementation of S_1 satisfying $\{s = \epsilon\} S_1 \{s = \langle c!!5, d!!3 \rangle\}$, so initially s is empty and upon termination s contains the communication records $c!!5$ and $d!!3$, in that order. Clearly, $S = c!!5; d!!3$ is a correct implementation of the specification. However it can be shown that $S' = d!!3; c!!5$ is semantically indistinguishable from S , in the sense that there exists no *context* which, when applied to S , gives rise to a different state-transformation than when applied to S' . But S' does *not* satisfy the above (interface) specification. Thus we conclude that the specification includes irrelevant information with respect to the observable behavioural properties of programs. This would prohibit an efficient implementation in the case of two environmental processes S_2 and S_3 which are given by $S_2 = d??x; e!!(x+1)$ and $S_3 = e??y; c??z$. Note

that S_3 needs input from S_2 before it can read channel c ; therefore there is no point for S_1 in sending over channel c first.

To obtain a more appropriate level of abstraction one could simply introduce in the assertion language a binary predicate on histories $R(s_1, s_2)$ which expresses that s_2 is a permutation (allowed by the fully abstract semantics) of s_1 . Thus we would have, for example, the following axiom for calculating the weakest precondition of an output $c!!e$:

$$\{\exists s'[p[s' \cdot c!!e/s] \wedge R(s', s)]\}c!!e\{p\}$$

where s' is a new history variable, and $s' \cdot c!!e$ denotes the result of appending the communication event $c!!v$, with v the value of e , to the sequence denoted by s' ; substitution of a term t for a variable x in an assertion p is denoted by $p[t/x]$. However the obvious disadvantage of this approach is that to express the predicate $R(s_1, s_2)$ one needs either quantification over (finite) sequences of histories or the introduction of recursively defined predicates, since it involves the transitive closure of a relation $s_1 \rightarrow s_2$, which expresses that s_2 can be obtained from s_1 by swapping two elements (as allowed by the fully abstract semantics). As such it highly complicates the reasoning process.

A more manageable and direct description of the interface of a local process can be based on the following observation. Let s be a sequence of communication records, and $[s]$ denote the set of permutations of s resulting from input and output swaps (on different channels). In the following, let I, O denote functions (also called streams) from C to \mathbb{N}^* . Let, for arbitrary s , IO_s be a function which transforms input streams into output streams, such that $IO_s(I) = O$ iff there exists a prefix s' of s ending in an output which is not immediately followed in s by another output, such that I records for each (input) channel the sequence of values read in s' and O records for each (output) channel the sequence of values sent in s' . (In case s does not contain outputs we define $IO_s(I) = O$ iff I records for each input channel of s the sequence of values read and O assigns to each output channel the empty sequence.) We then have that $s' \in [s]$ iff $IO_{s'} = IO_s$ and the projection of s and s' onto any input channel gives the same result (this last condition takes care of trailing inputs, namely those inputs which are not followed by any output). Thus the function IO_s (with the additional information about trailing inputs) is a canonical representative of the equivalence class $[s]$.

Example 3.2 Consider the history $s = \langle c??0, c??1, d!!0, e!!0, f??0, e!!1 \rangle$. Then with the output $e!!0$ of s there corresponds a pair $\langle I, O \rangle$ such that $I(c) = \langle 0, 1 \rangle$, $I(f) = \epsilon$, and $O(d) = \langle 0 \rangle$, $O(e) = \langle 0 \rangle$ (here ϵ denotes the empty sequence). With the last output $e!!1$ there corresponds a pair $\langle I, O \rangle$ such that $I(c) = \langle 0, 1 \rangle$, $I(f) = \langle 0 \rangle$, and $O(d) = \langle 0 \rangle$, $O(e) = \langle 1 \rangle$.

Our proof theory is based on the following representation of the above defined canonical representative of the equivalence class $[s]$ of a sequence of communication records s . We introduce input/output variables $c??$ and $c!!$ which record the values read from the input channel c and the values sent to the output channel c , respectively. Note that these input/output variables abstract from the order of the communications on different channels. We denote the set of variables $c??$ and $c!!$ by \mathcal{I} resp. \mathcal{O} ; their union is denoted \mathcal{IO} . Moreover, the set of variables $c??$ where c is an input channel of process i , and the set of variables $c!!$ where c is an output channel of process i , will be denoted by \mathcal{I}_i and \mathcal{O}_i , respectively. The union of \mathcal{I}_i and \mathcal{O}_i we denote by \mathcal{IO}_i . The additional information about the relevant ordering between communications on different channels can be described as a function h which transforms *multisets* of channel names, i.e. $h \in \mathcal{P}_m(C) \rightarrow \mathcal{P}_m(C)$, where $\mathcal{P}_m(C)$ denotes the set of all finite multisets of elements of C . Typical multisets will be represented by I, O . The idea is that the argument I in $h(I) = O$ contains the number of inputs which are necessary for the outputs of O to take place. For example, $h(\{c, c\}) = \{d, d\}$ indicates that the process needs at least two inputs on c to produce two outputs on d . Thus the number of occurrences of a channel c in $I(O)$ corresponds to the number of input (output) actions on c .

It is not difficult to see that indeed we can thus represent IO_s as a function $h \in \mathcal{P}_m(C) \rightarrow \mathcal{P}_m(C)$ and a function which assigns to each input/output variable a sequence of values. An element of $\mathcal{P}_m(C) \rightarrow \mathcal{P}_m(C)$ we call an *abstract history*.

Example 3.3 Consider the process $c??x; d!!0; c??y; e!!1$. Its abstract history consists of the pairs $\langle \emptyset, \emptyset \rangle, \langle \{c\}, \{d\} \rangle, \langle \{c, c\}, \{d, e\} \rangle$. Thus the pair $\langle \{c, c\}, \{d, e\} \rangle$ for example indicates that the process needs two inputs on c to output once on d and e . The pair $\langle \emptyset, \emptyset \rangle$ indicates that the process is not able to output when no inputs are given. Assuming that first 0 and then 1 has been read from c we have that the input variable $c??$ denotes the sequence $\langle 0, 1 \rangle$, and the output variables $d!!$ and $e!!$ denote the sequences $\langle 0 \rangle, \langle 1 \rangle$, respectively.

Together with the information of its input/output variables the abstract history determines the communication interface of a local process. Note that the input/output variables thus can be considered as the data component of the communication interface, whereas the control component, which specifies the relevant order between the communication events, is described by the abstract history.

The local proof system formalizes reasoning about the correctness of a process in terms of its communication interface as specified by its abstract history and its input/output variables as defined above. Therefore we assume given a many-sorted assertion language (defined formally in the full paper) which, besides the usual vocabulary for describing properties of integers, includes a variable h which denotes an abstract history; input/output variables $c??, c!!$, along with sequence operations like append, prefixing etc; the set C of channel names as constants, and variables which range over multisets of channel names. We include the relation \subseteq of multiset-inclusion, the operation \cap of multiset-intersection, and for each channel name c the unary multiset operation S_c , which consists of adding a copy of c . In order to reason about the abstract history h we introduce terms of the form h_i , where i denotes the index of a local process. A term h_i denotes the *projection* of the (global) history h onto the input/output variables of the process i . Formally, the meaning of a term h_i is given by the following formula

$$\forall I \forall O [h_i(I) = O \leftrightarrow \exists I' \exists O' [h(I') = O' \wedge I = I' \cap \mathcal{I}_i \wedge O = O' \cap \mathcal{O}_i]]$$

where I, I', O, O' are multiset variables. Note that for this notion of projection to be well-defined we have to require that

$$\forall I \forall I' [I \cap \mathcal{I}_i = I' \cap \mathcal{I}_i \rightarrow h(I) \cap \mathcal{O}_i = h(I') \cap \mathcal{O}_i]$$

Finally, we also include expressions of the form $h(e_1) := e_2$, where e_1 and e_2 are multiset expressions, the meaning of which is given by the formula

$$\forall I [I \neq e_i \rightarrow (h(e_1) := e_2)(I) = h(I) \wedge I = e_1 \rightarrow (h(e_1) := e_2)(I) = e_2]$$

This notation for describing an update to the abstract history is derived from the axiomatization of assignments to arrays (see [Gri87]).

In the assertion language quantification is only allowed over integer variables and variables ranging over multisets of channel names. Assertions will be denoted by p, q, \dots

Definition 3.4 Local correctness formulas are of the form

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

where it is implicitly assumed that the free integer variables of p and q are included in the program variables of S , and p and q only refer to the input/output variables of S . These syntactical

conditions ensure that local specifications are free from *interference* (occurrences of the abstract history variable h will be interpreted at this level as the local history of S , only when combining the local specifications we substitute h by h_i , see rule 5).

Global correctness formulas are of the form

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

Both type of correctness formulas are provided with a partial correctness interpretation: for example, $\{p\}S\{q\}$ is valid if every terminating computation of S starting in a state satisfying precondition p results in a state satisfying postcondition q .

3.2 Axioms and Rules

Next we discuss the axioms and rules of the proof system. We have the following well-known axiom for the assignment:

Axiom 1 (*assignment*) $\{p[e/x]\}x := e\{p\}$

Here $p[e/x]$ denotes the result of substituting occurrences of x in p by e . Simultaneous substitution of e_1, \dots, e_n for the variables x_1, \dots, x_n in p will be denoted by $p[e_1/x_1, \dots, e_n/x_n]$.

An input statement $c??x$ is axiomatized by a multiple assignment to the input variable $c??$, the variable x , and a distinguished multiset variable I . The variable I indicates the number of inputs which have been executed so far. So if, for example, $I = \{c, c, d\}$ holds in the current state, this means that until now, the process has read twice from channel c and once from d . The assignment to the input variable $c??$ models the operation of appending the value read to $c??$ and the assignment to the variable x models the storage of the value read in the local memory. The value read is represented by a new integer variable v . Since the value of v is not known locally, the variable v is universally quantified.

Axiom 2 (*input*) $\{\forall v[p[v/x, c?? \cdot v/c??, S_c(I)/I]\}c??x\{p\}$

Here the operation of appending an element to a sequence is denoted by \cdot . Note that $S_c(I)$ denotes the result of adding a copy of c to I . Observe that an input statement does not affect the abstract history.

An output statement $c!!e$ is axiomatized by a multiple assignment to the output variable $c!!$, the abstract history variable h , and a distinguished multiset variable O , which indicates the number of outputs which have been executed so far. The assignment to the output variable $c!!$ models the operation of appending the value sent to $c!!$ and the assignment to the abstract history records the causal dependencies between the outputs executed so far and the inputs that have been executed until now, and which are represented by the multiset variables I and O , respectively.

Axiom 3 (*output*)

$$\{\forall I'[I \subseteq I' \rightarrow p[c!! \cdot e/c!!, (h(I') := S_c(O))/h, S_c(O)/O]\}c!!e\{p\}$$

Here I' denotes a new multiset variable. Note that we update the abstract history for an arbitrary I' which contains I , where I records the number of inputs which have been executed so far. This corresponds with the monotonicity property that if a multiset I of inputs enables a multiset O of

outputs then any $I' \supseteq I$ enables O . The monotonicity property allows abstraction from the order between an output and a subsequent input as illustrated in example 3.5.

The following rules for sequential composition, the guarded statement and the guarded iteration are the usual ones.

Rule 1 (*sequential composition*)

$$\frac{\{p\}S_1\{r\}, \{r\}S_2\{q\}}{\{p\}S_1; S_2\{q\}}$$

Rule 2 (*guarded statement*)

$$\frac{\{p \wedge b_i\}c_i??x_i; S_i\{q\}}{\{p\}\|_i[b_i; c_i??x_i \rightarrow S_i]\{q\}}$$

Rule 3 (*guarded iteration*)

$$\frac{\{p\}\|_i[b_i; c_i??x_i \rightarrow S_i]\{p\}}{\{p\} \star \|_i[b_i; c_i??x_i \rightarrow S_i]\{p \wedge \bigwedge_i \neg b_i\}}$$

Moreover we have the following consequence rule.

Rule 4 (*Local consequence rule*)

$$\frac{\models p \rightarrow p', \{p'\}S\{q'\}, \models q' \rightarrow q}{\{p\}S\{q\}}$$

We have the following rule for parallel composition.

Rule 5 (*parallel composition*)

$$\frac{\{p_i \wedge \text{Init}_i\}S_i\{q_i\}, (q_i' \wedge \forall I'[h(I') \neq \emptyset \rightarrow I' \subseteq I]) \rightarrow q_i}{\{\bigwedge_i p_i[h_i/h]\}S_1 \parallel \dots \parallel S_n \{\bigwedge_i q_i[h_i/h]\}}$$

Here Init_i denotes the assertion $h = \emptyset \wedge I = \emptyset \wedge O = \emptyset \wedge \bigwedge_{c?? \in \mathcal{I}_i} (c?? = \epsilon) \wedge \bigwedge_{c!! \in \mathcal{O}_i} (c!! = \epsilon)$. (In $h = \emptyset$ the function which assigns to each multiset the empty multiset is denoted by \emptyset , whereas in $I = \emptyset$ ($O = \emptyset$), \emptyset denotes the empty multiset. The empty sequence is denoted by ϵ .) The additional information expressed by $\forall I'[h(I') \neq \emptyset \rightarrow I' \subseteq I]$ ensures that the choices for the input set in the output axiom are consistent with the final set of inputs. It is implicitly assumed that p_i and q_i do not contain the logical variables I and O (these variables are only used locally to generate the abstract history, globally they are irrelevant). When combining the local specifications in the conclusion of the rule we substitute the abstract history variable h by its corresponding projection.

We now arrive at a suitable point to illustrate the use of the output axiom.

Example 3.5 Let $S_1 = c!!0; d??x$ and $S_2 = [\text{true} \rightarrow c!!0; d??x \| d??x \rightarrow c!!0]$. It is not difficult to see that S_1 and S_2 are observationally equivalent, that is in every context they give rise to the same state-transformation. Using the above calculus we can derive the postcondition $\exists I'[\emptyset \subseteq I' \wedge h = \{\langle I', \{c\} \rangle\}] \wedge I = \{d\}$ for S_1 . Thus using the additional information $\forall I'[h(I') \neq \emptyset \rightarrow I' \subseteq I]$ we derive the postcondition $h = \{\langle \emptyset, \{c\} \rangle\} \vee h = \{\langle \{d\}, \{c\} \rangle\}$, which is also satisfied by S_2 . In other words, S_1 and S_2 are provably equivalent.

The following rule allows to strengthen the postcondition of a program with the information that the sequence of values read from a channel is a prefix of the sequence of values sent.

Rule 6 (FIFO rule)

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

Here \preceq denotes the prefix relation on sequences.

Next we introduce a rule which allows to strengthen the postcondition of a program with additional information about the global history.

Rule 7 (Compatibility predicate)

$$\frac{\{p\}P\{q\}}{\{p\}P\{q \wedge \text{compat}(h)\}}$$

Here the compatibility predicate $\text{compat}(h)$ is defined as follows:

$$\forall I(I \neq \emptyset \wedge h(I) \neq \emptyset \rightarrow \exists I' \subset I(I \subseteq h(I')))$$

It roughly expresses that each input of h must be generated by a proper subset. In the full paper we show on the basis of a formally defined semantics that this requirement ensures that there exists an execution of P which complies with the condition that on each channel, each value should be written before it is read. This is best explained by a simple example.

Example 3.6 Consider again the sequential processes

$$S_1 = [c??x \rightarrow c??y] \text{true} \rightarrow c??y; d!!0$$

and

$$S_2 = c!!0; [d??z \rightarrow c!!1]$$

After termination of $[S_1 \parallel S_2]$ we have that $y = 0$. This can be proved by deriving locally the postcondition

$$p_1 = (h = \{\{c, c\}, \{d\}\} \wedge \forall u, v [c?? = \langle u, v \rangle \rightarrow y = v]) \vee \\ (h = \{\{c\}, \{d\}\} \wedge \forall u [c?? = \langle u \rangle \rightarrow y = u])$$

for S_1 and the postcondition

$$p_2 = (h = \{\{\emptyset, \{c\}\}, \{\{d\}, \{c, c\}\}\} \wedge c!! = \langle 0, 1 \rangle)$$

for S_2 . Then we have that $(p_1 \wedge p_2 \wedge \text{compat}(h) \wedge c?? \preceq c!!) \rightarrow y = 0$. This can be seen as follows: we consider the following two cases. Let h_1 denote the local history of S_1 and h_2 the local history of S_2 :

1. $h_1 = \{\{c, c\}, \{d\}\}$: there are a number of possible global histories that obey the projection restriction, i.e. are such that their respective projections yield h_1 and h_2 . We only list a few below:

- $h = \{\{\emptyset, \{c\}\}, \{\{d\}, \{c, c\}\}, \{\{c, c\}, \{d\}\}\}$
- $h = \{\{\{c, c\}, \{c, d\}\}, \{\{d\}, \{c, c\}\}\}$

- $h = \{\langle\{c, c\}, \{c, d\}\rangle, \langle\{d\}, \{c, c\}\rangle, \langle\emptyset, \{c\}\rangle\}$
- $h = \{\langle\emptyset, \{c\}\rangle, \langle\{c, c, d\}, \{c, c, d\}\rangle\}$
- $h = \{\langle\{c, c\}, \{c, d\}\rangle, \langle\{c, c, d\}, \{c, c, d\}\rangle, \langle\emptyset, \{c\}\rangle, \langle\{d\}, \{c, c\}\rangle, \langle\{c, c\}, \{d\}\rangle\}$

None of these possible global histories is compatible. For instance, in the first case the pair $\langle\{c, c\}, \{d\}\rangle$ has no predecessor as described in the definition of *compat* because the multiset $\{c, c\}$ cannot be generated. Note that $\{d\}$ generates $\{c, c\}$, but $\{d\} \not\subseteq \{c, c\}$. Without this requirement that $\{c, c\}$ should be generated by a proper subset we thus would allow circularities: $\{d\}$ generates $\{c, c\}$ and $\{c, c\}$ in its turn generates $\{d\}$.

2. $h_1 = \{\langle\{c\}, \{d\}\rangle\}$: again we list some representative possible global histories:

- $h = \{\langle\emptyset, \{c\}\rangle, \langle\{d\}, \{c, c\}\rangle, \langle\{c\}, \{d\}\rangle\}$
- $h = \{\langle\{c\}, \{c, d\}\rangle, \langle\{d\}, \{c, c\}\rangle\}$
- $h = \{\langle\emptyset, \{c\}\rangle, \langle\{c, d\}, \{c, c, d\}\rangle\}$
- $h = \{\langle\{c\}, \{c, d\}\rangle, \langle\{c, d\}, \{c, c, d\}\rangle, \langle\emptyset, \{c\}\rangle\}$

Out of these possible h 's, for instance the last one listed here fulfils the *compat*-demands.

Therefore we conclude that $\text{compat}(h)$ implies that the first disjunct of p_1 should be discarded of, and hence from $\forall u[c?? = \langle u \rangle \rightarrow y = u]$, $c!! = \langle 0, 1 \rangle$ and $c?? \preceq c!!$ we conclude $y = 0$.

We conclude the exposition of the proof system with the global consequence rule and the substitution rule.

Rule 8 (*global consequence rule*)

$$\frac{\models p \rightarrow p', \{p'\}P\{q'\}, \models q' \rightarrow q}{\{p\}P\{q\}}$$

Rule 9 (*Substitution rule*)

$$\frac{\{p\}P\{q\}}{\{p[z/x]\}P\{q\}}$$

where x is a variable not occurring in P or q , and z is a variable of the same sort as x .

4 Multiset Logic

In this section, we look a bit closer at the subset of the logic that is needed in order to formalize reasoning about multisets. Formally, we introduce the following logic of multisets: Consider a structure $\langle \mathcal{P}_m(C), S_{c_1}, \dots, S_{c_n}, \leq, \cup, \cap, \emptyset \rangle$, where $\mathcal{P}_m(C)$ denotes the set of multisets of elements of the finite set $C = \{c_1, \dots, c_n\}$; S_{c_i} , for $c_i \in C$, denotes the —unary— c_i -successor function; \leq is the subset-relation between multisets and \emptyset denotes the empty multiset. Furthermore, we add two binary operators on multisets: the least upperbound (lub, denoted by \cup) and the greatest lowerbound (glb, denoted by \cap). Note that the glb of two elements of $\mathcal{P}_m(C)$ corresponds with the usual multiset intersection, whereas the least upperbound of two multisets x and y of $\mathcal{P}_m(C)$ corresponds to the multiset which contains for each element the maximal number of occurrences in x and y . Hence the least upperbound operation differs from the operation of multiset addition,

which *adds* the number of occurrences: consider for instance the multisets $\{c, d\}$ and $\{c, c, d\}$. Their least upperbound is $\{c, c, d\}$ whereas their multiset addition is $\{c, c, c, d\}$.

Given multiset variables x, y, \dots ranging over $\mathcal{P}_m(C)$, terms and formulas in this logic are given by

$$t ::= \emptyset \mid x \mid S_{c_i}(t) \mid t_1 \cup t_2 \mid t_1 \cap t_2$$

$$\varphi ::= t_1 = t_2 \mid t_1 \leq t_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists x[\varphi].$$

The logic of multisets includes for each successor function its logic of discrete time, and consists of the following set T of axioms.

Firstly, we have the axioms concerning the ordering relation:

MS-REFL	$\forall x[x \leq x]$
MS-ANTISYM	$\forall x \forall y[x \leq y \wedge y \leq x \rightarrow x = y]$
MS-TRANS	$\forall x \forall y \forall z[x \leq y \wedge y \leq z \rightarrow x \leq z]$
MS-LEASTEL	$\forall x[\emptyset \leq x]$
MS-LUB	$\forall x \forall y[x \leq (x \cup y) \wedge y \leq (x \cup y)$ $\wedge \forall z[x \leq z \wedge y \leq z \rightarrow (x \cup y) \leq z]$
MS-GLB	$\forall x \forall y[x \geq (x \cap y) \wedge y \geq (x \cap y)$ $\wedge \forall z[x \geq z \wedge y \geq z \rightarrow (x \cap y) \geq z]$

Then, we have axioms concerning the successor functions:

MS-INJ	$\forall x \forall y[S_{c_i}(x) = S_{c_i}(y) \rightarrow x = y]$
MS-IMMSUC	$\forall x[x \leq S_{c_i}(x) \wedge x \neq S_{c_i}(x)$ $\wedge \neg \exists y[x \leq y \wedge y \leq S_{c_i}(x) \wedge x \neq y \wedge y \neq S_{c_i}(x)]]]$
MS-PRED	$\forall x[S_{c_i}(\emptyset) \leq x \rightarrow \exists y[S_{c_i}(y) = x]]]$
MS-SUCDIFF	$\forall x[S_{c_i}(x) \neq S_{c_j}(x)]$ (for $i \neq j$)
MS-COMM	$\forall x[S_{c_i}(S_{c_j}(x)) = S_{c_j}(S_{c_i}(x))]$ (successors commute)

The following axiom allows to factorize a term t without occurrences of the \cup and \cap operations into a union of terms t_i , where t_i only contains the S_{c_i} successor function (from here on, $S_{c_i}^{k_i}(x)$ denotes the term resulting from applying S_{c_i} k_i times to x):

$$\text{MS-SPLIT} \quad \forall x[S_{c_1}^{k_1} \dots S_{c_n}^{k_n}(x) = \bigcup_i S_{c_i}^{k_i}(x)]$$

Next we have an axiom (actually a family of axioms for $n, m \geq 0$ and each successor function S_{c_i}) which states in a sense that terms which only contain the successor operation S_{c_i} are totally ordered:

$$\text{MS-TOT} \quad \forall x \forall y[S_{c_i}^n(x) \leq x \cup S_{c_i}^m(y) \vee S_{c_i}^m(y) \leq y \cup S_{c_i}^n(x)]$$

Note that the occurrence of x in the right hand side of $S_{c_i}^n(x) \leq x \cup S_{c_i}^m(y)$ causes the inequality to be valid automatically for the non- c_i elements of x . In this way, x serves as a 'mask' which filters out the irrelevant ordering information.

The following axiom allows to reduce basic formulas of the form $S_{c_j}^n(x) \leq \bigcup_i S_{c_i}^{k_i}(y)$ to a conjunction of formulas $t_1^i \leq t_2^i$, where t_1^i and t_2^i contain only the S_{c_i} successor function:

$$\text{MS-COMP} \quad \forall x \forall y [S_{c_j}^n(x) \leq \bigcup_i S_{c_i}^{k_i}(y) \longleftrightarrow \bigwedge_{i \neq j} (S_{c_i}(x) \leq x \cup S_{c_i}^{k_i+1}(y)) \\ \wedge (S_{c_j}^{n+1}(x) \leq x \cup S_{c_j}^{k_j+1}(y))]]$$

Finally we have an axiom which allows to distribute existential quantification over a conjunction of formulas which do not share occurrences of the same successor function: let $Succ(\phi)$ denote the set of successor functions occurring in ϕ .

$$\text{MS-CONSPLIT} \quad \exists x [\phi_1 \wedge \phi_2] \longleftrightarrow (\exists x [\phi_1] \wedge \exists x [\phi_2]) \\ (\text{provided } Succ(\phi_1) \cap Succ(\phi_2) = \emptyset)$$

We have the following theorem:

Theorem 4.1 *The theory T admits elimination of quantifiers, i.e. for any formula ϕ there exists a quantifier free formula ψ such that $T \vdash \phi \leftrightarrow \psi$; hence T is complete, and thus the theory of the structure $(\mathcal{P}_m(C), S_{c_1}, \dots, S_{c_n}, \leq, \cup, \cap, \emptyset)$ is decidable.*

The proof is given in the full paper. Globally, we proceed as follows: it suffices to show that for any quantifier free formula ϕ the formula $\exists x \phi$ is equivalent (with respect to the above axioms) to a quantifier free formula. So let ϕ be a quantifier free formula. It is not difficult to transform, using the axioms, ϕ into a equivalent formula ψ which is constructed from basic formulas of the form $S_{c_i}^n(t) \leq t \cup S_{c_i}^m(t')$, where t and t' are either a variable or the constant \emptyset . Transform the resulting formula ψ into disjunctive normal form, and distribute the $\exists x$ over the disjunction. Now each disjunct is of the form $\exists x \bigwedge_i \phi_i$, where each ϕ_i is a conjunction of basic formulas which only contain the successor function S_{c_i} . Using the last axiom we can distribute the $\exists x$ over the conjuncts ϕ_i . Thus we obtain formulas $\exists x \phi_i$, where ϕ_i only contains the successor function S_{c_i} . But to these formulas we can apply the well-known theorem which states that the theory of one successor function admits elimination of quantifiers. Note that indeed our axioms contain for each successor function S_{c_i} the standard theory of one successor function, consisting of the axioms MS-REFL, MS-ANTISYM, MS-TRANS, MS-LEASTEL, MS-IMMSUC, MS-PRED and MS-TOT.

5 Conclusion and Future Work

The present paper presents a proof methodology which supports reasoning about the correctness of asynchronously communicating processes at an abstraction level at least as high as that of the programming language.

In the full paper we have proved that the proof system is *sound*, i.e. every derivable correctness formula is valid, and (*relative*) *complete*, i.e. every valid correctness formula is derivable. The formal justification is given with respect to a formally defined fully abstract semantics. Due to a close correspondence of the proof rules and the semantics, the proofs of soundness and completeness are fairly straightforward.

Because of its high level of abstraction the proof system provides a suitable basis for top-down program development along the lines of a refinement calculus as defined in [Bac80]. Such a refinement calculus for asynchronously communicating processes we are currently investigating.

Additionally, to study the practical usage of the presented proof methodology we plan to investigate some case studies.

Exploiting the fact that our multiset logic is decidable, we can transform our rules in a format amenable for use by verification tools; in particular, we use PVS (Prototype Verification System, see [SOS92]) to study the impact of automated theorem proving techniques in our setting. PVS allows for interactive generation of proofs, defining of strategies and decision procedures for arithmetic, thus alleviating the user from trivial but tedious tasks.

In this way, using the proof system from [dBvH94], we already verified the problem of determining a network topology by means of a so-called heartbeat algorithm (see also [And91]). The algorithm entails for each participating process to repeatedly send and receive information about network links to its direct neighbours, which finally leads to a complete picture of the networks topology. (The topology information can be used by the processes for instance to route their messages efficiently along the links of the network.)

Finally, we are currently investigating the decidability of the extension of our multiset logic with the operation of multiset addition, denoted by \oplus and defined by:

$$\begin{aligned}x \oplus \emptyset &= x \\x \oplus S_{c_i}(y) &= S_{c_i}(x \oplus y)\end{aligned}$$

References

- [AdB94] P.H.M. America and F.S. de Boer. Reasoning about dynamically evolving process structures. *Formal Aspects of Computing*, 6:269–316, 1994.
- [AFdR80] K.R. Apt, N. Francez, and W.-P. de Roever. A proof system for communicating sequential processes. *ACM-TOPLAS*, 2(3):359–385, 1980.
- [And91] Gregory R. Andrews. *Concurrent Programming, Principles and Practice*. The Benjamin/Cummings Publishing Company, Inc., 1991.
- [Bac80] R.J.R. Back. *Correctness Preserving Program Refinements: Proof Theory and Applications*. Number 131 in Mathematical Centre Tracts. Mathematical Centre, Amsterdam, 1980.
- [dBvH94] F.S. de Boer and M. van Hulst. A proof system for asynchronously communicating deterministic processes. In B. Rovani, I. Privarova and P. Ružička, editors, *Proc. MFCS '94*, volume 841 of *Lecture Notes in Computer Science*, pages 256–265. Springer-Verlag, 1994.
- [Fra92] N. Francez. *Program Verification*. Addison Wesley, 1992.
- [Gri87] David Gries. *The Science of Programming*. Texts and Monographs in Computer Science. Springer, 4th print edition, 1987.
- [HdR86] J. Hooman and W.-P. de Roever. The quest goes on: a survey of proof systems for partial correctness of CSP. In *Current trends in concurrency*, volume 24 of *Lecture Notes in Computer Science*, pages 343–395. Springer-Verlag, 1986.
- [Jon89] B. Jonsson. A fully abstract trace model for dataflow networks. In *Proc. POPL '89*, 1989.
- [Jos92] M.B. Josephs. Receptive process theory. *Acta Informatica*, 29, 1992.
- [OG76] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6:319–340, 1976.

- [Pan88] P.K. Pandya. *Compositional Verification of Distributed Programs*. PhD thesis, Tata Institute of Fundamental Research, Homi Bhabha Road, Bombay 400 005, INDIA, 1988.
- [SOS92] J. Rushby S. Owre and N. Shankar. PVS: A prototype verification system. In *11th Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer-Verlag, 1992.
- [Zwi88] J. Zwiers. *Compositionality, Concurrency and Partial Correctness*. PhD thesis, Technical University Eindhoven, 1988.

**Recent technical reports from the Department of Computer Science,
Utrecht University**

Requests for Technical Reports can be directed to

Librarian
Department of Computer Science
Utrecht University
P.O. Box 80.089
Utrecht University
the Netherlands
Or by e-mail: guus@cs.ruu.nl

Many technical reports are also available via ftp ([ftp ftp.cs.ruu.nl](ftp://ftp.cs.ruu.nl), login as **anonymous** or **ftp**, directory: [/pub/RUU/CS/techreps](ftp://ftp.cs.ruu.nl/pub/RUU/CS/techreps)).

The archive of technical reports is also accessible via the World Wide Web,
URL-address:

<http://www.cs.ruu.nl/docs/research/publication/TechRep.html>

- [UU-CS-1995-01] H.L. Bodlaender, R.G. Downey, M.R. Fellows, and H.T. Wareham. The parameterized complexity of sequence alignment and consensus.
- [UU-CS-1995-02] H.L. Bodlaender and D.M. Thilikos. Treewidth and small separators for graphs with small chordality.
- [UU-CS-1995-03] H.L. Bodlaender, J.S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, and Zs. Tuza. Rankings of graphs.
- [UU-CS-1995-04] T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings.
- [UU-CS-1995-05] J. van Leeuwen and R.B. Tan. Compact routing methods: A survey.
- [UU-CS-1995-06] P.K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf. Constructing levels in arrangements and higher order voronoi diagrams.
- [UU-CS-1995-07] I.S.W.B. Prasetya and S.D. Swierstra. Formal design of self-stabilizing programs.
- [UU-CS-1995-08] B. van Linder, W. van der Hoek, and J.-J. Ch. Meyer. Seeing is believing and so are hearing and jumping.
- [UU-CS-1995-09] P.D. Bruza and T.W.C. Huibers. How nonmonotonic is aboutness?
- [UU-CS-1995-10] W. Fokkink and H. Zantema. A complete equational axiomatization for $\text{bpa}\delta\epsilon$ with prefix iteration.
- [UU-CS-1995-11] N.B. Peek and L.C. van der Gaag. A case-based filter for diagnostic belief networks.
- [UU-CS-1995-12] M. de Berg and K.T.G. Dobrindt. On levels of detail in terrains.

- [UU-CS-1995-13] J. van Leeuwen, N. Santoro, J. Urrutia, and S. Zaks. Guessing games, binomial sum trees and distributed computations in synchronous networks.
- [UU-CS-1995-14] J. Vleugels and M. Overmars. Approximating generalized voronoi diagrams in any dimension.
- [UU-CS-1995-15] H.L. Bodlaender and B. de Fluiter. Intervalizing k -colored graphs.
- [UU-CS-1995-16] M. Flammini, J. van Leeuwen, and A. Marchetti-Spaccamela. The complexity of interval routing on random graphs.
- [UU-CS-1995-17] T. Arts and H. Zantema. Termination of constructor systems using semantic unification.
- [UU-CS-1995-18] J.J.Ch. Meyer and W. van der Hoek. A modal contrastive logic: The logic of 'but' (revised version of uu-cs-1994-07).
- [UU-CS-1995-19] J.J.Ch. Meyer and W. van der Hoek. Modal logics for representing incoherent knowledge.
- [UU-CS-1995-20] H.L. Bodlaender and B. de Fluiter. On intervalizing k -colored graphs for DNA physical mapping.
- [UU-CS-1995-21] M. de Berg. Trends and developments in computational geometry.
- [UU-CS-1995-22] P. Švestka and M.H. Overmars. Probabilistic path planning.
- [UU-CS-1995-23] M.J. Druzdzel and L.C. van der Gaag. Elicitation of probabilities for belief networks: Combining qualitative and quantitative information.
- [UU-CS-1995-24] M.L. Wessels. On belief networks and diagnosis.
- [UU-CS-1995-25] H.L. Bodlaender and T. Hagerup. Parallel algorithms with optimal speedup for bounded treewidth.
- [UU-CS-1995-26] M. de Berg, M. van Kreveld, and S. Schirra. A new approach to subdivision simplification.
- [UU-CS-1995-27] T. Biedl, G. Kant, and M. Kaufmann. On triangulating planar graphs under the four-connectivity constraint.
- [UU-CS-1995-28] M. de Berg, H. Everett, and L.J. Guibas. The union of moving polygonal pseudodiscs - combinatorial bounds and applications.
- [UU-CS-1995-29] K. Aardal and S. van Hoesel. Polyhedral techniques in combinatorial optimization i : Theory.
- [UU-CS-1995-30] P. d'Altan, J.-J.Ch. Meyer, and R.J. Wieringa. An integrated framework for ought-to-be and ought-to-do constraints.
- [UU-CS-1995-31] J. Verriet. Scheduling uet, uct dags with release dates and deadlines.
- [UU-CS-1995-32] T. Arts. A technique for automatically proving termination of constructor systems.
- [UU-CS-1995-33] A.F. van der Stappen and M.H. Overmars. Motion planning in environments with low obstacle density.
- [UU-CS-1995-34] H. Bodlaender, T. Kloks, D. Kratsch, and H. Müller. Treewidth and minimum fill-in on d -trapezoid graphs.

- [UU-CS-1995-35] K. Aardal. Capacitated facility location: Separation algorithms and computational experience.
- [UU-CS-1995-36] H.L. Bodlaender, M.R. Fellows, M.T. Hallet, H.T. Wareham, and T.J. Warnow. The hardness of problems on thin colored graphs.
- [UU-CS-1995-37] H.L. Bodlaender and B. de Fluiter. Reduction algorithms for graphs with small treewidth.
- [UU-CS-1995-38] J.-J.Ch. Meyer and J.C. van Leeuwen. Possible world semantics for analogous reasoning.
- [UU-CS-1995-39] C. Witteveen and W. van der Hoek. Semantic based theory revision in nonmonotonic logic.
- [UU-CS-1995-40] B. van Linder. A dynamic logic of iterated belief change.
- [UU-CS-1995-41] K. Aardal, M. Labbé, J. Leung, and M. Queyranne. On the two-level uncapacitated facility location problem.
- [UU-CS-1995-42] K. Aardal and S. van Hoesel. Polyhedral techniques in combinatorial optimization ii: Computations.
- [UU-CS-1996-01] I.S.W.B. Prasetya. Formalizing unity with hol.
- [UU-CS-1996-02] H.L. Bodlaender. A partial k -arboretum of graphs with bounded treewidth.
- [UU-CS-1996-03] H. Zantema and A. Geser. Non-looping rewriting.
- [UU-CS-1996-04] Th.W.C. Huibers, M. Lalmas, and C.J. van Rijsbergen. Information retrieval and situation theory.
- [UU-CS-1996-05] F.S. de Boer, N. Francez, M. van Hulst, and F.A. Stomp. A proof theory of asynchronously communicating sequential processes.
- [UU-CS-1996-06] F.S. de Boer and M. van Hulst. A compositional proof system for asynchronously communicating processes.

