An Epistemic Proof System for Parallel Processes *

(extended abstract)

M. van Hulst and J.-J.Ch. Meyer

University of Nijmegen †
Dept. of Mathematics and Computer Science
Toernooiveld 1
6525 ED Nijmegen
The Netherlands
email: marten@cs.ruu.nl

Abstract

A proof system for the correctness of parallel programs using knowledge in distributed systems is presented and proven sound and relatively complete with respect to a formal Kripke style semantics that supports truly concurrent computations.

1 Introduction

Because of its practical importance and technical complexity, the verification of parallel programs has been a major topic of investigation for many years ([OG76, AFdR80]). In the last decade attention has primarily been focused on finding compositional methods to prove the correctness of parallel programs, e.g. [Zwi88].

On the other hand in the last decade—in a slightly different context— epistemic logic [HM85, MvdH], a logic to reason about the knowledge of agents, has been advocated to be used as a tool to verify a special class of programs running on distributed systems, viz. protocols (e.g. [HZ87]), aiming to correctly send messages across the distributed system (a network of processors).

In this paper we aim to employ epistemic logic to reason about parallel programs more in general, in the sense of the first paragraph above. The general idea here is that program statements that have to do with receiving a message create a kind of ignorance on the part of the process/agent involved that can be resolved by synchronization. The receiving process and the sending one together know what has been sent/received. This can be expressed by means of an epistemic logic.

The reason why we believe epistemic logic is advantageous for reasoning about parallel programs is the fact that it enables us to prefix assertions with a modal (epistemic) operator that states what the agent (processor) knows. This means that our approach is modular in the sense that assertions can be proven correct in a local context, after which they can be lifted to a global context without loosing the information where it came from, i.e. that the assertion at hand pertains to the knowledge of a particular agent.

We then may 'group' knowledge of the agents engaging in a parallel execution by means of a notion of group or distributed knowledge. By sharing the knowledge of these agents we finally arrive at the global assertions we are

*A full version appeared as Technical Report No. 93-17.

†authors' current address: Utrecht University, Dept. of Comp. Sc., P.O. Box 80089, 3508 TB Utrecht, The Netherlands

interested in concerning the behaviour of the entire system.

In the paper, a proof system will be presented which allows one to reason about and proof assertions concerning knowledge of (groups of) processes in a distributed system. For example, the formalism enables one to state and prove assertions like "$K_i(x = 3)$" (agent i knows that the value of x is 3 in the current situation) and "$K_{\{1,2,3\}}(x = y)$" (it is group knowledge of the processors 1,2 and 3 that x equals y).

There have been other attempts at defining and proving the notion of knowledge in distributed systems, of which we mention [KT86]; they used an interleaving semantics, as opposed to our poset semantics (a form of true concurrency semantics, cf. [BRR89]), and, moreover, their proof method is based on the well-known proof systems of [AFdR80] and [OG76], and is therefore not compositional.

This paper must be viewed as a sequel to [HHM93]; the ideas of that paper are elaborated further in the concrete context of a simple programming language, culminating in a sound and complete proof system for this language.

The proof system essentially consists of two layers, one dealing with the usual reasoning about a process in isolation, and the other dealing with programs possibly consisting of more programs in parallel. The first layer concerns the realm of set semantics, the second concerns that of Kripke-semantics.

The proof of completeness of the proof system also consists of two parts: the usual part, proving derivability of local assertions of single processes (e.g. $\{x = 1\}l : x := 5\{x = 5\}$) and a "knowledge"-part, dealing with formulae like $\{true\}P_1 :: S_1 \parallel P_2 :: S_2\{K_2\varphi_2\}$ for some local assertion φ_2. With respect to the latter part, the completeness achieved is somewhat special in that it is only shown that all triples with a postcondition consisting of an epistemic formula can be deduced, leaving out of consideration 'classical' triples, i.e. triples with a non-epistemic postcondition. It reflects the fact that we are only interested in assertions that are known by the agents involved. This feature renders our approach modular with respect to agents, so that we can speak of an agent-oriented approach to program correctness.

The programming language used here is of a striking simplicity: for example there is no recursion included in its definition. We would like to emphasize that this is no necessary restriction imposed by our epistemic approach, and we expect that many extensions can be made to the language without too much effort. The reason for our choice is a clearer focus on the main contribution of the paper: the introduction of epistemic operators in the proof system.

2 Syntax

We will now give the syntax of our language, which is a variant of CSP ([Hoa78]). Assume a set CHAN of communication channels, and a set VAR of variables, both finite. We will denote the variables of a statement (or program for that matter) by and VAR(S) respectively.

i-Expression	$e_i ::$	$\alpha \mid x_i \mid e_i + e_i' \mid e_i - e_i' \mid e_i \times e_i'$
i-Basic command	$s_i ::$	$skip \mid x_i := e_i \mid c!e_i \mid c?x_i$
i-Statement	$S_i ::$	$l : s_i \mid S_i; S_i' \mid S_i[]S_i'$
Program	$PR ::$	$[P_1 :: S_1 \parallel \cdots \parallel P_n :: S_n]$

The indices indicate the type of an expression or variable. To avoid cumbersome notation, they are omitted when derivable from the context or of no importance. Furthermore, we have various syntactic restrictions guaranteeing that each channel in CHAN is unidirectional and connected to exactly 2 processes.

In this syntax, α is a constant, *skip* is the null command, $x := e$ denotes an assignment, $c!e$ denotes 'output the value of e on channel c', and $c?x$ denotes 'input a value from channel c and store it in x'. Typically, basic commands will be denoted by s. Furthermore, a statement S is a labeled basic command, and the operations ; and [] are sequential composition and nondeterministic choice, respectively. A program PR finally is a parallel composition of a number of processes, where a process is a statement labeled with an –indexed– process label P.

3 Semantics

In this section, we prepare grounds for the use of first order epistemic logic, of which our assertion language is an instance. As a model for our logic, we use the well-known notion of *Kripke models.*

In order to do so, we first define the *view* semantics for individual processes, which consists of a set semantics. Then, we proceed to define the Kripke semantics of programs, thereby using the view semantics of an individual process to define the reachability (possible worlds) relation of that process. We then are able to interpret Hoare-triples containing (also) epistemic assertions, to be defined later on. For a more general introduction to this approach, we refer to [HHM93] which provides some background on this.

Remark We will not be concerned with deadlock behaviour of programs, for reasons of simplicity. We intent to do so in the near future.

3.1 The Semantical Domain

In the following, let PR be a program. We define $\mathcal{S}$ with typical element σ to be the set of valuations of $\mathsf{VAR}(PR) \bigcup \mathsf{LVAR} \bigcup \mathsf{LHVAR}$, where LVAR with typical elements x, x' is the set of logical variables, and LHVAR with typical elements hx, hx' is the set of logical variables . σ maps elements of VAR and logical variables onto the domain $\mathbb{Z}$, with typical element α, and logical history variables hx onto $\mathcal{H}$, to be defined shortly. We will use the notation $\sigma[\alpha/x]$ to denote the valuation function which is equal to σ but for the valuation of x, which is α. Similarly we use $\sigma[h/hx]$ where $h \in \mathcal{H}$.

Next we define the set of program labels $\Lambda = \{l, \langle \alpha, c, ?, m\rangle, \langle \alpha, c, l, ?\rangle, \langle \alpha, c, l, m\rangle | l, m \text{ appear in } PR\}$. Thus, there are two possible formats for labels, with as intended meaning that a 'simple' label l reflects some internal action, whereas a 'quadruple' label describes a communication, or an attempt at a communication. The appearance of the question marks in the quadruple labels is due to incomplete information: they typically occur in the semantics of communication statements in isolation. In the sequel, both simple and quadruple labels will be denoted by $\lambda, \mu, \ldots$ when we are not interested in their inner structure. The set of i-labels, Λ_i, consists of the simple labels from Λ which appear syntactically in P_i and the quadruple labels from Λ which contain a simple label which appears syntactically in P_i. The set of global labels, Λ_0, consists of the labels from Λ that do not contain a question mark. Finally, we define the set of histories $\mathcal{H}$, with typical element h, as the set of posets $(H, <)$ over Λ_0. Note that, as labels are unique, and our language contains no recursion, the notion of poset causes no problems. The basic building blocks of our semantical domain, *points*, are pairs $\langle \sigma, h\rangle \in \mathcal{S} \times \mathcal{H}$, where the first component of a pair is a state, and the second component describes the history via which the state in the first component was reached. We sometimes abuse notation and write $\lambda \in h$ when we mean $\lambda \in H$, where $h = (H, <)$. We will use the dot "$\cdot$" to denote the concatenation operation between histories. Furthermore, when h is a poset with only one element, we may use that element as denotation for h, and ϵ denotes the empty history.

In the next section, individual processes will be provided with a semantics, which we will call *view*-semantics, because of the fact that it provides local processes with certain information of the whole semantics. Therefore, next to the full domain we will use a kind of *local* domain for each process P_i: $\mathcal{S}_i \times \mathcal{H}_i$, where $\mathcal{S}_i$ is defined analogously to $\mathcal{S}$ but for the fact that valuations are now restricted to variables of process i and $\mathcal{H}_i$ are posets over Λ_i. Elements of $\mathcal{S}_i$ map logical history variables onto $\mathcal{H}_i$.

3.2 View Semantics for Statements

In this section we will provide some of the semantic clauses giving meaning (assigning views) to statements. Assume in the following that S is a statement of process i. The semantical operator is typed as follows: $[\![\cdot]\!]_v : S \times \wp(\mathcal{S}_i \times \mathcal{H}_i) \to \wp(\mathcal{S}_i \times \mathcal{H}_i)$. It is defined pointwise, as in [FLP84], which means we only have to define $[\![S]\!]_v(\langle \sigma, h\rangle)$, for all pairs. Once this is done, we derive the semantics for sets as follows: $[\![S]\!]_v(V) = \bigcup_{\langle \sigma, h\rangle \in V}\{[\![S]\!]_v(\langle \sigma, h\rangle)\}$. We will use $\langle \sigma, h\rangle$ also to denote elements of $\mathcal{S}_i \times \mathcal{H}_i$.

Note the update of the history in the assignment clause:

$\bullet[\![l : x := e]\!]_v(\langle\sigma, h\rangle) = \{\langle\sigma[\sigma(e)/x], h \cdot l\rangle\}$

Intuitively, the clause states that when executing $l : x := e$ in local point $\langle\sigma, h\rangle$, one obtains a new point, of which the state is updated with respect to variable x, and the (local) history is suffixed by l.

The semantics of the output statement is obtained by augmenting the history with a quadruple label which expresses that a corresponding communication has taken place.

$\bullet[\![l : c!e]\!]_v(\langle\sigma, h\rangle) = \{\langle\sigma, h \cdot \langle e_\sigma, c, l, ?\rangle\rangle\}$

In order to obtain a compositional semantics, we have to define the semantics of individual statements in such a way that their behaviour in all possible contexts is captured. Therefore, in the case of the input statement, which is highly context-dependent, we have to include all pairs of changed states and extended histories describing possible communications:

$\bullet[\![l : c?x]\!]_v(\langle\sigma, h\rangle) = \{\langle\sigma[\alpha/x], h \cdot \langle\alpha, c, ?, l\rangle\rangle \mid \alpha \in \mathbb{Z}\}$

3.3 Kripke-style Semantics of Programs

Now that we have defined the view-semantics of statements, we are able to define the semantics of programs, possibly composed of several processes. Our domain will be a pair consisting of a Kripke structure $\mathcal{M} = (\mathcal{S}\times\mathcal{H}, \pi, R_1, ..., R_n)$ and a pair $\langle\sigma, h\rangle$, where the relation R_i represents the accessibility relation of process i, giving the points that are "equivalent" as far as process i is concerned. The pair $\langle\sigma, h\rangle$ fulfils the role of current world in the Kripke semantics of statements and assertions.

A few words are in order here to explain the emerging Kripke structure, in particular concerning the relations R_i. Each equivalence relation R_i, belonging to process i, is derived from the semantics V_i of the statement that process i contains, in a simple fashion: V_i divides the set $\mathcal{S} \times \mathcal{H}$ into two classes in a trivial way. The resulting relation, which is written R_{V_i} is formally defined as follows (the formal definition of the restriction operator $\restriction$ follows):

$$\langle\sigma, h\rangle R_{V_i} \langle\sigma', h'\rangle \iff [\langle\sigma, h\rangle \restriction i \in V_i \Leftrightarrow \langle\sigma', h'\rangle \restriction i \in V_i]$$

That is, two points are equivalent according to i iff their projections onto process i are either both in V_i or both *not* in V_i. The intuition behind this is the inability of i to distinguish points that are "the same locally", i.e. with respect to i. We now define formally the operations of restriction and chaotic closure:

Definition 1 *The restriction operator* $\restriction: (\mathcal{S} \times \mathcal{H}) \times I\!N \to (\mathcal{S}_i \times \mathcal{H}_i)$ *is defined as follows:*

$$\langle\sigma, h\rangle \restriction i = \langle\sigma \restriction_s i, h \restriction_h i\rangle$$

where $\sigma \restriction_s i$ is defined by $\sigma \restriction_s i(hx) = \sigma(hx) \restriction_h i, hx \in \mathsf{LHVAR}$
$\sigma \restriction_s i(x) = \sigma(x), x \in \mathsf{VAR}(P_i) \bigcup \mathsf{LVAR}$

and $h \restriction_h i$ denotes the i-local projection of some global history $h \in \mathcal{H}$.

Thus, the restriction operator consists of a state restriction and a history restriction. One could view the restriction of a global point with respect to process P_i as a projection, or selection of only i-relevant information of that global point. The definition of history projection is omitted here (see full paper [HM93]), but we would like to remark that although global histories are posets, projections of global histories are sequences (or linear posets), which reflects the sequential nature of processes.

Definition 2 *The chaotic closure operators* $CC' : \mathcal{S}_i \times \mathcal{H}_i \to \mathcal{P}(\mathcal{S} \times \mathcal{H})$ *and* $CC : \mathcal{P}(\mathcal{S}_i \times \mathcal{H}_i) \to \mathcal{P}(\mathcal{S} \times \mathcal{H})$ *are defined as follows:*

$$CC'(\langle \sigma, h \rangle) = \{\langle \sigma', h' \rangle \mid \langle \sigma', h' \rangle \restriction i = \langle \sigma, h \rangle\}$$

$$CC(H) = \bigcup_{\langle \sigma, h \rangle \in H} CC'(\langle \sigma, h \rangle)$$

From this definition, it is clear that chaotic closure is a kind of 'inverse projection': the chaotic closure 'blows up' a local point in every non-i aspect.

Remark From these definitions, it follows that the R_{V_i}-equivalence class of a point $\langle \sigma, h \rangle$ in $\mathcal{S} \times \mathcal{H}$ is equal to $CC(V_i)$ if $\langle \sigma, h \rangle \restriction i \in V_i$, and $(\mathcal{S} \times \mathcal{H}) \backslash CC(V_i)$ if $\langle \sigma, h \rangle \restriction i \notin V_i$.

Definition 3 *(semantics of programs) Define* $\mathcal{K}$ *as the set of all Kripke structures of the form* $(\mathcal{S} \times \mathcal{H}, \pi, R_1, ..., R_n)$ *for* $n \in \mathbb{N}$. *Let* $PR = [P_1 :: S_1 \parallel ... \parallel P_n :: S_n]$. *Now*

$$[\![\cdot]\!] : PR \times \mathcal{K} \times (\mathcal{S} \times \mathcal{H}) \to \mathcal{K} \times (\mathcal{S} \times \mathcal{H})$$

is defined by

•$[\![[P_1 :: S_1 \parallel \cdots \parallel P_n :: S_n]]\!](\mathcal{M}_0, \langle \sigma_0, h_0 \rangle) = (\mathcal{M}, \langle \sigma, h \rangle)$, *where*

$$\begin{array}{l} \mathcal{M} = (\mathcal{S} \times \mathcal{H}, \pi, R_{V_1}, ..., R_{V_n}), \\ V_i = [\![S_i]\!]_v(\langle \sigma_0, h_0 \rangle \restriction i), \\ \langle \sigma, h \rangle = f_c(POSS(P_1, ..., P_n, \langle \sigma_0, h_0 \rangle)), \end{array}$$

where f_c *is a choice function, picking an arbitrary pair* $\langle \sigma, h \rangle$ *from a given set in* $\wp(\mathcal{S} \times \mathcal{H})$, *and* $POSS$ *is defined below.*

The function $POSS$ serves to describe the set of worlds that are held possible by all the processes involved, starting from a common point $\langle \sigma_0, h_0 \rangle$. It is defined as $POSS(P_1, ..., P_n, \langle \sigma, h \rangle) = \bigcap_i CC([\![S_i]\!](\langle \sigma, h \rangle \restriction i))$, with as direct consequence $POSS(P_1, ..., P_n, \langle \sigma, h \rangle) = \bigcap_i POSS(P_i, \langle \sigma, h \rangle)$. (For a more constructive definition, we refer to [HHM93]. However, the intersection used in the definition of $POSS$ above allows us to relate the semantics with the logic in a clearer way.) The choice function is needed in order to obtain an arbitrary point out of this set, which, together with the model $\mathcal{M}$ constitutes a world.

4 Syntax of Formulae

In this section, we define our language of assertions. There will be three kinds of assertions: local assertions $Assn_i$, non-epistemic assertions $Assn_-$ and epistemic assertions $Assn$. These will correspond to the different correctness formulae to be defined further on. Moreover, we define sets of (local) expressions and local history expressions:

$$\begin{array}{lrl} Expr_i & e_i :: & \alpha \mid x_i(\in \mathsf{VAR}_i) \mid e_i + e_i' \mid e_i - e_i' \mid e_i \times e_i' \\ Expr & e :: & \alpha \mid x(\in \mathsf{VAR}) \mid e + e' \mid e - e' \mid e \times e' \\ Hexpr_i & he_i :: & \epsilon \mid l_i \mid \langle e_i, c, l_i, ? \rangle \mid \langle x_i, c, ?, l_i \rangle \mid he_i \cdot he_i' \mid hx \restriction i \mid hist \restriction i \\ Assn_i & \varphi_i :: & e_i = e_i' \mid he_i = he_i' \mid \neg \varphi_i \mid \varphi_i \wedge \varphi_i' \mid \exists x[\varphi_i] \mid \exists hx[\varphi_i] \\ Assn_- & \varphi_- :: & e = e' \mid \varphi_i \mid \varphi_- \wedge \varphi_-' \\ Assn & \varphi :: & \varphi_- \mid K_i \varphi \mid K_G \varphi \end{array}$$

The definition of expressions is as in the definition of the syntax of the language. As to the history expressions, these consist of the empty history ϵ, or a (simple or quadruple) label, to be understood as a poset, or two history expressions composed sequentially, or the projection of a history variable hx, or the projection of the current history

hist. Note that there are no global history expressions, forcing us to reason about history expressions on the local level only.

The local assertions consecutively denote equality of expressions, equality of history expressions, and the operations negation, conjunction and quantification. The non-epistemic assertions include equality of non-local expressions, the set of all local assertions and the operation of conjunction. The epistemic assertions finally consist of the non-epistemic assertions, and two predicates expressing the knowledge of a processor of some local assertion, and the knowledge of a group of processors of some assertion.

In the sequel, we will use the expressions $hx = hx'$, $hx = hist$, etcetera to denote the assertions $\bigwedge_i hx \restriction i = hx' \restriction i$ and $\bigwedge_i hx \restriction i = hist \restriction i$.

5 Semantics of Formulae

In order to assign a meaning to (history) expressions in local and global points, we have the following functions available: $\mathcal{V} : Expr \times (\mathcal{S} \times \mathcal{H}) \to \mathbb{Z}$, $\mathcal{V}_i : Expr_i \times (\mathcal{S}_i \times \mathcal{H}_i) \to \mathbb{Z}$ and $\mathcal{V}_{hi} : Hexpr_i \times (\mathcal{S}_i \times \mathcal{H}_i) \to \mathcal{H}_i$. From the last function $\mathcal{V}_{hi}$, we list two cases:

$$\mathcal{V}_{hi}(\langle e_i, c, l, ?\rangle)(\langle \sigma, h\rangle) = \langle \mathcal{V}_i(e_i)(\langle \sigma, h\rangle), c, l, ?\rangle$$
$$\mathcal{V}_{hi}(hist \restriction i)(\langle \sigma, h\rangle) = h$$

One can easily prove that the functions $\mathcal{V}$ and $\mathcal{V}_i$ yield the same results when given a local expression. Formally: for all $e_i \in Expr_i$: $\mathcal{V}(e_i)(\langle \sigma, h\rangle) = \mathcal{V}_i(e_i)(\langle \sigma, h\rangle \restriction i)$.

Furthermore there are three interpretation functions on assertions: $\mathcal{T}_i : Assn_i \times (\mathcal{S}_i \times \mathcal{H}_i) \to \{\mathbf{true}, \mathbf{false}\}$, $\mathcal{T}_- : Assn_- \times (\mathcal{S} \times \mathcal{H}) \to \{\mathbf{true}, \mathbf{false}\}$ and $\mathcal{T} : Assn \times \mathcal{K} \times (\mathcal{S} \times \mathcal{H}) \to \{\mathbf{true}, \mathbf{false}\}$. Again we give only two interesting clauses (in the second clause, R_G denotes $\bigcap_i R_{V_i}$):

$$\mathcal{T}(K_i\varphi)(\mathcal{M}, \langle \sigma, h\rangle) = \forall\langle \sigma', h'\rangle[\langle \sigma, h\rangle R_{V_i} \langle \sigma', h'\rangle \Rightarrow \mathcal{T}(\varphi)(\mathcal{M}, \langle \sigma', h'\rangle)]$$
$$\mathcal{T}(K_G\varphi)(\mathcal{M}, \langle \sigma, h\rangle) = \forall\langle \sigma', h'\rangle[\langle \sigma, h\rangle R_G \langle \sigma', h\rangle \Rightarrow \mathcal{T}(\varphi)(\mathcal{M}, \langle \sigma, h\rangle)]$$

Note that in this way, we obtain $S5$-properties, and that group knowledge is defined like *implicit* or *distributed* knowledge ([HM85, vdHM92]).

6 Reasoning about Programs

In order to define our correctness formulas further on, we need the definitions of validation as defined below. One should be aware of the limitations (with respect to the assertions that can be evaluated) of the semantical units involved. For instance, in a local point, say from $\mathcal{S}_i \times \mathcal{H}_i$, only local assertions from $Assn_i$ can be evaluated; and in a global point from $\mathcal{S} \times \mathcal{H}$, only non-epistemic assertions (i.e. from $Assn_-$) can be evaluated. An epistemic assertion from $Assn$ can only be interpreted in a *world*.

Definition 4 *For all closed φ_i, φ_-, and φ we define*

$$(\mathcal{S}_i \times \mathcal{H}_i \ni)\langle \sigma, h\rangle \models_i \varphi_i \Leftrightarrow \mathcal{T}_i(\varphi_i)(\langle \sigma, h\rangle)$$
$$(\mathcal{S}_i \times \mathcal{H}_i \supseteq)V \models_i \varphi_i \Leftrightarrow \forall\langle \sigma, h\rangle \in V[\langle \sigma, h\rangle \models \varphi_i]$$
$$(\mathcal{S} \times \mathcal{H} \ni)\langle \sigma, h\rangle \models \varphi_- \Leftrightarrow \mathcal{T}_-(\varphi_-)(\langle \sigma, h\rangle)$$
$$(\mathcal{S} \times \mathcal{H} \supseteq)V \models \varphi_- \Leftrightarrow \forall\langle \sigma, h\rangle \in V[\langle \sigma, h\rangle \models \varphi_-]$$
$$(\mathcal{M}, \langle \sigma, h\rangle) \models \varphi \Leftrightarrow \mathcal{T}(\varphi)(\mathcal{M}, \langle \sigma, h\rangle)$$

We next want to express that evaluating a local i-assertion in a global point amounts to evaluating the assertion in the local point which is the i-restriction of that global point. Stated otherwise: the validity of local assertions does not depend on the environment (= other processes). Formally:

Lemma 1

$$\forall \varphi_i \in Assn_i \forall \langle \sigma, h \rangle \in \mathcal{S} \times \mathcal{H}[\langle \sigma, h \rangle \models \varphi_i \leftrightarrow \langle \sigma, h \rangle \restriction i \models_i \varphi_i]$$

We now come to the definition of two kinds of correctness formulae, connecting program semantics with semantics of assertions. These are instances of the well-known Hoare-triples, with the following informal interpretation: if the precondition holds at a certain point, then, after execution of the statement/program involved, the postcondition holds.

Definition 5 • $\models_i \{\varphi_i\} S_i \{\psi_i\} \Leftrightarrow \forall \langle \sigma, h \rangle \in \mathcal{S}_i \times \mathcal{H}_i[\langle \sigma, h \rangle \models_i \varphi_i \Rightarrow [\![S_i]\!]_v(\langle \sigma, h \rangle)) \models_i \psi_i]$

• $\models \{\varphi_-\} PR \{\psi\} \Leftrightarrow \forall \mathcal{M} \in \mathcal{K} \forall \langle \sigma, h \rangle \in \mathcal{S} \times \mathcal{H}[\langle \sigma, h \rangle \models \varphi_- \Rightarrow [\![PR]\!](\mathcal{M}, \langle \sigma, h \rangle)) \models \psi]$

Note that in the second rule, as a special case of PR we can have $[P_i :: S_i]$; moreover, note that the postassertion of the second rule is the only assertion that is interpreted in a Kripke-model.

7 Proof System

The proof system is divided into three parts: a general part, a local part and a global part. The general part contains rules that hold in both the local and the global system. The local part contains rules and axioms to describe the individual program constructs of processes. The global part deals with parallel constructs, or programs, and also covers knowledge-related issues. In the general part, the symbol S denotes either a process S or a program PR.

7.1 General Part

Axiom 1 *(tautologies)* All valid assertions in first-order arithmetic

Axiom 2 *(K-axiom)* $(K_i\varphi \wedge K_i(\varphi \rightarrow \psi)) \rightarrow K_i\psi$

Axiom 3 *(veridicality)* $K_i\varphi \rightarrow \varphi$

Axiom 4 *(positive introspection)* $K_i\varphi \rightarrow K_iK_i\varphi$

Axiom 5 *(negative introspection)* $\neg K_i\varphi \rightarrow K_i\neg K_i\varphi$

Axiom 2'–5' replace i by G in axioms 2–5

Axiom 6 *(group knowledge)* $K_i\varphi \rightarrow K_G\varphi$, *where* $i \in G$

Rule 1 *(modus ponens)* $\dfrac{\varphi, \varphi \rightarrow \psi}{\psi}$

Rule 2 *(necessitation)* $\dfrac{\varphi}{K_i\varphi}$

Rule 3 *(generalization)* $\dfrac{\varphi_i}{\forall x[\varphi_i]}$

Rule 4 *(consequence)* $\dfrac{p \to p', \{p'\}\mathsf{S}\{q'\}, q' \to q}{\{p\}\mathsf{S}\{q\}}$

Rule 5 *(conjunction)* $\dfrac{\{p_1\}\mathsf{S}\{q_1\}, \{p_2\}\mathsf{S}\{q_2\}}{\{p_1 \wedge p_2\}\mathsf{S}\{q_1 \wedge q_2\}}$

Axioms 1–5 together with rules 1–2 form the well-known logic $S5$. Axioms 2'–5' and 6 add group knowledge to the system. Rule 3 allows first-order local formulae in the system.

The consequence rule allows to strengthen the precondition and weaken the postcondition of a Hoare-triple, thus weakening the triple as a whole; the conjunction rule allows to combine two Hoare triples.

One may wonder why we do not need the Barcan axiom $\forall x[K_i\varphi] \to K_i(\forall x[\varphi])$. The reason for this is that our language does not contain arbitrary first-order $S5$ expressions; for instance, the expression $\forall x[K_i\varphi]$ is not in the language (cf. section 4).

7.2 Local Part

Axiom 7 *(skip)* $\{\varphi_i[hist \cdot l/hist]\}l : skip\{\varphi_i\}$

Axiom 8 *(assignment)* $\{\varphi_i[hist \cdot l/hist, e/x]\}l : x := e\{\varphi_i\}$

Axiom 9 *(output)* $\{\varphi_i[hist \cdot (e, c, l, ?)/hist]\}l : c!e\{\varphi_i\}$

Axiom 10 *(input)* $\{\forall x'[\varphi_i[hist \cdot (x', c, ?, l)/hist, x'/x]]\}l : c?e\{\varphi_i\}$

Rule 6 *(sequential composition)* $\dfrac{\{\varphi_i\}S_1\{\varphi_i''\}, \{\varphi_i''\}S_2\{\varphi_i'\}}{\{\varphi_i\}S_1; S_2\{\varphi_i'\}}$

Rule 7 *(nondeterministic choice)* $\dfrac{\{\varphi_i\}S_1\{\varphi_i'\}, \{\varphi_i\}S_2\{\varphi_i''\}}{\{\varphi_i\}S_1[]S_2\{\varphi_i' \vee \varphi_i''\}}$

Consider for instance the assignment axiom. In order to be sure that φ_i holds after execution of $l : x := e$, we have to guarantee that, in the state before the assignment, φ_i holds where e is substituted for x (the same holds for the history assignment: all non-composed statements have this assignment in their axiom, because semantically, they all do a history assignment).

As for the input axiom: because it is not known what value has been received, the only way to guarantee that φ_i holds after execution is to have that substituting *any* value for x results in a true assertion in the state before the execution.

7.3 Global Part

Rule 8 *(K-introduction)* $\dfrac{\{\varphi_i\}S_i\{\varphi_i'\}}{\{\varphi_i\}P_i :: S_i\{K_i\varphi_i'\}}$

Rule 9 *(K-persistence)* $\dfrac{\{\varphi_-\}P_i :: S_i\{K_i\varphi_i\}(for\ i = 1, ..., n)}{\{\varphi_-\}[P_1 :: S_1 \parallel ... \parallel P_n :: S_n]\{K_i\varphi_i\}}$

Rule 10 *(variable substitution)* As usual; omitted

The K-introduction rule allows us to lift a 'classical' local assertion (valid in the set-theoretic context) to a new assertion, valid in a Kripke-model which states that, within the Kripke framework, the process under consideration *knows* the local assertion.

The K-persistence rule states that knowledge of processes on their own does not change when placed within a bigger context.

The variable substitution rule is needed for technical reasons, in the completeness proof (to get rid of 'freeze' variables).

7.4 Examples

1. Let $PR = [P_i :: l : c?x \parallel P_j :: l : c!5]$.
   $\vdash \{hist \restriction i = \epsilon\} l : c?x \{hist \restriction i = \langle x, c, ?, l\rangle\}$ (axiom 10)
   $\vdash \{hist \restriction i = \epsilon\} P_i :: l : c?x \{K_i(hist \restriction i = \langle x, c, ?, l\rangle)\}$ (K-introduction)
   $\vdash \{hist \restriction i = \epsilon\} PR \{K_i(hist \restriction i = \langle x, c, ?, l\rangle)\}$ (K-persistence)
   $\vdash \{hist \restriction j = \epsilon\} m : c!5 \{hist \restriction j = \langle 5, c, m, ?\rangle\}$ (axiom 9)
   $\vdash \{hist \restriction j = \epsilon\} P_j :: m : c!5 \{K_j(hist \restriction j = \langle 5, c, m, ?\rangle)\}$ (K-introduction)
   $\vdash \{hist \restriction j = \epsilon\} PR \{K_j(hist \restriction j = \langle 5, c, m, ?\rangle)\}$ (K-persistence)
   $\vdash \{hist \restriction i = \epsilon \wedge hist \restriction j = \epsilon\} PR \{K_i(hist \restriction i = \langle x, c, ?, l\rangle) \wedge K_j(hist \restriction j = \langle 5, c, m, ?\rangle)\}$ (Conjunction)
   $\vdash \{hist = \epsilon\} PR \{K_G(hist \restriction i = \langle x, c, ?, l\rangle) \wedge K_G(hist \restriction j = \langle 5, c, m, ?\rangle)\}$ (Consequence, Group kn.)
   $\vdash \{hist = \epsilon\} PR \{K_G(hist \restriction i = \langle x, c, ?, l\rangle \wedge hist \restriction j = \langle 5, c, m, ?\rangle)\}$ (Consequence)
   $\vdash \{hist = \epsilon\} PR \{K_G(hist = \langle 5, c, m, l\rangle \wedge x = 5)\}$ (Consequence)
2. Let $S_1 \equiv l_{11} : c!0; l_{12} : c'!1$, $S_2 \equiv l_{21} : c'?x; l_{22} : c''!x + 1$, $S_3 \equiv (l_{31} : c?z; l_{32} : c''?y) [] (l_{33} : c''?y; l_{34} : c?z)$, and $PR \equiv [P_1 :: S_1 \parallel P_2 :: S_2 \parallel P_3 :: S_3]$. Then we can derive in a similar way $\{true\} PR \{K_{\{1,2,3\}} x = 1 \wedge y = 2 \wedge z = 0\}$, and also $\{true\} PR \{K_{\{1,3\}} z = 0\}$ and $\{true\} PR \{\neg K_{\{2,3\}} y = 2\}$. Note that the last formula says that the combined knowledge of processes 2 and 3 is not enough to derive the value of y. However, we can prove the formula $\{true\} PR \{ K_{\{2,3\}} y = x + 1\}$.

8 Soundness

The axioms 1–6 and rules 1–3 are known to be sound with respect to first order S5 (see e.g. [HC84]). So it follows from our semantics that these axioms and rules are sound. The program axioms together with the rules for sequential composition and nondeterministic choice from the local part can checked to be sound in a standard way, as is the case for the rules of consequence and conjunction. There remains the proof of the K-introduction rule and the K-persistence rule. We only prove the soundness of the K-introduction rule here.

Proposition 1

$$\models_i \{\varphi_i\} S_i \{\psi_i\} \Rightarrow \models \{\varphi_i\} P_i :: S_i \{K_i \psi_i\}$$

proof
Suppose $\models_i \{\varphi_i\}S_i\{\psi_i\} \Leftrightarrow \forall\langle\sigma,h\rangle \in \mathcal{S}_i \times \mathcal{H}_i[\langle\sigma,h\rangle \models_i \varphi_i \Rightarrow [\![S_i]\!]_v(\langle\sigma,h\rangle) \models_i \psi_i]$
To prove: $\models \{\varphi_i\}P_i :: S_i\{K_i\psi_i\} \Leftrightarrow \forall\langle\sigma,h\rangle \in \mathcal{S} \times \mathcal{H}, \mathcal{M} \in \mathcal{K}[\langle\sigma,h\rangle \models_i \varphi_i \Rightarrow [\![P_i :: S_i]\!](\mathcal{M},\langle\sigma,h\rangle)$
$\models_i K_i\psi_i]$. So suppose $\langle\sigma,h\rangle \models \varphi_i$. Then also $\langle\sigma,h\rangle \restriction i \models_i \varphi_i$, by Lemma 1
So, by assumption $[\![S_i]\!]_v(\langle\sigma,h\rangle \restriction i) \models_i \psi_i \Leftrightarrow V_i \models_i \psi_i$, where $V_i = [\![S_i]\!]_v(\langle\sigma,h\rangle \restriction i)$
$\Leftrightarrow CC(V_i) \models \psi_i$ (Lemma 1) $\Leftrightarrow ((\mathcal{S} \times \mathcal{H}, \pi, R_{V_i}), f_c(CC(V_i)) \models K_i\psi_i \Leftrightarrow [\![P_i :: S_i]\!](\mathcal{M},\langle\sigma,h\rangle) \models K_i\psi_i$ □

9 Completeness

In order to prove completeness of our system, we extend $Assn_i$ by adding $sp(\varphi_i, S_i)$ to it, the strongest postcondition with respect to a statement and a local assertion. This is an assertion with the following properties:

1. $\{\varphi_i\}S_i\{sp(\varphi_i, S_i)\}$
2. $\{\varphi_i\}S_i\{\psi\} \Rightarrow sp(\varphi_i, S_i) \rightarrow \psi$

1. states that $sp(\varphi_i, S_i)$ *is* a postcondition of φ_i and S_i, 2. states that it is the strongest one.

The semantics is given as follows, as an extension of definition 4:

$$(\mathcal{S}_i \times \mathcal{H}_i \ni)\langle\sigma,h\rangle \models_i sp(\varphi_i, S_i) \Leftrightarrow \exists\langle\sigma_0,h_0\rangle[\langle\sigma_0,h_0\rangle \models_i \varphi_i \wedge \langle\sigma,h\rangle \in [\![S_i]\!]_v(\langle\sigma_0,h_0\rangle)]$$

The proof that this definition indeed provides a semantical characterization of the strongest postcondition is standard and not given here (see e.g.[dB80]). As usual, we have that the strongest postcondition can be expressed in $Assn_i$, for any $\varphi_i \in Assn_i$ and statement S_i (we only list two cases).

- $\models_i sp(\varphi_i, l : c?x) \leftrightarrow (\exists x', hx : \varphi_i[x'/x, hx/hist] \wedge hist \restriction i = hx \restriction i \cdot (x, c, ?, l))$
- $\models_i sp(\varphi_i, S_1; S_2) \leftrightarrow sp(sp(\varphi_i, S_1), S_2)$

The following lemma justifies the previously introduced syntactical abbreviation $hx = hist$ meaning $\bigwedge_i hx \restriction i = hist \restriction i$. Although in our assertion language there is no means of reasoning directly about global history expressions such as *hist*, it follows from this lemma that we *can* make global statements using the abbreviations.

Lemma 2 *A history $h \in \mathcal{H}$ is completely determined by all its projections $h \restriction_h i$, where i ranges over all processes.*

Finally, we are ready for the main theorem of this section. Let $PR = [P_1 :: S_1 \parallel ... \parallel P_n :: S_n]$.

Theorem 1 *The proof system presented in this paper is complete resp. K-complete:*

1. *if* $\models \{\varphi_i\}S_i\{\psi_i\}$ *then* $\vdash \{\varphi_i\}S_i\{\psi_i\}$
2. *if* $\models \{\varphi\}PR\{K_G\psi\}$ *then* $\vdash \{\varphi\}PR\{K_G\psi\}$ $(G \subseteq \{1,...,n\}, \psi \in Assn_-)$

We only prove the second clause of this theorem, that asserts that any correctness formula involving a postcondition referring to the knowledge of (groups of) processes can be derived in our axiom system, provided it is valid.

proof
Let $G = \{1, ..., n\}$, and suppose $\models \{\varphi\}PR\{K_G\psi\}$. Let, for all i, $\overline{x_i}$ denote the list of i-local variables in VAR. Define $\overline{\varphi} = \varphi[\overline{v_1}/\overline{x_1}, ...\overline{v_n}/\overline{x_n}, hx/hist] \wedge \overline{x_1} = \overline{v_1} \wedge ... \wedge \overline{x_n} = \overline{v_n} \wedge hist = hx$ ($hx, \overline{v_i}$ fresh for all i). Clearly, $\overline{\varphi} \rightarrow \varphi$, so $\models \{\overline{\varphi}\}PR\{K_G\psi\}$ holds.
Now let φ_i be the i-local restriction of $(\overline{\varphi})$, formally defined in the full paper.
Then we can prove $\overline{\varphi} \leftrightarrow \bigwedge_i \varphi_i$. Now by definition of $sp(\varphi_i, S_i)$, we have $\models \{\varphi_i\}S_i\{sp(\varphi_i, S_i)\}$.
Thus, by 1. it follows $\vdash \{\varphi_i\}S_i\{sp(\varphi_i, S_i)\}$.
By rule 8 it follows that $\vdash \{\varphi_i\}P_i :: S_i\{K_i(sp(\varphi_i, S_i))\}$, all i
By K-persistence, $\vdash \{\varphi_i\}PR\{K_i(sp(\varphi_i, S_i))\}$, all i
Then, by conjunction, $\vdash \{\bigwedge_i \varphi_i\}PR\{\bigwedge_i K_i(sp(\varphi_i, S_i))\}$
Group knowledge: $\vdash \{\bigwedge_i \varphi_i\}PR\{\bigwedge_i K_G(sp(\varphi_i, S_i))\}$
Distrib. of K_G over $\wedge$: $\vdash \{\bigwedge_i \varphi_i\}PR\{K_G \bigwedge_i(sp(\varphi_i, S_i))\}$
K-axiom, and $\bigwedge_i(sp(\varphi_i, S_i)) \rightarrow \psi$ (see full paper): $\vdash \{\bigwedge_i \varphi_i\}PR\{K_G\psi\}$
Consequence: $\vdash \{\overline{\varphi}\}PR\{K_G\psi\}$
Variable substitution rule: $\vdash \{\varphi\}PR\{K_G\psi\}$

10 Conclusion

In this paper we have presented a proof system for the correctness of a simple parallel programming language using a logic in which epistemic operators are included to be able to speak about the knowledge of the sub-processes involved in the execution of parallel programs in this language. As we have seen this proof system comprises of a local and a global part. The former is classical dealing with the correctness of local processes, whereas the latter part concerns the parallel composition of processes and eventually the whole process of parallel computation.

It is in this latter part where the use of epistemic operators comes into the picture. These operators enable us to (still) refer to assertions along with the agents (processes) that know them. Combining this knowledge to knowledge of larger groups of processes eventually gives us the desired assertions known by the process as a whole but again we still can refer to the knowledge of every subgroup of processes when we want to. So combining knowledge into group knowledge does not destroy the information about what is known by subgroups. This illustrates the modularity of our approach. So, summarizing, one could state that in our proof system, the constructs on the local level are handeled in a more or less standard way, whereas the parallel (top) construct is treated by means of epistemic operators.

We believe that the epistemic approach to the correctness of parallel programs may be used fruitfully for a range of programming languages. In particular, since our approach is agent-oriented we believe that the approach is amenable to object-oriented parallel programming languages since the objects in these languages are exactly the agents/processes involved in the execution of a program. This will be investigated in future research. We would finally like to mention that we do not need a merging lemma ([Apt83]), due to compositionality of the semantics (cf. also [AdB93]).

acknowledgement Thanks to Jozef Hooman for providing us with his very useful lecture notes ([Hoo93]) and for suggesting us to use channeled communication.

References

[AdB93] P.H.M. America and F.S. de Boer. Reasoning about dynamically evolving process structures. *Formal Aspects of Computing*, 3, 1993.

[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.

[Apt83] K.R. Apt. Formal justification of a proof system for communicating sequential processes. *Journal of the ACM*, 30:197–216, 1983.

[BRR89] J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors. *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency.* Springer-Verlag, 1989. LNCS 354.

[dB80] J. W. de Bakker. *Mathematical Theory of Program Correctness.* Prentice-Hall, 1980.

[FLP84] N. Francez, D. Lehmann, and A. Pnueli. A linear-history semantics for languages for distributed programming. *Theoretical Computer Science*, 32:25–46, 1984.

[HC84] G.E. Hughes and M.J. Cresswell. *A Companion to Modal Logic.* Methuen, 1984.

[HHM93] W. van der Hoek, M. van Hulst, and J.-J.Ch. Meyer. Towards an epistemic approach to reasoning about concurrent programs. In G. Rozenberg J.W. de Bakker, W.-P. de Roever, editor, *Semantics: Foundations and Applications*, number 666 in LNCS, pages 261–287. Springer-Verlag, 1993.

[HM85] J.Y. Halpern and Y.O. Moses. A guide to the modal logics of knowledge and belief. In *Proc. 9th IJCAI*, pages 480–490, 1985.

[HM93] M. van Hulst and J.-J.Ch. Meyer. An epistemic proof system for parallel processes. Technical Report 93-17, Nijmegen University, The Netherlands, 1993.

[Hoa78] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[Hoo93] J. Hooman. Verification of parallel systems, 1993. Course Notes.

[HZ87] J.Y. Halpern and L.D. Zuck. A little knowledge goes a long way: Simple knowledge-based derivations and correctness proofs for a family of protocols. In *Proc. of 6th PODC*, pages 269–280, 1987.

[KT86] S. Katz and G. Taubenfeld. What processes know: definitions and proof methods. *ACM-PODC*, pages 249–262, 1986.

[MvdH] J.-J.Ch. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science.* Cambridge Univ. Press. forthcoming.

[OG76] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6:319–340, 1976.

[vdHM92] W. van der Hoek and J.-J. Ch. Meyer. Making some issues of implicit knowledge explicit. *Foundations of Computer Science*, 1992.

[Zwi88] J. Zwiers. *Compositionality, Concurrency and Partial Correctness.* PhD thesis, Technical University Eindhoven, 1988.