Autonomous Agents and Multi-Agent Systems, 6, 185–219, 2003
© 2003 Kluwer Academic Publishers. Manufactured in The Netherlands.

A Verification Framework for Agent Communication

ROGIER M. VAN EIJK, FRANK S. DE BOER,
WIEBE VAN DER HOEK* and JOHN-JULES CH. MEYER {rogier, frankb, wiebe, jj}@cs.uu.nl
Institute of Information and Computing Sciences, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

Abstract. In this paper, we introduce a verification method for the correctness of multiagent systems as described in the framework of ACPL (Agent Communication Programming Language). The computational model of ACPL consists of an integration of the two different paradigms of CCP (Concurrent Constraint Programming) and CSP (Communicating Sequential Processes). The constraint programming techniques are used to represent and process information, whereas the communication mechanism of ACPL is described in terms of the synchronous handshaking mechanism of CSP. Consequently, we show how to define a verification method for ACPL in terms of an integration of the verification methods for CCP and CSP. We prove formally the soundness of the method and discuss its completeness.

Keywords: agent communication, semantics, verification methods, specification, agent communication language, multi-agent systems

1. Introduction

In the research on agent communication languages, the following problem has been frequently pointed out: given a particular multiagent system, how do we check whether it acts in accordance with a particular specification of behaviour [29, 48]? In the agent literature this issue is also known as the problem of *conformance testing* [48]. Among others, the problem has been encountered in the design of the KQML standard for agent communication [17], where, for example, there is the issue how to verify that a particular agent performing the communication action `tell`(φ) to tell another agent that it believes the information φ to hold, indeed has φ as one of its beliefs? The problem of conformance testing has the following three parameters (see Figure 1):

1. a multiagent system,
2. a specification of behaviour,
3. a method to verify that the multiagent system satisfies the specification.

On the one hand, in the literature, various logics to reason about agent communication have been proposed, like (Cohen and Levesque, [11]), (Linder et al., [30]), (Bretier and Sadek, [10]) and (Labrou and Finin, [28]). The last two form the

* Additional affiliation: Department of Computer Science, University of Liverpool, UK

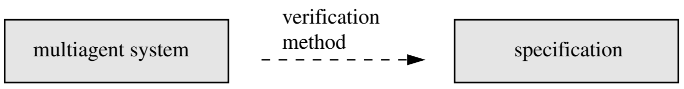

Figure 1. Parameters of conformance testing.

basis of the FIPA and the KQML agent communication languages, respectively. However, in general, these logics are not supported by a corresponding *computational model* for multiagent systems. On the other hand, various agent-oriented programming languages have been proposed, such as AGENT-0 [43], PLACA [46], 3APL [23], DESIRE [9] and AGENTSPEAK [40]. However, in general, these programming languages are not fully supported by corresponding logics for verification.

The main contribution of this paper consists of the introduction of a verification method for the correctness of multiagent systems as described in the framework of ACPL (Agent Communication Programming Language) [6, 13–16]. The computational model of ACPL consists of an integration of the two different paradigms of CCP (Concurrent Constraint Programming) [41] and CSP (Communicating Sequential Processes) [25]. The constraint programming techniques are used to represent and process information, whereas the communication mechanism of ACPL is described in terms of the synchronous handshaking mechanism of CSP. Consequently, we show how to define a verification method for ACPL in terms of an integration of the verification methods for CCP and CSP.

The paper is organised as follows. First, in Section 2, we give a short historical overview of the various verification calculi that have been developed in the context of concurrent and distributed programming. Then in the following sections, we describe the building blocks of our verification framework for agent communication (see Fig. 2). In Section 3, we consider the multiagent programming language ACPL that is designed to program systems of agents that communicate by exchanging information among each other. In Section 4, we consider the operational semantics of the programming language, which describes the behaviour of multiagent systems in terms of their computations. As we want to abstract from irrelevant details of these computations, we additionally identify a notion of observable behaviour that exactly captures those aspects of computations that are visible to an external observer. We prove that this notion of observable behaviour is compositional. The topic of

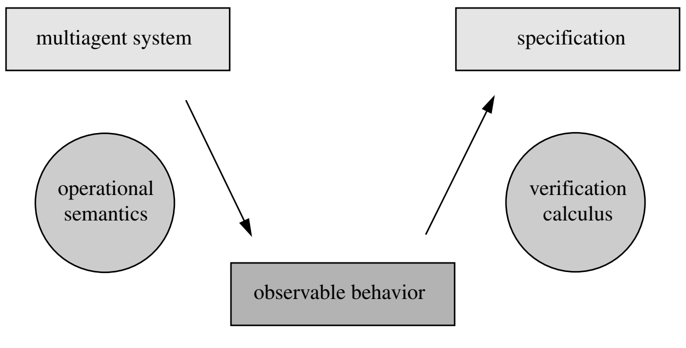

Figure 2. A verification framework for agent communication.

Section 5 is the introduction of an appropriate assertion language to express specifications of agent behaviour. In Section 6, we define a compositional proof system, or verification calculus, that can be used to verify that a multiagent system satisfies a particular specification. Finally, Section 7 discusses related approaches as well as issues for future research.

2. Program verification

The importance of being able to establish that computer programs exactly behave in the way they are supposed to, has been recognised since the early days of computer science [47]. A brute-force approach to establish the accuracy or correctness of a program is through a *testing* procedure in which for all possible inputs and under all possible circumstances the program is checked to exhibit the desired behaviour. In practice, however, due to the large—if not infinite—number of different situations to be considered, this process of accomplishing correctness is normally approximated through a reduction of the search space to a set of representative situations. However, although the latter form of testing may substantially increase the confidence in the correctness of the program, it does not yield a guarantee that the program is entirely free of errors. That is, the program may still contain errors that have not presented themselves during the testing phase, but can result in incorrect behaviour during the actual execution of the program.

A second approach is to establish the correctness of a computer program by means of a *mathematical proof*. The desired behaviour of a program is then given in terms of a *specification* that is expressed in some formal language. An example of such a specification is for instance the following:

$$P \ sat \ \Phi,$$

which can be thought of expressing that the program P satisfies the behaviour as described by the *assertion* Φ. In particular, it expresses that the assertion Φ is satisfied by all possible executions of the program P.

The procedure of checking that a particular program satisfies its specification is referred to as the process of *verification*. A straightforward approach is to perform the verification process at the operational level of the programming language. However, at this level, correctness proofs appear to be rather laborious and typically contain many repetitions of the same line of reasoning [31]. Therefore, a more structured approach for the verification of programs is through the development of a *proof system*. Such a system constitutes a calculus for the formal derivation of correct specifications, which for instance comprises rules of the following format:

$$\frac{P \ sat \ \Phi}{Q \ sat \ \Psi}$$

This rule states that if it can be derived that the program P satisfies the assertion Φ then it can also be derived that the program Q satisfies the assertion Ψ. In the construction of such a proof system, two important properties should be taken into account, namely the soundness and the completeness of the system. A proof system that is *sound* only derives specifications of the form P *sat* Φ that are valid, while a proof system that is *complete* derives all possible valid specifications.

Additionally, an important issue concerns the *compositionality* of the verification process, which amounts to the property that the verification of a specification can be based upon specifications of the behaviour of the *components* of the program, without making reference to the inner structure of these components [49]. An example of a rule in a compositional proof system is the following:

$$\frac{P_1 \ sat \ \Phi_1 \cdots P_n \ sat \ \Phi_n}{P \ sat \ \Phi}$$

where P denotes a program that is composed of the components $P_1, \ldots, P_n$ and Φ constitutes an assertion that is obtained from the assertions $\Phi_1, \ldots, \Phi_n$. The rule states that the specification of the behaviour of the program P can be derived from the specification of its components, without using any information about the components P_i except for their behaviour that is described by the assertion Φ_i.

2.1. *Verification of communicating processes*

The purpose of this paper is the development of a calculus for proving the correctness of programs that are expressed in the basic multiagent programming language ACPL. Fortunately, we do not need to start from scratch, but we can build upon techniques that have been developed in the field of concurrent programming, and in particular, upon the proof systems for the Communicating Sequential Processes paradigm (CSP) [25] and the paradigm of Concurrent Constraint Programming (CCP) [41].

The first proof systems to reason about the correctness of programs have been developed in the context of *sequential* programming languages, of which the most influential systems are the *inductive assertion method* [18, 35] and *Hoare logic* [25]. About a decade later, proof systems for the correctness of *concurrent* programs began to emerge, where the initial focus was on programming languages in which concurrent processes interact with each other by means of a set of shared variables. In one of the most influential methods [36], an important feature is—what is referred to as—the *interference freedom test*, which amounts to the condition that the specification of the behaviour of a parallel component does not interfere with the specification of the behaviour of any other component of a parallel program. This condition ensures that correct specifications of the different parallel subprograms can be combined to form a correct specification of the overall program.

The techniques of the proof systems for shared-variable languages have subsequently been applied and refined to programming languages for distributed programming, in which processes do not have a set of shared variables but instead interact with each other via the exchange of data along a network of communication channels. In the proof system of [2], an important notion is—what is called—the *cooperation test*, which amounts to the condition that the two parallel programs that are involved in a bilateral communication step cooperate with each other; that is, the first program is required to send a particular data item at the same point at which the second process is ready to receive it. However, this proof system does not satisfy the property of compositionality, as the cooperation test makes reference to the internal structure of the parallel components.

A compositional proof system for the Communicating Sequential Programming paradigm has been described in [53], which is inspired on earlier work published in [34]. The main feature of this compositional calculus is the introduction of a *communication invariant* that describes the communication history of a program and its components. In particular, this invariant defines which communication steps have taken place and moreover, denotes in which order they have occurred. Accordingly, the communication invariant models the *interface* of the program with its environment, as it is expressed in terms of the visible communication actions without making any reference to the internal structure of the program.

2.2. *Verification of concurrent constraint programming*

In developing a proof system for agent communication, we combine the ideas of the above compositional proof system for CSP with techniques from the proof system for the Concurrent Constraint Programming paradigm, which has been reported in (Boer et al. [7]). The main feature of the latter calculus is that it makes use of the logic of the underlying constraint system to reason about the correctness of programs. That is, constraints themselves are viewed as descriptions of the behaviour of a program. To illustrate this point, let us consider the following specification:

$$\texttt{update}(\varphi)\ \mathit{sat}\ \Phi,$$

which expresses that after termination of the program $\texttt{update}(\varphi)$ a particular assertion Φ holds. This assertion Φ should be such that it expresses a suitable property of the result of the execution of the program. An obvious candidate for this assertion is the property that the information store, as computed by the program, contains the information φ. If we write this assertion simply as φ, we obtain the specification $\texttt{update}(\varphi)$ *sat* φ, which thus expresses that after the execution of the program $\texttt{update}(\varphi)$, the resulting store contains the information φ.

In order to take account of the other programming constructs of CCP, the language of assertions is enriched with some additional logical operations. For instance, the parallel composition '&' can be modelled by the conjunction of assertions:

$$\texttt{update}(\varphi)\ \&\ \texttt{update}(\psi)\ \mathit{sat}\ \varphi \wedge \psi.$$

This specification expresses that after the parallel execution of the actions $\texttt{update}(\varphi)$ and $\texttt{update}(\psi)$, the resulting information stores contains the information φ and ψ.

An additional feature of this proof system for CCP is that the verification of the parallel composition of programs can be straightforwardly derived from the verification of its parallel components, without the necessity of testing for interference freedom. This is in contrast to the non-logical shared-variable programming languages in which this test is necessary (Owicki and Gries, 1976).

2.3. *Verification of agent communication*

In this paper, we present a proof system to reason about the correctness of programs that are expressed in the basic multiagent programming language ACPL. It is used to derive specifications that are of the following form:

$$A \ sat \ \Phi,$$

where the assertion Φ not only describes the information stores as computed by the agents in the system A but additionally specifies the communication behaviour that is exhibited during the execution. Similar to the proof system for CCP, we consider the *partial* correctness of programs, which says that Φ holds for all *terminating* computations of the multiagent system A.

3. Multiagent systems

In this section, we define the basic multiagent programming language ACPL.

3.1. *Information*

When agents in a multiagent system communicate with each other, they do this with respect to some particular *domain of discourse* [19]. This domain can be anything ranging from the objects in an unknown external world that the agents are exploring, to, for instance, the quotations at the stock-exchange. In our abstract view on this domain of discourse, we assume that there is some particular set of *tokens* that are used to describe the basic pieces of information about it.

In order to be able to refer to the elements of the domain we adopt the traditional notion of a *variable*. Thus, a token contains variables and in saying that a particular token is true of the domain, an agent *constrains* the interpretation of the variables that occur in it. Therefore, these pieces of information are also referred to as *constraints* [33].

For example, consider the situation of two agents negotiating for the price x of a particular good [32], where the purpose of the negotiation is to determine a value for x. The selling agent starts the negotiation with a proposal, which does not directly constitute the final value for x, but rather denotes an upper bound on the price of the good. That is, the buyer is not expected to pay a price that exceeds the proposal, but can either accept the offer or can make a counteroffer. The purpose of such a counteroffer is to make up a lower bound on the price; i.e., the seller is not expected to sell the good for a price less than the counteroffer; i.e., it can either accept the counteroffer or propose a new offer. A typical example of this is a scenario in which the buyer and the seller alternately put forward the following constraints:

$$x \leq 40, x \geq 20, x \leq 30, x \geq 25, x \leq 27, x \geq 27.$$

Thus, the seller proposes a price of 40, the buyer a price of 20, the seller a price of 30 and so on. The negotiation stops if a value for x, namely 27, has been determined and thus the price of the good has been finalised.

Having the notion of a token, let us consider the basic operations that are needed for the manipulation of tokens. First of all, it should be possible to express that *multiple* tokens are true of particular elements in the domain of discourse, that is, we need a conjunction operation $\wedge$ to *accumulate* pieces of information. Secondly, it should be possible to compare pieces of information with respect to their content; that is, we need an operator $\vdash$ to test whether one particular piece of information *entails* another piece of information. Additionally, we need the ability to hide information. To support such a hiding facility, we introduce operators of the form $\exists_x$ to express that the information on the variable x is hidden and not visible from outside. For instance, the information $\exists_x((x = y + 5) \wedge x = 7)$ is equivalent to the information $y = 2$. Finally, we need a substitution operation: $\varphi[y/x]$ denotes the information that is obtained from the information φ by replacing all the free occurrences of the variable x by occurrences of the variable y.

A set of tokens together with a conjunction operator $\wedge$, an entailment relation $\vdash$ and a set of hiding operators of the form $\exists_x$, forms a *constraint system* [42]. The application of constraint systems ranges from arithmetic [41] to the Blocks World [22] to electronic negotiations as indicated above.

Agent communication languages, like the language KQML [17], can be thought of being divided in different layers (see Figure 3). Constraint systems can be used to represent the *content layer* of KQML, which involves information on the domain of discourse. Additionally, we will represent the speech act types of the *message layer* of KQML by corresponding operators on the underlying constraint system. For example, a KQML expression consisting of a content expression φ that is encapsulated in a message wrapper containing the speech act untell is represented by the expression `untell`(φ). We assume an extension of the entailment relation of the constraint

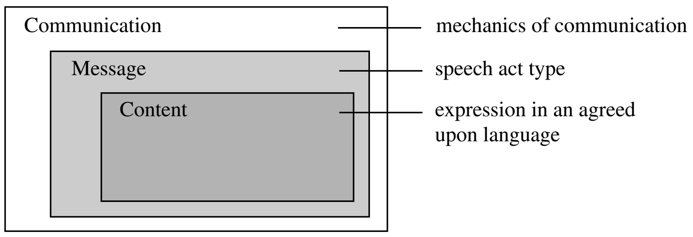
Communication
Message
Content
mechanics of communication
speech act type
expression in an agreed upon language

Figure 3. Layers of a KQML expression.

system that includes these operators. For instance, given the constraints ψ and φ, we stipulate:

$$\psi \nvdash \varphi \;\Leftrightarrow\; \psi \vdash \texttt{untell}(\varphi).$$

Assuming that ψ represents the belief base of an agent, this rule allows to state that a specific constraint is not entailed.The anti-monotonicity property of the `untell` operator is expressed by:

$$\texttt{untell}(\varphi) \vdash \texttt{untell}(\psi) \;\Leftrightarrow\; \psi \vdash \varphi.$$

From now on, we will identify the speech act extension of a given constraint system with the system itself. This means that a constraint φ can be either a constraint of the underlying constraint system or an application of a speech act to a constraint. We refer to constraints of the constraint system itself as *basic constraints*.

In the next section, we will describe the basic mechanics of the KQML *communication layer* (see Figure 3), which involves aspects like senders, recipients, communication channels and synchronicity.

3.2. Syntax of the programming language

We now consider the actions that model the basic information-processing capabilities of agents. They are of two different categories: actions `query`(φ) and `update`(φ) for the interaction with the agent's belief state, and communication actions $c?\varphi$ and $c!\varphi$ for the interaction with other agents.

First of all, the execution of a basic action `query`(φ) consists in checking whether the belief state of the agent entails the information φ. If φ is entailed by the belief state then execution can proceed; otherwise execution is suspended. Secondly, the execution of `update`(φ) consists in adding the information φ to the current beliefs. In this paper, we do not address the general issue of belief revision [20], where, in order to keep a belief state consistent, beliefs should be removed as well [14].

The execution of the output action $c!\varphi$ consists in sending the constraint φ along the channel c, while the execution of an input action $c?\psi$ consists in anticipating the receipt of the constraint ψ along c. Output and input actions need to be synchronised with each other; that is, an action $c!\varphi$ needs to be synchronised with a matching action $c?\psi$, where matching means $\varphi \vdash \psi$; that is, the information φ that is sent entails the information ψ that is anticipated.

The general use of the entailment relation in the semantics of communication actions allows us to abstract from among others the following :

—the particular *syntax* of information, for instance, `untell`($p \wedge q$) entails `untell`($q \wedge p$)
—redundant logical strength, e.g., `untell`(p) entails `untell`($p \wedge q$).
—the *kind* of communicated information, e.g., simple constraints on the domain of discourse or information containing speech acts.

Note that the basic communication mechanism is *synchronous*, i.e., the receipt of information takes place at the same time at which it is sent. Our choice for synchronous communication is motivated by the fact that in the field of concurrency theory, asynchronous communication can be modelled in terms of synchronous communication [26]. After the introduction of the syntax of the programming language we will show a corresponding modelling of asynchronous communication in our agent framework by means of communication facilitators.

In order to give an example of some basic communications, we assume the following rule for the KQML speech act ask, which is used to ask for specific information:

$$\mathtt{ask}(\varphi) \vdash \mathtt{ask}(\psi) \;\Leftrightarrow\; \psi \vdash \varphi.$$

This rule expresses the anti-monotonicity of the ask speech act, which allows dialogues like the following.

Example 3.1. Consider an agent that performs the output action $c!\mathtt{ask}(p)$ to send the question $\mathtt{ask}(p)$ along the channel c, followed by an input action $c?p$ to anticipate the receipt of the answer p. Additionally, consider an agent that executes the input action $c?\mathtt{ask}(p \wedge q)$ to anticipate the receipt of the question $\mathtt{ask}(p \wedge q)$ along c followed by the output action $c!p \wedge q$ to send the answer $p \wedge q$. The two agents' subsequent actions can be synchronised since $\mathtt{ask}(p) \vdash \mathtt{ask}(p \wedge q)$ and $p \wedge q \vdash p$.

We have the following basic programming constructs to build more complex agent behaviour: an operator '$\cdot$' for sequential composition, '&' for parallelism inside an agent, '$\sum$' for non-deterministic choice, 'loc' for local variable declarations, $P(x)$ denoting a procedure call and finally, skip denoting the empty statement. Additionally, for the construction of multiagent systems we have an operator '$\|$' denoting the parallel composition of agents.

The syntax of the core of the programming language ACPL is as follows.

Definition 3.2 (Syntax of programming language). Given a countable (possibly infinite) index set I, statements S and multiagent systems A are defined as follows:

$$\begin{aligned} S ::= {} & \mathtt{update}(\varphi) \cdot S \mid c!\psi \cdot S \mid \sum_{i \in I} c_i?\psi_i \cdot S_i \mid \\ & \sum_{i \in I} \mathtt{query}(\varphi_i) \cdot S_i \mid S_1 \;\&\; S_2 \mid \mathtt{loc}_y S \mid P(y) \mid \mathtt{skip} \\ A ::= {} & (D_1, S_1) \parallel \cdots \parallel (D_n, S_n) \end{aligned}$$

where φ and φ_i denote basic constraints and ψ and ψ_i may also contain speech act operators, c and c_i are channels, and the variable y ranges over a set *Var* of logical variables. Additionally, for all $1 \leq i \leq n$, the set D_i consists of procedure declarations of the form $P(x) :- S$, where P is the name of the procedure from a set *Proc* of procedure names, x is its formal parameter and the statement S constitutes its body.

For technical convenience, we assume that each procedure declaration has one and the same formal parameter x. If the set D_i is empty or clear from the context, it is usually omitted from notation, in which case an agent (D_i, S_i) is simply written as S_i. We write the non-deterministic choice $\sum_{i\in I} \texttt{query}(\varphi_i) \cdot S_i$ also as follows:

$$\texttt{query}(\varphi_1) \cdot S_1 + \cdots + \texttt{query}(\varphi_n) \cdot S_n,$$

and the same for $\sum_{i\in I} c_i?\varphi_i \cdot S_i$. Additionally we write $a \cdot \texttt{skip}$ simply as a, for each action a. Note that belief states are not part of the syntax of the programming language. As we will see in Section 4, they are an important part of the operational model.

We illustrate the language by means of the following example.

Example 3.3. We consider a situation of two agents that are involved in a conversation of which the goal is to establish that the proposition p holds [5]. The individual programs of these agents, *Agent1* and *Agent2*, are depicted in Figure 4 and Figure 5, respectively. We assume that the conversation proceeds along the communication channels c and d.

The first step of *Agent1* is to send the question $\texttt{ask}(p)$. After this, it anticipates two possible replies from its communication partner: an answer p, after which the belief state is updated with p and the agent terminates, or a counter question $\texttt{ask}(q_1 \wedge q_2)$. Note that for any other reply of its communication partner no reaction is defined; such replies will thus lead to a deadlock. However, the program can be straightforwardly accommodated to handle other types of replies, but for reasons of brevity we have not done this here.

In the latter case, after receipt of a counter question $\texttt{ask}(q_1 \wedge q_2)$ there are two possible scenarios. First, the agent verifies that $q_1 \wedge q_2$ follows from its belief state, after which the answer $q_1 \wedge q_2$ is sent, the reply p is anticipated, and upon receipt, p is added to the belief state. Or secondly, the implication $(r_1 \vee r_2) \rightarrow (q_1 \wedge q_2)$ holds,[1] after which the counter question $\texttt{ask}(r_1 \vee r_2)$ is sent, the corresponding answer

```
c!ask(p)·
{ d?p·
   update(p) }
+
{ d?ask(q₁ ∧ q₂)·
   { query(q₁ ∧ q₂)·
      c!q₁ ∧ q₂·
      d?p·
      update(p) }
   +
   { query((r₁ ∨ r₂) → (q₁ ∧ q₂))·
      c!ask(r₁ ∨ r₂)·
      d?r₁ ∨ r₂·
      update(r₁ ∨ r₂)·
      c!q₁ ∧ q₂·
      d?p·
      update(p) } }
```

Figure 4. Program of *Agent1*.

```
c?ask(p)·
{ query(p)·
   d!p }
+
{ query(q1 → p)·
   d!ask(q1)·
   { c?q1·
      update(q1)·
      d!p }
   +
   { c?ask(r1)·
      query(r1)·
      d!r1·
      c?q1·
      update(q1)·
      d!p } }
```

Figure 5. Program of *Agent2*.

$r_1 \vee r_2$ is anticipated, upon receipt, the information $r_1 \vee r_2$ is added to the store, the reply $q_1 \wedge q_2$ is sent to the communication partner, the answer p is anticipated, after which, finally, the information p is added to the agent's beliefs.

The program of *Agent2* is similar. Initially, it anticipates the receipt of a question $\mathtt{ask}(p)$. Then, upon receipt, if the proposition p immediately follows from its belief state it replies with the answer p and terminates, or alternatively, if the implication $q_1 \rightarrow p$ holds then it replies with the counter question $\mathtt{ask}(q_1)$. In the latter case, it anticipates two possible replies of its communication partner: an answer q_1 after which q_1 is added to the store and the agent truthfully sends the reply p, or a counter question $\mathtt{ask}(r_1)$ after which the agent checks that r_1 holds, sends the reply r_1, anticipates a reply q_1, upon receipt, adds q_1 to its belief state and finally replies with p.

In the following, we will use the name *TwoAgents* to refer to the multiagent system that consists of the parallel composition *Agent1* $\|$ *Agent2* of the above agents.

In our framework, the basic form of dialogue is synchronous. However, various forms of asynchronous communication can be modelled. In the next example, we show how we can model a form of dialogue that consists of sending asynchronously a question and subsequently receiving the answer without waiting for it. The question will be handled by a communication facilitator that is associated with the addressed agent.

Example 3.4. Sending a question $\mathtt{ask}(\varphi)$ along a channel c without waiting for its answer (to be received along a channel d) can be described by the following code:

$$c!\mathtt{ask}(\varphi) \cdot (S_1 \;\&\; (d?\varphi \cdot S_2)),$$

where S_1 represents the remaining activities of the agent and S_2 represents the agent's subsequent response to the answer.

Secondly, the corresponding receipt of a question $\mathtt{ask}(\psi)$ along a channel c, where $\psi \vdash \varphi$, will be handled by the addressed agent's communication facilitator

Fac which satisfies the following recursive equation:

$$Fac :- c?\mathtt{ask}(\psi) \cdot ((i?(\psi) \cdot d!(\psi) \;\&\; Fac),$$

where i denotes an internal channel connecting the facilitator with the agent. Note that consequently this facilitator in fact describes a bag of received requests (along channel c) for answering ψ.

Obviously, this basic form of asynchronous dialogue can be extended to more involved patterns of interaction.

As an additional example we consider the KQML speech acts `ask_one`, which is used to ask for one instantiation of the specified question. In the concurrency framework of ACP [3], value-passing can be modelled by synchronisation of actions and non-deterministic choice. Similarly, in our framework, we can model the generation and communication of solutions to constraints as described by the KQML speech acts like `ask_one` in the following way.

Example 3.5. Anticipated responses (to be received along channel c) to the KQML expression $\mathtt{ask\_one}(\varphi)$ can be modelled as follows:

$$\sum_{i \in I} c?\varphi(\theta_i) \cdot S_i,$$

where the set $\{\theta_i \mid i \in I\}$ denotes the set of all suitable substitutions, $\varphi(\theta_i)$ denotes the application of the substitution θ_i to φ and S_i represents the corresponding subsequent reaction.

For instance, consider the question $\mathtt{ask\_one}(\mathit{price}(x, \mathit{item464}))$ to ask for the price of a particular item. Let us assume that prices can be any natural number between 0 and 1000. The reception of an answer to the question can then be described by:

$$\sum_{i \in [0..1000]} c?\mathit{price}(i, \mathit{item464}) \cdot S_i.$$

We assume for each multiagent system an underlying network of communication channels.

Definition 3.6 (Communication channels)**.** We consider communication channels that are *one-to-one*, which means that each channel is a connection between two different agents. Additionally, we assume channels to be *directed*, which means that one of the agents uses it to send information, but cannot use it to receive information, while the other agent uses it to receive information, but cannot use it to send information. In particular, a communication channel c is of the form:

$$(c^{in}, c^{out}).$$

The first element c^{in} of the tuple denotes the *input port* of the channel, to which information is sent, and c^{out} constitutes the *output port*, along which information can be received. Given a multiagent system, we use the notation $port_i$ to denote the set of ports the agent i in the system can use.

For instance, in the multiagent system *TwoAgents* from Example 3.3, the set $port_1$ that is associated with *Agent1* consists of the ports c^{out} and d^{in}, while the set $port_2$ of *Agent2* is comprised of the ports c^{in} and d^{out}.

4. Operational semantics

In this section, we define the operational semantics of the multiagent programming language.

4.1. Histories

In the operational model of the language, there is a central role for the histories of communication. These histories are composed of the visible communication actions that are performed by an agent, and as such constitute its *interface* to the other agents in a system. The purpose of the histories is to enable a compositional proof system, which does not assume any knowledge about the internal structure of the agents in a system other than their interface.

Definition 4.1 (Local and global histories). Given a constraint system $\mathscr{C}$, we define the set $\mathscr{H}_{loc}$ of *local* histories to be the smallest set that satisfies $\varepsilon \in \mathscr{H}_{loc}$, where ε denotes the empty history, and secondly, if $h \in \mathscr{H}_{loc}$, $c \in chan$ and $\varphi \in \mathscr{C}$, then $h \cdot (c, \varphi) \in \mathscr{H}_{loc}$. Additionally, the set $\mathscr{H}_{glob}$ of *global* histories is the smallest set that satisfies $\varepsilon \in \mathscr{H}_{glob}$ and if $h \in \mathscr{H}_{glob}$, $c \in chan$ and $\varphi, \psi \in \mathscr{C}$ with $\varphi \vdash \psi$, then $h \cdot (c, \varphi, \psi) \in \mathscr{H}_{glob}$. Moreover, the set $\mathscr{H}$ of *histories* is the union of the sets $\mathscr{H}_{loc}$ and $\mathscr{H}_{glob}$. Usually, we abbreviate $\varepsilon \cdot h$ to h.

A *local* history is a sequence of elements of the form (c, φ), which records a communication step along the channel c that involves the information φ. Depending on the fact with which agent the local history is associated, the information φ either represents information that has been *sent* or information that has been anticipated to be *received* along the channel c. For instance, for an agent that is associated with the input ports c^{in} and the output port d^{out}, the following local history:

$$(c, \varphi) \cdot (c, \psi) \cdot (d, \varphi \wedge \psi)$$

records a sequence of communication steps in which first the receipt of the information φ is anticipated along c, followed by an anticipated receipt of the information ψ via c, and concluded with the emittance of the information $\varphi \wedge \psi$ along the channel d. On the other hand, for an agent that is associated with the output port c^{out} and input port d^{in}, the sequence $(c, \varphi) \cdot (c, \psi) \cdot (d, \psi)$ denotes a local communication history in which first the information φ is sent along the channel c, which is succeeded by the emittance of the information ψ along c, and followed by the anticipated receipt of the information ψ along the channel d.

A *global* history is a sequence of elements that are of the form (c, φ, ψ), which records a communication step along the channel c in which the information φ is sent and the information ψ is anticipated to be received, where it is required that the information φ entails the information ψ. For instance, the following global history:

$$(c, \varphi, \varphi) \cdot (c, \psi, \psi) \cdot (d, \varphi \wedge \psi, \psi)$$

denotes a sequence of successive communication steps. In the first step, the information φ is sent and anticipated to be received along the channel c, while in the second communication step the information ψ is both sent and anticipated along c. Finally, in the third communication, the information $\varphi \wedge \psi$ is sent along the channel d while the information ψ is anticipated. The latter is a valid communication step since the information ψ is entailed by the sent information $\varphi \wedge \psi$.

We define the following operations on local and global histories.

Definition 4.2 (Operations on histories)**.**

—Consider a global history $h \in \mathcal{H}_{glob}$ and a set G of ports such that

$$c^{in} \in G \Rightarrow c^{out} \notin G$$

for all channels c, which means that G does not contain both the input and the output port of a channel. The reason for this assumption is that G represents the ports that are used by a particular agent, which cannot have access to both the input and output port of a channel. We define the local history $h \upharpoonright G$ in $\mathcal{H}_{loc}$ as follows:

$$\varepsilon \upharpoonright G = \varepsilon$$

$$h \cdot (c, \varphi, \psi) \upharpoonright G = \begin{cases} h \upharpoonright G \cdot (c, \varphi) & \text{if } c^{out} \in G \\ h \upharpoonright G \cdot (c, \psi) & \text{if } c^{in} \in G \\ h \upharpoonright G & \text{otherwise} \end{cases}$$

— Given a history $h \in \mathcal{H}_{loc}$ and a variable $x \in Var$, we define $\exists_x h$ to be the history that derives from h by replacing each constraint φ that occurs in a communication record in h by the constraint $\exists_x \varphi$.

— For each history $h \in \mathcal{H}_{loc}$ and for all variables x and y in Var, we define $h[y/x]$ to be the local history that derives from h by replacing each constraint φ by the constraint $\varphi[y/x]$, which denotes φ in which the variable x is renamed to y.

— Given a natural number n and a history $h \in \mathcal{H}$, we use the notation $h[n]$ to denote the n-th element of h. Additionally, we write $\#h$ to denote the number of communication records in h.

Thus, $h \upharpoonright G$ is a local history that derives from the global history h by removing all communication records that do not involve a port in G and additionally, by replacing each communication record (c, φ, ψ) by the record (c, φ) if c^{out} in G and by (c, ψ) if c^{in} in G. For instance, let the global history h be given by:

$$(c, \varphi, \varphi) \cdot (c, \psi, \psi) \cdot (d, \varphi \wedge \psi, \psi),$$

the projection $h \upharpoonright \{c^{in}, d^{out}\}$ of h with respect to the ports c^{in} and d^{out} is defined by $(c, \varphi) \cdot (c, \psi) \cdot (d, \varphi \wedge \psi)$, while the projection $h \upharpoonright \{c^{out}, d^{in}\}$ is given by $(c, \varphi) \cdot (c, \psi) \cdot (d, \psi)$.

Furthermore, $\exists_x h$ denotes the application of the hiding operator $\exists_x$ to each constraint in the local history h. For instance, we have that $\exists_x((c, \varphi) \cdot (c, \psi))$ is given by the history $(c, \exists_x\varphi) \cdot (c, \exists_x\psi)$. Moreover, $h[y/x]$ denotes the application of the renaming operator $[y/x]$ to each constraint in h, as illustrated by: $((c, \varphi) \cdot (c, \psi))[y/x]$ which is equal to $(c, \varphi[y/x]) \cdot (c, \psi[y/x])$. Finally, for $h = (c, \varphi) \cdot (c, \psi)$, we have $h[1] = (c, \varphi)$ and $h[\#h] = (c, \psi)$.

4.2. *Transition system*

We define the operational semantics of the programming language by means of a transition system, which is used for the formal derivation of transitions [38]. A transition is of the form: $A \xrightarrow{l} A'$, which denotes a computation step of the multiagent system A, where A' constitutes the resulting configuration of the system. The label l can be of several forms: a label ε is employed to denote that a transition does not involve communication, a label (c, φ) is used to denote that the transition is a local communication step in which the information φ is either sent or anticipated to be received along the channel c (depending on which port of the channel is employed), and a label of the form (c, φ, ψ) is employed to denote a global communication step in which the information φ is sent and the information ψ is anticipated to be received along the channel c, such that φ entails ψ.

The transition system is defined over agent configurations that are of the form:

$$\langle D, S, \psi \rangle,$$

where D is a set of procedure declarations, the statement S denotes the agent program to be executed and ψ denotes the agent's belief state or information store, which is represented by a basic constraint from the constraint system. Note that a finite set of constraints can be represented by one constraint. Normally, if the set D of procedure declarations is clear from the context we omit it from notation by writing the configuration simply as $\langle S, \psi \rangle$. Additionally, a configuration of a multiagent system that consists of n different agents, is of the form:

$$\langle A, \sigma \rangle,$$

where A is the parallel composition of the individual agent programs and $\sigma : [1, n] \rightarrow \mathscr{C}$ denotes the *state* of the system, which is a function that associates with each agent $i \in [1, n]$ in the system its corresponding belief state $\sigma(i)$, which is represented by a basic constraint from the underlying constraint system $\mathscr{C}$.

In defining the dynamics of this state, we use the variant notation $\sigma\{i \mapsto \varphi\}$, which denotes the function that behaves like σ except for the input i, for which it yields the output φ. Moreover, we write $\sigma\{i \mapsto \varphi, j \mapsto \psi\}$ instead of $\sigma\{i \mapsto \varphi\}\{j \mapsto \psi\}$.

Definition 4.3 (Operational semantics). We define the following local and global transition rules:

T1 $\langle \texttt{skip}, \psi\rangle \xrightarrow{\varepsilon} \langle E, \psi\rangle$

T2 $\langle c!\varphi \cdot S, \psi\rangle \xrightarrow{(c,\varphi)} \langle S, \psi\rangle$

T3 $\langle \texttt{update}(\varphi) \cdot S, \psi\rangle \xrightarrow{\varepsilon} \langle S, \psi \wedge \varphi\rangle$

T4 $\langle \sum_{i\in I} \texttt{query}(\varphi_i) \cdot S_i, \psi\rangle \xrightarrow{\varepsilon} \langle S_j, \psi\rangle$ if $j \in I$ and $\psi \vdash \varphi_j$

T5 $\langle \sum_{i\in I} c_i?\varphi_i \cdot S_i, \psi\rangle \xrightarrow{(c_j,\varphi_j)} \langle S_j, \psi\rangle$ if $j \in I$

T6 $$\frac{\langle S_1, \psi\rangle \xrightarrow{h} \langle S_1', \psi'\rangle}{\begin{array}{l}\langle S_1 \,\&\, S_2, \psi\rangle \xrightarrow{h} \langle S_1' \,\&\, S_2, \psi'\rangle\\ \langle S_2 \,\&\, S_1, \psi\rangle \xrightarrow{h} \langle S_2 \,\&\, S_1', \psi'\rangle\end{array}}$$

T7 $$\frac{\langle S, \exists_x\psi \wedge \varphi\rangle \xrightarrow{h} \langle S', \psi'\rangle}{\langle \texttt{loc}_x^{\varphi} S, \psi\rangle \xrightarrow{\exists_x h} \langle \texttt{loc}_x^{\psi'} S', \psi \wedge \exists_x\psi'\rangle}$$

T8 $$\frac{\langle S, \exists_x\psi \wedge x = y\rangle \xrightarrow{h} \langle S', \psi'\rangle}{\langle \texttt{ren}_x^y S, \psi\rangle \xrightarrow{h[y/x]} \langle \texttt{ren}_x^y S', \psi \wedge \exists_x\psi'\rangle}$$

T9 $\langle D, P(y), \psi\rangle \xrightarrow{\varepsilon} \langle D, \texttt{ren}_x^y S, \psi\rangle$ where $P(x) :\!- S \in D$.

T10 $$\frac{\langle S_i, \sigma(i)\rangle \xrightarrow{\varepsilon} \langle S_i', \psi_i\rangle}{\langle \cdots \parallel S_i \parallel \cdots, \sigma \rangle \xrightarrow{\varepsilon} \langle \cdots \parallel S_i' \parallel \cdots, \sigma\{i \mapsto \psi_i\}\rangle}$$

T11 $$\frac{\langle S_i, \sigma(i)\rangle \xrightarrow{(c,\varphi)} \langle S_i', \psi_i\rangle, \langle S_j, \sigma(j)\rangle \xrightarrow{(c,\psi)} \langle S_j', \psi_j\rangle}{\langle \cdots S_i \parallel \cdots \parallel S_j \cdots, \sigma\rangle \xrightarrow{(c,\varphi,\psi)} \langle \cdots S_i' \parallel \cdots \parallel S_j' \cdots, \sigma'\rangle}$$

where agent i is associated with the output port c^{out} and agent j with the input port c^{in} and $\sigma' = \sigma\{i \mapsto \psi_i, j \mapsto \psi_j\}$. Note that in the communication record (c, φ, ψ) it is required that $\varphi \vdash \psi$.

The transition **T1**, in which the symbol E denotes successful termination, shows that the statement `skip` leaves the agent's information store ψ intact and does not involve communication as denoted by the label ε.

In the transition **T2**, the label (c, φ) expresses that the information φ is sent along the (output port of the) channel c. Note that we do not assume the *sincerity condition* [37] here; that is, the communicated information φ is not required to follow from the agent's information store ψ. However, a programmer can accomplish sincerity by letting each communication action precede by a test that φ indeed follows from the information store:

$$\texttt{query}(\varphi) \cdot c!\varphi.$$

In this case, the communication action will only be executed in case the test $\texttt{query}(\varphi)$ has been successfully executed first.

Transition **T3** shows that the execution of the action $\texttt{update}(\varphi)$ amounts to the addition of the information φ to the belief state ψ. Additionally, **T4** defines that if the information φ_j follows from the information store ψ, the non-deterministic choice over a set of statements, each of which is guarded by a query action, evolves itself to a configuration in which S_j denotes the statement to be executed next. Note that in case none of the constraints φ_j follows from the store, the execution of the non-deterministic choice is suspended.

In the transition rule **T5**, the non-deterministic choice over a set of statements, each of which is guarded by an action $c_i?\varphi_i$, can evolve itself to a configuration in which the statement S_j is to be executed next. The label (c_j, φ_j) denotes the anticipation of the receipt of the information φ_j along the (input port of the) channel c_j. Note that the information φ_j is not automatically added to the belief state. The addition of this information can be established through a subsequent execution of the update operator:

$$c_j?\varphi_j \cdot \texttt{update}(\varphi_j).$$

Again, note that in case none of the constraints φ_j is received, the execution of the non-deterministic choice is suspended.

The rule **T6** for internal parallelism shows how the transition of a parallel statement can be derived from a transition of one of its parallel operands.

With respect to the transition rule **T7**, the syntax is extended with a construct of the form $\texttt{loc}_x^{\varphi}S$, where x is a local variable and φ collects the information about this local variable. In particular, using this construction, the statement $\texttt{loc}_xS$ can be modelled as $\texttt{loc}_x^{true}S$, which reflects that initially there are no constraints on the local variable x.

The idea of rule **T7** is that the transition of the construct $\texttt{loc}_x^{\varphi}S$ is derived from the transition of the statement S. In order to achieve this, we need to replace the information about the global variable x in the state ψ (if present) by the information φ about the local variable x. This yields the state $\exists_x\psi \wedge \varphi$. The statement S is then executed relative to this state. After the computation step, ψ' denotes the new state and S' represents the part of S that still remains to be executed. In order to obtain from ψ' the resulting store, we remove from it the information about the local variable x and add the remainder to the old information store ψ, yielding the new state $\psi \wedge \exists_x\psi'$. Note that it is essential here that the information store exhibits a monotonically increasing behaviour, otherwise it would make no sense to incorporate ψ in the resulting state. Finally, the information ψ' concerning the local variable x needs to be stored for later use; hence, the statement $\texttt{loc}_x^{\psi'}S'$ denotes the part of the program that needs to be executed next.

Additionally, whereas information about *global* variables can be communicated to other agents, the idea of an agent's *local* variables is that they are private to the agent and therefore cannot be referred to in communications. This explains why in **T7** the information about the local variable x in a communication is existentially quantified, yielding a label $\exists_x h$ of the form $(c, \exists_x \varphi)$ in case h is of the form (c, φ).

In order to facilitate the operational description of procedure calls, we introduce a special renaming construct of the form $\mathtt{ren}_x^y S$, which denotes the statement S in which the variable x is renamed to y. The transition rule **T8** for renaming is similar to **T7**.

Additionally, the transition of a procedure call is given by an execution of the body of the procedure in which the formal parameter x is renamed to y, as described by **T9**. We take $\mathtt{ren}_x^x S$ to be equal to S. That is, if the actual parameter of the procedure is the same as the formal one, then no renaming needs to take place.

Finally, the global transition of a multiagent system is derived from a local transition of one of its agents, as described by the rule **T10**, or is derived from the local transitions of two communicating agents, as defined in the transition rule **T11**, where the label (c, φ, ψ) reflects a communication step in which the information φ is sent along the channel c and ψ is anticipated to be received along it. It is required that ψ is entailed by φ.

4.3. *Observable behaviour of multiagent systems*

In this section, we define the behaviour of multiagent systems that is visible to an external observer. The observable behaviour of an *agent* is defined over the domain $\mathcal{D}_{loc} = \wp(\mathcal{C} \times \mathcal{H}_{loc})$, which consists of sets of *local traces* $[\varphi, h]$, where φ is the agent's final information store and h its local communication history. For a *multiagent system* that consists of n agents, the observational behaviour is defined over the domain $\mathcal{D}_{glob} = \wp(([1, n] \to \mathcal{C}) \times \mathcal{H}_{glob})$, which is comprised of sets of *global traces* $[\sigma, h]$, where σ is a function that assigns each agent identifier $i \in [1, n]$ its final information store $\sigma(i)$, and h is the global communication history of the system.

Additionally, we use the following notation:

$$A \overset{h}{\Longrightarrow} B$$

to denote that the system A can evolve itself into B through executing the communication actions as collected in the history h. Moreover, for each agent configuration (S, φ), we write $(S, \varphi) \not\longrightarrow$ to denote that it has terminated, which means that from (S, φ) there are no further transitions possible (neither an internal computation step nor a communication step). Additionally, for each multiagent system configuration (A, σ) we use the notation $(A, \sigma) \not\longmapsto$ to denote its termination, which means that each of its agents has terminated; that is, for each agent configuration $(S_i, \sigma(i))$ in A, we have that $(S_i, \sigma(i)) \not\longrightarrow$ holds. The behaviour of agents and multiagent systems that is visible to an external observer is defined as follows.

Definition 4.4 (Observable behaviour $\mathcal{O}$)**.** For each agent S and for each multiagent system A, the observable behaviour $\mathcal{O}(S) \subseteq \mathcal{D}_{loc}$ and the observable behaviour $\mathcal{O}(A) \subseteq \mathcal{D}_{glob}$ are defined as follows:

$$\mathcal{O}(S) = \{[\varphi, h] \mid \text{ exist } \varphi_0 \text{ and } T \text{ such that } \langle S, \varphi_0\rangle \overset{h}{\Longrightarrow} \langle T, \varphi\rangle \not\longrightarrow \}$$

$$\mathcal{O}(A) = \{[\sigma, h] \mid \text{ exist } \sigma_0 \text{ and } B \text{ such that } \langle A, \sigma_0\rangle \overset{h}{\Longrightarrow} \langle B, \sigma\rangle \not\longmapsto \}.$$

Thus, from an agent and multiagent system we observe the exhibited communication history and the final information stores, where we quantify over all possible initial stores φ_0 and σ_0. In particular, this notion of observable behaviour models the *strongest postcondition* of an agent and a multiagent system [12]. Moreover, this notion of observable behaviour is suited to observe successful computations and abstracts from situations of deadlock.[2] For an analysis of the deadlock behaviour of agent systems see [6].

Example 4.5. Let us consider the observable behaviour of the multiagent system *TwoAgents* from Example 3.3, which is depicted in Figure 6. It contains three different types of histories. The first history reflects the case that *Agent2* has the proposition p as one of its beliefs. The conversation then simply consists of the question $\texttt{ask}(p)$ followed by the answer p.

The second history additionally contains an embedded subconversation in which the question $\texttt{ask}(q_1)$ is posed while the question $\texttt{ask}(q_1 \wedge q_2)$ is anticipated. Successful communication takes place since q_1 is entailed by $q_1 \wedge q_2$. Additionally, the answer is $q_1 \wedge q_2$ is sent while the answer q_1 is anticipated. Again, successful communication takes place since q_1 is entailed by $q_1 \wedge q_2$. The third history comprises yet another embedded subconversation which involves the propositions r_1 and r_2.

Note that in all final states σ in $\mathcal{O}(\mathit{TwoAgents})$ the proposition p has become shared information; that is, p is entailed by both $\sigma(1)$ and $\sigma(2)$.

$$\begin{array}{l}
\mathcal{O}(\mathit{TwoAgents}) \\
= \\
\{[\sigma, h] \mid \sigma(1) \vdash p,\ \sigma(2) \vdash p, \text{ and} \\
h = (c, \texttt{ask}(p), \texttt{ask}(p)) \cdot (d, p, p)\} \\
\cup \\
\{[\sigma, h] \mid \sigma(1) \vdash (p \wedge q_1 \wedge q_2),\ \sigma(2) \vdash (p \wedge q_1) \text{ and} \\
h = (c, \texttt{ask}(p), \texttt{ask}(p)) \cdot (d, \texttt{ask}(q_1), \texttt{ask}(q_1 \wedge q_2)) \cdot \\
(c, q_1 \wedge q_2, q_1) \cdot (d, p, p)\} \\
\cup \\
\{[\sigma, h] \mid \sigma(1) \vdash (p \wedge q_1 \wedge q_2 \wedge (r_1 \vee r_2)), \\
\sigma(2) \vdash (p \wedge q_1 \wedge r_1) \text{ and} \\
h = (c, \texttt{ask}(p), \texttt{ask}(p)) \cdot (d, \texttt{ask}(q_1), \texttt{ask}(q_1 \wedge q_2)) \cdot \\
(c, \texttt{ask}(r_1 \vee r_2), \texttt{ask}(r_1)) \cdot (d, r_1, r_1 \vee r_2) \cdot \\
(c, q_1 \wedge q_2, q_1) \cdot (d, p, p)\}
\end{array}$$

Figure 6. Observable behaviour of *TwoAgents*.

4.4. *Compositional semantics*

In this section, we define a compositional semantics $[\![\ ]\!]$ for the multiagent language ACPL, satisfying the inclusion $\mathcal{O} \subseteq [\![\ ]\!]$, that will form the basis for the verification calculus developed in Section 6.

First, we define the following semantic operations on $\mathcal{D}_{loc}$.

Definition 4.6 (Operations on traces). We define the following semantic operators, where $\mathcal{T}$, $\mathcal{T}_1$ and $\mathcal{T}_2 \in \mathcal{D}_{loc}$.

$$\begin{aligned}
(c, \varphi) \cdot \mathcal{T} &= \{[\psi, (c, \varphi) \cdot h] \mid [\psi, h] \in \mathcal{T}\} \\
\mathcal{T}_1 \,\&\, \mathcal{T}_2 &= \{[\psi, h] \mid [\psi, h_1] \in \mathcal{T}_1, [\psi, h_2] \in \mathcal{T}_2, \\
&\qquad h \text{ is an interleaving of } h_1 \text{ and } h_2\} \\
\exists_x \mathcal{T} &= \{[\psi, h] \mid \text{exists } [\psi', h'] \in \mathcal{T} \text{ with} \\
&\qquad \exists_x \psi' = \exists_x \psi \text{ and } \exists_x h' = \exists_x h\} \\
\mathcal{T}[y/x] &= \{[\psi[y/x], h[y/x]] \mid [\psi, h] \in \mathcal{T}\}.
\end{aligned}$$

The set $(c, \varphi) \cdot \mathcal{T}$ consists of all local traces $[\psi, h]$ in $\mathcal{T}$ in which the history h is extended with the communication record (c, φ). Additionally, the set $\mathcal{T}_1 \,\&\, \mathcal{T}_2$ consists of the traces $[\psi, h]$, such that $[\psi, h_1]$ in $\mathcal{T}_1$ and $[\psi, h_2]$ in $\mathcal{T}_2$ and the history h can be obtained from an interleaving of the histories h_1 and h_2. The set $\exists_x \mathcal{T}$ consists of all traces that are equal to a trace in $\mathcal{T}$ except for the constraints on the variable x. For instance, for ψ and ψ' that are defined by:

$$\begin{aligned}
\psi &\equiv (y = x + 3) \wedge (x = 2) \\
\psi' &\equiv (y = x + 4) \wedge (x = 1),
\end{aligned}$$

we have $[\varphi, (c, \psi)] \in \exists_x \{[\varphi, (c, \psi')]\}$, since $\exists_x \psi = \exists_x \psi' = (y = 5)$. Finally, the set $\mathcal{T}[y/x]$ consists of all traces from $\mathcal{T}$ in which the variable x is renamed to y.

As sets of information stores that share some particular piece of information play an important role in the proof system, we introduce a special notation for them. That is, we introduce the notion of an *upward closure* of a constraint, which is given by the set of constraints that entail it.

Definition 4.7 (Closure of constraints). Given a constraint system $\mathcal{C}$, and a constraint $\varphi \in \mathcal{C}$, we define its upward closure as follows: $\uparrow \varphi = \{\psi \mid \psi \in \mathcal{C} \text{ and } \psi \vdash \varphi\}$.

The language ACPL contains recursive procedures. In order to model recursion, we employ an environment $e \in Env$ which is a partial function

$$e : Proc \to \mathcal{D}_{loc},$$

where *Proc* is a set of procedure names. The function e is used to map a procedure name $P \in Proc$ to its actual denotation $e(P)$ in $\mathcal{D}_{loc}$, which is given by the *least* *fixpoint* of the procedure with respect to the subset-ordering [31]. Given a function F, we use the notation μF to denote such a least fixpoint.

The compositional semantics $[\![\ ]\!]$ is defined as follows.

Definition 4.8 (Compositional semantics). For each agent S, its denotational semantics $[\![S]\!] : Env \to \mathscr{D}_{loc}$ is defined as follows:

$$
\begin{aligned}
[\![\mathtt{skip}]\!](e) &= \mathscr{C} \times \{\varepsilon\}\\
[\![c!\varphi \cdot S]\!](e) &= (c, \varphi) \cdot [\![S]\!](e)\\
[\![\mathtt{update}(\varphi) \cdot S]\!](e) &= (\uparrow \varphi \times \mathscr{H}_{loc}) \cap [\![S]\!](e)\\
\left[\!\!\left[\sum_i \mathtt{query}(\varphi_i) \cdot S_i \right]\!\!\right](e) &= \bigcap_i ((\mathscr{C} \setminus \uparrow \varphi_i) \times \{\varepsilon\}) \cup\\
&\quad \bigcup_i (\uparrow \varphi_i \times \mathscr{H}_{loc}) \cap [\![S_i]\!](e)\\
\left[\!\!\left[\sum_i c_i?\varphi_i \cdot S_i \right]\!\!\right](e) &= \bigcup_i (c_i, \varphi_i) \cdot [\![S_i]\!](e)\\
[\![S_1 \,\&\, S_2]\!](e) &= [\![S_1]\!](e) \,\&\, [\![S_2]\!](e)\\
[\![\mathtt{loc}_x S]\!](e) &= \exists_x [\![S]\!](e)\\
[\![D, P(y)]\!](e) &= [\![D, P(x)]\!](e)[y/x]\\
[\![D, P(x)]\!](e) &= \begin{cases} \mu F & \text{if } P(x) \;{:}{-}\; S \in D\\ e(P) & \text{otherwise}\end{cases}\\
&\quad \text{where} \quad F(\mathscr{T}) = [\![D', S]\!](e\{P \mapsto \mathscr{T}\})\\
&\qquad\qquad\quad D' = D \setminus \{P(x) \;{:}{-} S\}.
\end{aligned}
$$

Additionally, for each multiagent system S, its denotational semantics $[\![A]\!] \subseteq \mathscr{D}_{glob}$ is defined as follows:

$$[\![S_1 \parallel \cdots \parallel S_n]\!] = \{[\sigma, h] \mid (\sigma(i), h \restriction port_i) \in [\![S_i]\!], \text{ for all } i \in [1, n]\},$$

where $[\![S_i]\!]$ abbreviates $[\![S_i]\!](e)$ for all environments e.

Interpreting $[\![\ ]\!]$ as $\mathscr{O}$, we have that the observable behaviour of `skip` consists of the traces with an arbitrary information store (since we quantify over all initial information stores) together with an empty local communication history ϵ. From $c!\varphi \cdot S$ we observe the traces starting with the communication record (c, φ) followed by the observable behaviour of S. The behaviour that we observe from $\mathtt{update}(\varphi) \cdot S$ is given by the observable behaviour of S with the additional requirement that the traces' resulting information store entails φ.

The observable behaviour $\sum_i \mathtt{query}(\varphi_i) \cdot S_i$ is given by the observable behaviour of each S_i with the additional requirement that the traces' resulting information store entails φ_i. Additionally, the observable behaviour comprises all unsuccessfully terminating traces that contain an information store that does not entail any of the constraints φ_i and that contain an empty communication history.

From $\sum_i c_i?\varphi_i \cdot S_i$ we observe the traces that start with a communication record (c_i, φ_i) followed by the observable behaviour of S_i, for all i. The observable behaviour of the parallel composition $S_1 \,\&\, S_2$ is given by traces that form an interleaving of the communication histories of the traces of S_1 and S_2 that have the same final information store. The observable behaviour of $\texttt{loc}_x S$ is derived from the observable behaviour of S by abstracting from the variable x. The behaviour that we observe from a procedure call $P(y)$ is derived from that of a procedure declaration $P(x)$ (remember our assumption that each procedure has x as its formal parameter) by renaming the variable x to y.

The observable behaviour of a procedure declaration $P(x)$ is given by its least fixpoint. Such a fixpoint is recorded in the environment e. In the equation for $[\![P(x)]\!]$, the procedure declaration is removed from the set D yielding the set D' and its denotation $\mathcal{T}$ is added to the environment e; i.e., $e\{P \mapsto \mathcal{T}\}$ denotes the function that behaves like e except for the input P for which it yields the output $\mathcal{T}$. Note that only in the equations for procedure calls, we have made explicit the set D of procedure declarations; in the other cases, this set is omitted from notation.

Finally, from a multiagent system we observe global traces that consist of a store σ collecting the individual agent belief states $\sigma(i)$ and a global communication history h that is in correspondence with the local communication histories $h \restriction port_i$, for each i.

We have the following correspondence between the notion of observable behaviour and the above introduced semantics $[\![\ ]\!]$.

Lemma 4.9 (Correspondence between $\mathcal{O}$ and $[\![\ ]\!]$). *For all statements S and agent systems A, we have*:

$$\mathcal{O}(S) \subseteq [\![S]\!]$$
$$\mathcal{O}(A) \subseteq [\![A]\!]$$

Proof: The claim follows from a slight modification of the corresponding proof for CCP in [8]. We omit the details here. □

The semantics $[\![\ ]\!]$ forms the basis for the verification calculus that is developed in Section 6. First, in Section 5, we design an assertion language to specify agent communication behaviour.

5. Assertion languages

In this section, we introduce a language that is suited to reason about the communication behaviour of agents and multiagent systems. It is comprised of a *local* and a *global* assertion language, which each contain two types of assertions: assertions that express properties of the computed information stores, and assertions that express properties of the communication history.

Definition 5.1 (Local assertion language). The language $\mathcal{I}_{loc}$ of local assertions is given by the following grammar, which defines the syntax of history expressions *he*, communication record expressions *ce*, number expressions *ne* and assertions Φ.

$$\begin{aligned}
he &::= t \mid \varepsilon \mid ce \cdot he \\
ce &::= (c, \varphi) \mid he[ne] \\
ne &::= n \mid i \mid \#he \\
\Phi &::= \mathbf{B}_n\psi \mid ce_1 = ce_2 \mid ne_1 \leq ne_2 \mid \neg\Phi \mid \\
&\qquad \Phi_1 \wedge \Phi_2 \mid \exists t\Phi \mid \exists x\Phi \mid \Phi[y/x] \mid \exists i\Phi
\end{aligned}$$

where t ranges over the set *His Var* of history variables, which we assume to contain a *distinguished* history variable h that has a special interpretation, namely the particular history that is under consideration. Moreover, c ranges over communication channels, φ over constraints, ψ over basic constraints, n over natural numbers constants, i over the variables to denote natural numbers, and x, y range over the logical variables in *Var*. Moreover, we assume the usual logical abbreviations for universal quantification $\forall t$ and $\forall i$, disjunction $\vee$, implication $\rightarrow$ and so on.

A history expression *he* in the language $\mathcal{I}_{loc}$ is either a history variable t, the empty history ε, or a communication record *ce* that is appended to a history *he*. Additionally, a communication record expression *ce* is either a communication record of the form (c, φ), or the *ne*-th element of a history *he*. Additionally, a number expression *ne* is either a natural number n, a number variable i that is used to denote indices of histories, or an expression of the form *#he* that denotes the length of the history *he*.

Finally, an assertion Φ of the form $\mathbf{B}_n\varphi$ denotes that the agent n has φ in its information store, or in other words that agent n *believes* the formula φ to hold. Furthermore, an assertion can be an equation between two communication record expressions, can be of the form $ne_1 \leq ne_2$ expressing that the natural number denoted by ne_1 is less than or equal to the natural number denoted by ne_2, can be the negation of an assertion, the conjunction of two assertions or an assertion in which is quantified over a variable t denoting a history. Additionally, an assertion can be of the form $\exists x\Phi$, which expresses that in the assertion Φ the variable x is a local variable, an assertion of the form $\Phi[y/x]$, which expresses that in the assertion Φ the variable x is renamed to y, or an assertion $\exists i\Phi$ in which is quantified over the number variable i.

The language $\mathcal{I}_{loc}$ constitutes a *basic* assertion language. It can be extended with more involved user-defined predicates.

Example 5.2. We consider a predicate $h_1 \leq h_2$ that is defined by:

$$h_1 \leq h_2 \;\Leftrightarrow\; \#h_1 \leq \#h_2 \wedge \forall i((1 \leq i \wedge i \leq \#h_1) \;\rightarrow\; h_1[i] = h_2[i]),$$

which expresses that the history h_1 is an initial prefix of the history h_2. Additionally, we define $h_1 = h_2 \;\Leftrightarrow\; h_1 \leq h_2 \wedge h_2 \leq h_1$, which expresses that two histories are equal if they are an initial prefix of each other.

Another example of a user-defined predicate is $h \in h_1 \,\&\, h_2$ defined by:

$$h \in \varepsilon \,\&\, \varepsilon \Leftrightarrow h = \varepsilon$$

$$h \in h_1 \,\&\, h_2 \Leftrightarrow h \in (h_1 \mathbin{|\!\!\lfloor} h_2) \;\vee\; h \in (h_2 \mathbin{|\!\!\lfloor} h_1)$$

$$h \in ((c, \varphi) \cdot h_1) \mathbin{|\!\!\lfloor} h_2 \Leftrightarrow \exists h_3 \; (h = (c, \varphi) \cdot h_3 \;\wedge\; h_3 \in (h_1 \,\&\, h_2)),$$

where $\mathbin{|\!\!\lfloor}$ denotes the *leftmerge* operator of [3]. Thus, the predicate $h \in h_1 \,\&\, h_2$ expresses that the history h is an element of the set of interleavings of the histories h_1 and h_2.

Definition 5.3 (Global assertion language)**.** The language $\mathcal{I}_{glob}$ of global assertions is given by the following grammar.

$$\begin{aligned}
he &::= t \mid \varepsilon \mid ce \cdot he \mid he \upharpoonright G \\
ce &::= (c, \varphi, \psi) \mid he[ne] \\
ne &::= n \mid i \mid \#he \\
\Phi &::= \mathbf{B}_n\varphi \mid ce_1 = ce_2 \mid ne_1 \leq ne_2 \mid \neg\Phi \mid \\
&\qquad \Phi_1 \wedge \Phi_2 \mid \exists t\Phi \mid \exists x\Phi \mid \Phi[y/x] \mid \exists i\Phi
\end{aligned}$$

where t ranges over the set *His Var* of history variables, which contains a *distinguished* history variable $\mathtt{h}$. Moreover, G ranges over sets of ports, c ranges over communication channels, φ and ψ over constraints, where in (c, φ, ψ) it is required that $\varphi \vdash \psi$, the meta-variable n ranges over natural numbers constants, i over the variables to denote natural numbers, and x, y range over the logical variables in *Var*.

The global assertion language $\mathcal{I}_{glob}$ is almost equal to the local assertion language $\mathcal{I}_{loc}$ except that it concerns *global* histories of communication records of the form (c, φ, ψ) and additionally contains history expressions that are of the form $he \upharpoonright G$, denoting the projection of the history he to the set G of ports.

Example 5.4. Consider a multiagent system that is comprised of an agent n with port c^{out} and an agent m with port c^{in}. We introduce the following global assertion Ψ:

$$\forall i(h[i] = (c, \mathtt{insert}(\varphi), \mathtt{insert}(\psi)) \rightarrow \mathbf{B}_m\psi),$$

where the KQML speech act `insert` is characterised by the following entailment relation:

$$\mathtt{insert}(\varphi) \vdash \mathtt{insert}(\psi) \;\Leftrightarrow\; \varphi \vdash \psi \;\text{ and }\; \psi \vdash \varphi.$$

Note that thus this rule requires that in a communication involving the actions $c!\mathtt{insert}(\varphi)$ and $c?\mathtt{insert}(\psi)$, the constraint φ is logically equivalent to ψ. So, for instance, we have that neither $c!\mathtt{insert}(p)$ and $c?\mathtt{insert}(p \wedge q)$ match nor do $c!\mathtt{insert}(p \wedge q)$ and $c?\mathtt{insert}(p)$.

The above assertion expresses that if at a particular point in the communication history h, communication between the agents n and m along the channel c has taken place, where `insert(`φ`)` is sent and `insert(`ψ`)` is anticipated to be received, and thus $\varphi \vdash \psi$ and $\psi \vdash \varphi$, then the agent m has ψ in its final store.

Note that this assertion can be used to verify that a multiagent system A complies with the semantics of the KQML speech act `insert`, i.e., to verify that $A sat\ \Psi$ holds (see Section 6).

Next, we consider the semantics of the local and global assertion language. First, we define the value of expressions.

Definition 5.5 (Value of expressions). The value $Val(e)(s)$ of an expression e is defined as follows, where the assignment s is a function that maps number variables to numbers and history variables to histories:

$$\begin{aligned}
Val(t)(s) &= s(t)\\
Val(\varepsilon)(s) &= \varepsilon\\
Val(ce \cdot he)(s) &= Val(ce)(s) \cdot Val(he)(s)\\
Val((c,\varphi))(s) &= (c,\varphi)\\
Val(he[ne])(s) &= Val(he)(s)[Val(ne)(s)]\\
Val(n)(s) &= n\\
Val(i)(s) &= s(i)\\
Val(\#he)(s) &= \#Val(he)(s)
\end{aligned}$$

We define the satisfaction relation $\varphi, h \models \Phi$, which expresses that the information store φ and the local history $h \in \mathcal{H}_{loc}$ satisfy the local assertion $\Phi \in \mathcal{I}_{loc}$.

Definition 5.6 (Satisfaction of local assertions). The satisfaction relation $\varphi, h, s \models \Phi$, where $h \in \mathcal{H}_{loc}$ is a local history, φ a constraint in $\mathcal{C}$, s an assignment and Φ and assertion in $\mathcal{I}_{loc}$, is defined as follows:

$$\begin{aligned}
&\varphi, h, s \models \mathbf{B}_n\psi \Leftrightarrow \varphi \vdash \psi\\
&\varphi, h, s \models (ce_1 = ce_2) \Leftrightarrow Val(ce_1)(s) = Val(ce_2)(s)\\
&\varphi, h, s \models (ne_1 \leq ne_2) \Leftrightarrow Val(ne_1)(s) \leq Val(ne_2)(s)\\
&\varphi, h, s \models \neg\Phi \Leftrightarrow \varphi, h, s \not\models \Phi\\
&\varphi, h, s \models \Phi_1 \wedge \Phi_2 \Leftrightarrow \varphi, h, s \models \Phi_1 \text{ and } \varphi, h, s \models \Phi_2\\
&\varphi, h, s \models \exists t\Phi \Leftrightarrow \text{exists } h' \in \mathcal{H}_{loc} \text{ with } \varphi, h, s\{t \mapsto h'\} \models \Phi\\
&\varphi, h, s \models \exists x\Phi \Leftrightarrow \text{exist } \varphi' \text{and } h' \text{ with } \varphi', h', s\{\mathsf{h} \mapsto h'\} \models \Phi,\\
&\qquad\qquad \exists_x\varphi = \exists_x\varphi' \text{ and } \exists_x h = \exists_x h'\\
&\varphi, h, s \models \Phi[y/x] \Leftrightarrow \text{exist } \varphi' \text{ and } h' \text{ with } \varphi', h', s\{\mathsf{h} \mapsto h'\} \models \Phi,\\
&\qquad\qquad \varphi = \varphi'[y/x] \text{ and } h = h'[y/x]\\
&\varphi, h, s \models \exists i\Phi \Leftrightarrow \text{exists } n \in \mathbb{N} \text{ with } \varphi, h, s\{i \mapsto n\} \models \Phi,
\end{aligned}$$

where $\mathbb{N}$ denotes the natural numbers. Additionally, $\varphi, h \models \Phi$ holds if $\varphi, h, s \models \Phi$ for all assignments s with $s(\mathsf{h}) = h$. Finally, we have $\models \Phi$, if $\varphi, h \models \Phi$ holds for all φ and h.

Thus, the construction $\varphi, h \models \Phi$ expresses that the information store φ and the history h satisfy the assertion Φ, in which the distinguished history variable h is interpreted as the history h.

For instance, if $\varphi \vdash \psi$ we have:

$$\varphi, h_1 \cdot (c, \mathtt{insert}(\psi)) \cdot h_2 \models \forall i\ (\mathsf{h}[i] = (c, \mathtt{insert}(\psi)) \rightarrow \mathbf{B}_m\psi),$$

where h_1 and h_2 denote some local histories that do not contain the speech act `insert`.

Definition 5.7 (Satisfaction of global assertions). The definition of the satisfaction relation $\sigma, h, s \models \Phi$ for global states σ, global histories $h \in \mathcal{H}_{glob}$ and global assertions Φ proceeds in a similar way as that for local assertions; that is, the clauses need to be lifted to the level of multiagent systems. We mention $\sigma, h, s \models \mathbf{B}_n\psi \Leftrightarrow \sigma(n) \vdash \psi$. Additionally, we need to add one extra clause to the definition of the value of expressions: $Val(he \restriction G)(s) = Val(he)(s) \restriction G$.

For instance, if $\sigma(m) \vdash \psi$ and φ is logically equivalent to ψ we have:

$$\sigma, h_1 \cdot (c, \mathtt{insert}(\varphi), \mathtt{insert}(\psi)) \cdot h_2 \models$$
$$\forall i\ (\mathsf{h}[i] = (c, \mathtt{insert}(\varphi), \mathtt{insert}(\psi)) \rightarrow \mathbf{B}_m\psi),$$

where h_1 and h_2 denote some global histories that do not contain the speech act `insert`.

6. Compositional verification calculus

In this section, we define a compositional proof system for the programming language ACPL. Let us first define the notion of a correctness formula.

Definition 6.1 (Correctness formula). A *local correctness formula* is of the form: $S\ sat\ \Phi$, where S is an agent and Φ is a local assertion in $\mathcal{I}_{loc}$. Additionally, a *global correctness formula* is of the form: $A\ sat\ \Phi$, where A is a multiagent system and Φ is a global assertion in $\mathcal{I}_{glob}$.

The semantics of correctness formula is defined as follows.

Definition 6.2 (Validity of correctness formula). A local correctness formula $S\ sat\ \Phi$ is *valid*, which we write as $\models S\ sat\ \Phi$, if for all $[\varphi, h] \in \mathcal{O}(S)$ we have

$\varphi, h \models \Phi$. A global correctness formula $A\ sat\ \Phi$ is *valid*, which we write as $\models A\ sat\ \Phi$, if for all $[\sigma, h] \in \mathcal{O}(A)$ we have $\sigma, h \models \Phi$.

We define the following proof system to reason about the correctness of agents and multiagent systems. It consists of one axiom and a collection of rules, which closely follow the equations of the compositional semantics $[\![\ ]\!]$.

Definition 6.3 (Proof rules). We write $\vdash S\ sat\ \Phi$ to denote that the correctness formula $S\ sat\ \Phi$ can be derived by means of the following axiom and proof rules. Similarly, we use the notation $\vdash A\ sat\ \Phi$.

R1 $\texttt{skip}\, sat\ \mathsf{h} = \varepsilon$

R2 $\dfrac{S\ sat\ \Phi}{c!\varphi \cdot S\ sat \Psi}$

where $\Psi \ \equiv\ \exists t\ (\Phi[t/\mathsf{h}] \wedge \mathsf{h} = (c, \varphi) \cdot t)$

R3 $\dfrac{S\ sat\ \Phi}{\texttt{update}(\varphi) \cdot S\ sat\ \Phi \wedge \mathbf{B}_j\varphi}$

where j denotes the identifier of the corresponding agent

R4 $\dfrac{S_i\ sat\ \Phi_i,\ \text{for all } i}{\sum_i \texttt{query}(\varphi_i) \cdot S_i\ sat\ \Phi}$

where $\Phi \ \equiv\ (\bigwedge_i \neg\mathbf{B}_j\varphi_i \wedge \mathsf{h} = \varepsilon) \vee \bigvee_i(\Phi_i \wedge \mathbf{B}_j\varphi_i)$ and j denotes the identifier of the corresponding agent

R5 $\dfrac{S_i\ sat\ \Phi_i,\ \text{for all } i}{\sum_i c_i?\varphi_i \cdot S_i\ sat\ \Phi}$

where $\Phi \ \equiv\ \bigvee_i \exists t\ (\Phi_i[t/\mathsf{h}] \wedge \mathsf{h} = (c_i, \varphi_i) \cdot t)$

R6 $\dfrac{S_1\ sat\ \Phi_1 S_2\ sat\ \Phi_2}{S_1 \,\&\, S_2\ sat\ \Phi}$

where $\Phi \ \equiv\ \exists t_1 t_2\ (\Phi_1[t_1/\mathsf{h}] \wedge \Phi_2[t_2/\mathsf{h}] \wedge \mathsf{h} \in (t_1 \,\&\, t_2))$

R7 $\dfrac{S\ sat\ \Phi}{\texttt{loc}_x S\ sat\ \exists x \Phi}$

R8 $\dfrac{P(x)\ sat\ \Phi}{P(y)\ sat\ \Phi[y/x]}$

R9 $\dfrac{D', P(x)\ sat\ \Phi \ \vdash\ D', S\ sat\ \Phi}{D, P(x)\ sat\ \Phi}$

where $P(x)$:$-S$ in D and $D' = D \setminus \{P(x) :\!- S\}$.

R10 $\dfrac{S\ sat\ \Phi \models \Phi \rightarrow \Psi}{S\ sat\ \Psi}$

$$\textbf{R11}\ \frac{S_i\ \mathit{sat}\ \Phi_i \text{ for all } i \in [1, n]}{S_1 \parallel \cdots \parallel S_n\ \mathit{sat}\ \Phi}$$

where $\Phi \equiv \bigwedge_i \Phi_i[(\mathsf{h} \upharpoonright \mathit{port}_i)/\mathsf{h}]$

$$\textbf{R12}\ \frac{A\ \mathit{sat}\ \Phi \models \Phi \rightarrow \Psi}{A\ \mathit{sat}\ \Psi}$$

Note that whereas $\Phi[y/x]$ constitutes an assertion, constructions of the form $\Phi[h/\mathsf{h}]$ are not assertions, but denote the *application* of the substitution of the history h for the history variable h in the assertion Φ.

Example 6.4 (Proof rule for multiagent systems). To illustrate rule **R11**, let us consider a system of two agents: an agent S_1 that employs the port c^{out} and an agent S_2 that employs the port c^{in}. We have:

$$\frac{S_1\ \mathit{sat}\ \mathsf{h} = (c, \varphi)\ S_2\ \mathit{sat}\ \mathsf{h} = (c, \psi)}{S_1 \parallel S_2\ \mathit{sat}\ \Phi}$$

where the global assertion Φ is given by:

$$\mathsf{h} \upharpoonright c^{out} = (c, \varphi) \ \wedge \ \mathsf{h} \upharpoonright c^{in} = (c, \psi),$$

which is logically equivalent to the assertion $\mathsf{h} = (c, \varphi, \psi)$.

Next, we show the soundness of the proof system.

Theorem 6.5 (Soundness).

— $\vdash S\ \mathit{sat}\ \Phi$ implies $\models S\ \mathit{sat}\ \Phi$,
— $\vdash A\ \mathit{sat}\ \Phi$ implies $\models A\ \mathit{sat}\ \Phi$.

Proof: We show a stronger result, namely that if $\vdash S\ \mathit{sat}\ \Phi$ then for all $(\varphi, h) \in [\![S]\!]$ we have $\varphi, h \models \Phi$. Via Lemma 4.9 we can subsequently derive $\models S\ \mathit{sat}\ \Phi$. Similarly, we show that if $\vdash A\ \mathit{sat}\ \Phi$ then for all $(\sigma, h) \in [\![A]\!]$ we have $\sigma, h \models \Phi$. The proof proceeds by induction on the length of the derivation.

First of all, we consider the axiom **R1**. Suppose $\vdash \mathtt{skip}\ \mathit{sat}\ \mathsf{h} = \varepsilon$. We have to show that for all $(\psi, h) \in [\![\mathtt{skip}]\!]$ we have $\psi, h \models (\mathsf{h} = \varepsilon)$. This amounts to checking $\psi, \varepsilon \models (\mathsf{h} = \varepsilon)$, which clearly holds.

For the induction step, we assume that for derivations of length n the claim holds. Consider a derivation of length $n + 1$. We consider the different possibilities of the last rule that has been applied in the derivation.

R2 Under the assumption $\models S\ \mathit{sat}\ \Phi$, we have to show that $\models c!\varphi \cdot\ S\ \mathit{sat}\ \Psi$ holds, where $\Psi \equiv \exists t\ (\Phi[t/\mathsf{h}] \wedge \mathsf{h} = (c, \varphi) \cdot t)$. Thus, we have to prove that for all $(\psi, h) \in [\![c!\varphi \cdot\ S]\!]$ it holds that $\psi, h \models \Psi$. Consider a pair $(\psi, h) \in [\![c!\varphi \cdot\ S]\!]$, then from the definition of $[\![\ ]\!]$ we know that there exists h' such that $h = (c, \varphi) \cdot h'$ and $(\psi, h') \in [\![S]\!]$, for which by the induction hypothesis we have $\psi, h' \models \Phi$. But then we also have $\psi, h \models \exists t\ (\Phi[t/\mathsf{h}] \wedge \mathsf{h} = (c, \varphi) \cdot t)$.

R3 Assuming $\models S\ \mathit{sat}\ \Phi$, we have to show $\models \mathtt{update}(\varphi) \cdot S\ \mathit{sat}\ \Phi \wedge \mathbf{B}_j\varphi$. Consider a pair $(\psi, h) \in [\![\mathtt{update}(\varphi) \cdot S]\!]$, for which we have to show $\psi, h \models \Phi \wedge \mathbf{B}_j\varphi$. From the definition of $[\![\ ]\!]$, we know $(\psi, h) \in [\![S]\!]$ and $\psi \in\ \uparrow \varphi$. Additionally, from the induction hypothesis it follows that $\psi, h \models \Phi$. Thus, we conclude $\psi, h \models \Phi \wedge \mathbf{B}_j\varphi$.

R4 We assume $\models S_i\ \mathit{sat}\ \Phi_i$ for all i. Consider a pair $(\psi, h) \in [\![\sum_i \mathtt{query}(\varphi_i) \cdot S_i]\!]$ for which we have to show $(\psi, h) \models \Phi$ with $\Phi \equiv (\bigwedge_i \neg\mathbf{B}_j\varphi_i \wedge \mathsf{h} = \varepsilon) \vee \bigvee_i(\Phi_i \wedge \mathbf{B}_j\varphi_i)$. The definition of $[\![\ ]\!]$ tells us that either $(\psi, h) \in \bigcap_i((\mathscr{C} \setminus \uparrow \varphi_i) \times \{\varepsilon\})$ holds, in which case we thus have $\psi, h \models (\bigwedge_i \neg\mathbf{B}_j\varphi_i \wedge \mathsf{h} = \varepsilon)$ and are done, or that $(\psi, h) \in [\![S_i]\!]$ and $\psi \in\ \uparrow \varphi_i$ hold, for some i. In the second case, by the induction hypothesis, we obtain $\psi, h \models \Phi_i$, and thus we derive $\psi, h \models \mathbf{B}_j\varphi_i \wedge \Phi_i$ which proves the case.

R5 Let us assume $\models S_i\ \mathit{sat}\ \Phi_i$ for all i. Consider a pair $(\psi, h) \in [\![\sum_i c_i?\varphi_i \cdot S_i]\!]$ for which we have to show $(\psi, h) \models \Phi$ with $\Phi \equiv \bigvee_i \exists t\ (\Phi_i[t/\mathsf{h}] \wedge \mathsf{h} = (c_i, \varphi_i) \cdot t)$. The definition of $[\![\ ]\!]$ tells us that there exists i and h' such that $h = (c_i, \varphi_i) \cdot h'$ and $(\psi, h') \in [\![S_i]\!]$, for which by the induction hypothesis we have $\psi, h' \models \Phi_i$. But then also $\psi, h \models \exists t\ (\Phi_i[t/\mathsf{h}] \wedge \mathsf{h} = (c_i, \varphi_i) \cdot t)$. We conclude $\psi, h \models \Phi$.

R6 The case that the last step in the derivation is an application of the rule **R6** is shown as follows. Assume $\models S_1\ \mathit{sat}\ \Phi_1$ and $\models S_2\ \mathit{sat}\ \Phi_2$. Consider a pair $(\psi, h) \in [\![S_1 \parallel S_2]\!]$ for which we have to show $\psi, h \models \Phi$, where Φ is given by $\exists t_1 t_2\ (\Phi_1[t_1/\mathsf{h}] \wedge \Phi_2[t_2/\mathsf{h}] \wedge \mathsf{h} \in (t_1 \,\&\, t_2))$. From the definition of $[\![\ ]\!]$ we derive h_1 and h_2 such that $(\psi, h_1) \in [\![S_1]\!]$, $(\psi, h_2) \in [\![S_2]\!]$ such that h is an interleaving of h_1 and h_2. By applying the induction hypothesis twice, we obtain $\psi, h_1 \models \Phi_1$ and $\psi, h_2 \models \Phi_2$. But then it also follows that $\psi, h \models \Phi$.

R7 We assume $\models S\ \mathit{sat}\ \Phi$ and consider a pair $(\psi, h) \in [\![\mathtt{loc}_x S]\!]$, for which we have to show $\psi, h \models \exists x \Phi$. By the definition of $[\![\ ]\!]$ we obtain that there exist ψ' and h' with $\exists_x\psi = \exists_x\psi'$, $\exists_x h = \exists_x h'$ and $(\psi', h') \in [\![S]\!]$. The induction hypothesis then yields $\psi', h' \models \Phi$. From the definition of $\models$ we derive $\psi, h \models \exists x \Phi$.

R8 Assume $\models P(x)\ \mathit{sat}\ \Phi$ holds. Consider a pair $(\psi, h) \in [\![P(y)]\!]$, for which we have to show $\psi, h \models \Phi[y/x]$. The definition of $[\![\ ]\!]$ yields that there exist ψ' and h' with $\psi = \psi'[y/x]$, $h = h'[y/x]$ and $(\psi', h') \in [\![P(x)]\!]$. The induction hypothesis then derives $\psi', h' \models \Phi$. The definition of $\models$ subsequently yields $\psi, h \models \Phi[y/x]$.

R9 The proof of an application of rule **R9**, which is a standard rule called *Scott's induction rule*, is similar to the one given in [7].

R10 Assume $\models S\ \mathit{sat}\ \Phi$ and $\models \Phi \rightarrow \Psi$. Consider a pair $(\varphi, h) \in [\![S]\!]$ for which by the induction hypothesis we have $\varphi, h \models \Phi$. But by $\models \Phi \rightarrow \Psi$ we immediately obtain $\varphi, h \models \Psi$. The soundness of the rule **R12** can be shown similarly.

R11 Let us assume $\models S_i\ \mathit{sat}\ \Phi_i$, for all $i \in [1, n]$. Consider a pair $(\sigma, h) \in [\![S_1 \parallel \cdots \parallel S_n]\!]$ for which we have to show $\sigma, h \models \Phi$, where Φ is given by $\bigwedge_i \Phi_i[(\mathsf{h} \restriction \mathit{port}_i)/\mathsf{h}]$. From the definition of $[\![\ ]\!]$, we derive $(\sigma(i), h \restriction \mathit{port}_i) \in [\![S_i]\!]$, for all $i \in [1, n]$. By n applications of the induction hypothesis, we obtain $\sigma(i), h \restriction \mathit{port}_i \models \Phi_i$, for all i. We conclude that $\sigma, h \models \Phi$ holds. □

Completeness of the proof system is formulated as follows: $\models S\ sat\ \Phi$ implies $\vdash S\ sat\ \Phi$ and $\models A\ sat\ \Phi$ implies $\vdash A\ sat\ \Phi$, which say that every correctness formula that is valid can also be derived from the proof system. In Boer et al. [7] the calculus for CCP is proved to be complete for CCP programs that are confluent. Since the proof system for ACPL integrates the proof systems of CCP and CSP, we inherit this completeness result. In other words, we conjecture (but will not prove) that our verification calculus is complete for multiagent systems in which the individual agents' internal reasoning processes are confluent.

Let us end the section with an example.

Example 6.6 As an illustration of the calculus we consider the specification:

$$\mathit{TwoAgents}\ sat\ \mathbf{B}_1 p \wedge \mathbf{B}_2 p,$$

which expresses that upon termination of the system *TwoAgents* from Example 3.3, the proposition p has become shared information among the agents *Agent1* and *Agent2*. We will consider here only the main application of the parallel composition rule. In fact, in most practical cases, the local verification that an agent satisfies a local specification is straightforward though tedious (and can be done almost automatically). The verification that a given specification of the multiagent system follows from this local specifications is in general the most interesting part of the proof.

Consider the local agent specifications, which are of the form:

$$\mathit{Agent1}\ sat\ \Omega \qquad \text{and} \qquad \mathit{Agent2}\ sat\ \Psi,$$

where the local assertions Ω and Ψ are depicted in Figure 7 and Figure 8, respectively. These local assertions formalise the different scenarios encoded in the agent programs.

$$\begin{aligned}
\Omega &= \textstyle\bigvee_{1\le i\le 4} \Omega_i \\
\Omega_1 &= \mathbf{B}_1 p \wedge \mathsf{h} = (c, \mathtt{ask}(p)) \cdot (d, p) \\
\Omega_2 &= \mathbf{B}_1(p \wedge q_1 \wedge q_2) \wedge \\
&\quad\ \mathsf{h} = (c, \mathtt{ask}(p)) \cdot (d, \mathtt{ask}(q_1 \wedge q_2)) \cdot \\
&\quad\ (c, q_1 \wedge q_2) \cdot (d, p) \\
\Omega_3 &= \neg\mathbf{B}_1((r_1 \vee r_2) \to (q_1 \wedge q_2)) \wedge \\
&\quad\ \mathsf{h} = (c, \mathtt{ask}(p)) \cdot (d, \mathtt{ask}(q_1 \wedge q_2))) \\
\Omega_4 &= \mathbf{B}_1(p \wedge q_1 \wedge q_2 \wedge (r_1 \vee r_2)) \wedge \\
&\quad\ \mathsf{h} = (c, \mathtt{ask}(p)) \cdot (d, \mathtt{ask}(q_1 \wedge q_2)) \cdot \\
&\quad\ (c, \mathtt{ask}(r_1 \vee r_2)) \cdot (d, r_1 \vee r_2) \cdot \\
&\quad\ (c, q_1 \wedge q_2) \cdot (d, p)
\end{aligned}$$

Figure 7. Local assertion Ω expressing the behaviour of *Agent1*.

$$\begin{aligned}
\Psi &= \textstyle\bigvee_{1\leq i\leq 5}\Psi_i\\
\Psi_1 &= \mathbf{B}_2 p \wedge \mathsf{h} = (c, \mathtt{ask}(p))\cdot(d,p)\\
\Psi_2 &= \mathbf{B}_2(p\wedge q_1)\wedge\\
&\quad \mathsf{h} = (c,\mathtt{ask}(p))\cdot(d,\mathtt{ask}(q_1))\cdot\\
&\quad (c,q_1)\cdot(d,p)\\
\Psi_3 &= \neg\mathbf{B}_2(q_1\to p)\wedge \mathsf{h} = (c,\mathtt{ask}(p))\\
\Psi_4 &= \mathbf{B}_2(q_1\to p)\wedge\neg\mathbf{B}_2 r_1\wedge\\
&\quad \mathsf{h} = (c,\mathtt{ask}(p))\cdot(d,\mathtt{ask}(q_1))\cdot\\
&\quad (c,\mathtt{ask}(r_1))\\
\Psi_5 &= \mathbf{B}_2(p\wedge q_1\wedge r_1)\wedge\\
&\quad \mathsf{h} = (c,\mathtt{ask}(p))\cdot(d,\mathtt{ask}(q_1))\cdot\\
&\quad (c,\mathtt{ask}(r_1))\cdot(d,r_1,)\cdot\\
&\quad (c,q_1)\cdot(d,p)
\end{aligned}$$

Figure 8. Local assertion Ψ expressing the behaviour of *Agent2*.

By using rule **R11**, we obtain the following global specification of the multiagent system:

TwoAgents sat Φ,

where the global assertion Φ is depicted in Figure 9. Note the correspondence of this specification with the observable behaviour of the system as depicted in Figure 6. Note that the assertions Ω_3, Ψ_3 and Ψ_4 describe scenarios in which either *Agent1* or *Agent2* prematurely terminates, while the other agent is still waiting for communication. These scenarios are not contained in the assertion Φ from the specification *TwoAgents sat* Φ of the overall system, since there is no global trace that is consistent with these local traces. This corresponds with the fact that these scenarios, in which an agent waits for communication with another agent that has terminated, are not part of the observable behaviour of the agent system.

$$\begin{aligned}
\Phi &= \textstyle\bigvee_{1\leq i\leq 3}\Phi_i\\
\Phi_1 &= \mathbf{B}_1 p\wedge\mathbf{B}_2 p\wedge \mathsf{h} = (c,\mathtt{ask}(p),\mathtt{ask}(p))\cdot(d,p,p)\\
\Phi_2 &= \mathbf{B}_1(p\wedge q_1\wedge q_2)\wedge\mathbf{B}_2(p\wedge q_1)\wedge\\
&\quad \mathsf{h} = (c,\mathtt{ask}(p),\mathtt{ask}(p))\cdot(d,\mathtt{ask}(q_1),\mathtt{ask}(q_1\wedge q_2))\cdot\\
&\quad (c,q_1\wedge q_2,q_1)\cdot(d,p,p)\\
\Phi_3 &= \mathbf{B}_1(p\wedge q_1\wedge q_2\wedge(r_1\vee r_2))\wedge\mathbf{B}_2(p\wedge q_1\wedge r_1)\wedge\\
&\quad \mathsf{h} = (c,\mathtt{ask}(p),\mathtt{ask}(p))\cdot(d,\mathtt{ask}(q_1),\mathtt{ask}(q_1\wedge q_2))\cdot\\
&\quad (c,\mathtt{ask}(r_1\vee r_2),\mathtt{ask}(r_1))\cdot(d,r_1,r_1\vee r_2)\cdot\\
&\quad (c,q_1\wedge q_2,q_1)\cdot(d,p,p)
\end{aligned}$$

Figure 9. Global assertion Φ expressing the behaviour of *TwoAgents*.

Finally, the global specification *TwoAgents sat* $\mathbf{B}_1 p \wedge \mathbf{B}_2 p$ follows via rule **R12** by means of some straightforward logical deduction.

7. Conclusions and future research

In this paper, we have presented a formal framework for agent communication that covers the issues of semantics, specification and verification. The first component of this framework is given by an *operational semantics* that defines the behaviour of the basic multiagent programming ACPL. Next, we identified the part of the behaviour of agents and multiagent systems that is visible to an *external observer*. This observable behaviour is subsequently modelled in a compositional way by means of a *denotational semantics*. On top of this semantic framework, we defined an *assertion language* that is designed for *specifications* of agent communication. The final and very essential component of the framework is a calculus that is suited to the *verification* of specifications.

In the related approaches [21, 48], the distinction between the different components of agent communication is less obvious; in particular, in these approaches, there is no clear distinction between the *semantics* of agent communication and the *specification* of desired agent behaviour. For instance, in Wooldridge p. [48, 19] the semantics of agent communication is viewed as a specification that must be satisfied. In our framework, however, the aspects of semantics and specification play different roles; that is, the semantics constitutes a *means* in the process of verifying that agents and multiagent systems act in accordance with their specified behaviour. In particular, in this paper we have presented a proof system for the verification of specifications that is built on top of the semantic framework.

An important application of our framework lies in the (semi-)automatic verification of agent communication. In future research, we aim to develop computer-aided verification techniques for agent communication (by means of a theorem proving environment such as PVS), which will be based on the verification calculus as presented in this paper.

Another issue of future research is a definition in our assertion language of the semantics of KQML. A corresponding application of our framework then consists of the verification that a multiagent system satisfies the assertion that characterises the KQML semantics. Additionally, we would like to examine speech acts from argumentation systems [27, 39, 44].

Other issues for future research include the refinement of the proof system with a treatment of the *deadlock behaviour* of multiagent systems. In this extension, we would need to introduce *failures* [4], which are communication actions that, when executed by other agents, cause an agent to enter in a deadlock situation.

Acknowledgments

The authors would like to thank the anonymous referees for their valuable comments.

Notes

1. Note that if $q_1 \wedge q_2$ follows from the belief state then both queries of the non-deterministic choice succeed. Which of them is actually executed depends on the underlying scheduler, but we will not consider this issue, here. Additionally, note that if none of these two queries succeeds, the agent deadlocks.
2. Also we do not consider conventions like the *distributed termination convention* of CSP [1], which says that a system can also terminate although one of its processes still waits for communication with a process that has already terminated.

References

1. K. Apt and N. Francez, "Modeling the distributed termination convention of CSP," *ACM Transactions on Programming Languages and Systems*, vol. 6, no. 3, pp. 370–379, 1984.
2. K. Apt, N. Francez, and W. d. Roever, "A proof system for communicating sequential processes," *ACM Transactions on Programming Languages and Systems*, vol. 2, no. 3, pp. 359–385, 1980.
3. J. Bergstra and J. Klop, "Process algebra for synchronous communication," *Information and Control*, vol. 60, pp. 109–137, 1984.
4. J. Bergstra, J. Klop, and E.-R. Olderog, "Readies and failures in the algebra of communicating processes," *SIAM Journal on Computing*, vol. 17, pp. 1134–1177, 1988.
5. R. Beun, "On the generation of coherent dialogue: A computational approach," *Pragmatics and Cognition*, vol. 9, no. 1, pp. 37–68, 2001.
6. F. d. Boer, R. v. Eijk, W. v. d. Hoek, and J.-J. Meyer, "Failure semantics for the exchange of information in multi-agent systems," in C. Palamidessi (ed.), *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000)*, vol. 1877 of Lecture Notes in Computer Science, Springer-Verlag: Heidelberg, 2000, pp. 214–228.
7. F. d. Boer, M. Gabbrielli, E. Marchiori, and C. Palamidessi, "Proving concurrent constraint programs correct," *ACM Transactions on Programming Languages and Systems*, vol. 19, no. 5, pp. 685–725, 1997.
8. F. d. Boer, A. D. Pierro, and C. Palamidessi, "Infinite computations in nondeterministic constraint programming," *Theoretical Computer Science*, vol. 151, pp. 37–78, 1995.
9. F. Brazier, B. Dunin-Keplicz, N. Jennings, and J. Treur, "Formal specification of multi-agent systems: a real-world case," in *Proceedings of International Conference on Multi-Agent Systems (ICMAS'95)*, MIT Press, 1995, pp. 25–32.
10. P. Bretier and D. Sadek, "A rational agent as the kernel of a cooperative spoken dialogue system: Implementing a logical theory of interaction," in J. Müller, M. Wooldridge, and N. Jennings (eds.), *Intelligent Agents III—Agent Theories, Architectures, and Languages (ATAL'96)*, vol. 1193 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 1997, pp. 189–203.
11. P. Cohen and H. Levesque, "Communicative actions for artificial agents," in *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, MIT Press, 1995, pp. 65–72,
12. E. Dijkstra, *A Discipline of Programming*, Prentice-Hall: Englewood Cliffs, NJ, 1976.
13. R. v. Eijk, F. d. Boer, W. v. d. Hoek, and J.-J. Meyer, "Systems of communicating agents," in H. Prade (ed.), *Proceedings of the 13th Biennial European Conference on Artificial Intelligence (ECAI'98)*, John Wiley & Sons, Ltd: Chichester, UK, 1998, pp. 293–297.
14. R. v. Eijk, F. d. Boer, W. v. d. Hoek, and J.-J. Meyer, "Information-passing and belief revision in multi-agent systems," in J. P. M. Müller, M. P. Singh, and A. S. Rao (eds.), *Intelligent Agents V, Proceedings of 5th International Workshop on Agent Theories, Architectures, and Languages (ATAL'98)*, vol. 1555 of Lecture Notes in Artificial Intelligence, Springer-Verlag: Heidelberg, 1999, pp. 29–45.
15. R. v. Eijk, F. d. Boer, W. v. d. Hoek, and J.-J. Meyer, "Open multi-agent systems: agent communication and integration," in N. Jennings and Y. Lespèrance (eds.), *Intelligent Agents VI, Proceedings of 6th International Workshop on Agent Theories, Architectures, and Languages (ATAL'99)*, vol. 1757 of Lecture Notes in Artificial Intelligence, Springer-Verlag: Heidelberg, 2000, pp. 218–232.
16. R. v. Eijk, F. d. Boer, W. v. d. Hoek, and J.-J. Meyer, "On dynamically generated ontology translators in agent communication," *International Journal of Intelligent Systems*, vol. 16, no. 5, pp. 587–607, 2001.

17. T. Finin, D. McKay, R. Fritzson, and R. McEntire, "KQML: An information and knowledge exchange protocol," in K. Fuchi and T. Yokoi (eds.), *Knowledge Building and Knowledge Sharing*, Ohmsha and IOS Press, 1994.
18. R. W. Floyd, "Assigning meaning to programs," in J. T. Schwartz (ed.), *American Mathematics Society Symposia in Applied Mathematics*, 1967, vol. 19, pp. 19–31.
19. L. Gamut, *Logic, Language and Meaning Volume I*. Chicago: University of Chicago Press: Chicago, 1991.
20. P. Gärdenfors, *Knowledge in flux: Modelling the Dynamics of Epistemic States*, Cambridge: Bradford Books, MIT: Cambridge, MA, 1988.
21. F. Guerin and J. Pitt, "A semantic framework for specifying agent communication languages," in *Proceedings of Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, IEEE Computer Society: Los Alamitos, CA, 2000, pp. 395–396.
22. N. Gupta and D. Nau, "On the complexity of blocks-world planning," *Artificial Intelligence*, vol. 56, pp. 223–254, 1994.
23. K. Hindriks, F. d. Boer, W. v. d. Hoek, and J.-J. Meyer, "Agent programming in 3APL," *Autonomous Agents and Multi-Agent Systems*, vol. 2, pp. 357–401, 1999.
24. C. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, 1969.
25. C. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1978.
26. H. Jifeng, M. Josephs, and C. Hoare, "A theory of synchrony and asynchrony," in *Proc. of the IFIP Working Conference on Programming Concepts and Methods*, 1990, pp. 446–465.
27. S. Kraus, K. Sycara, and A. Evenchik, "Reaching agreements through argumentation: A logical model and implementation," *Artificial Intelligence*, vol. 104, no. 1–2, pp. 1–69, 1998.
28. Y. Labrou and T. Finin, "Semantics for an agent communication language," in M. Singh, A. Rao, and M. Wooldridge (eds.), *Proceedings of Fourth International Workshop on Agent Theories, Architectures and Languages (ATAL'97)*, vol. 1365 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 1998, pp. 209–214.
29. Y. Labrou, T. Finin, and Y. Peng, "Agent communication languages: The current landscape," *IEEE Intelligent Systems*, vol. 14, no. 2, pp. 45–52, 1999.
30. B. v. Linder, W. v. d. Hoek, and J.-J. Meyer, "Communicating rational agents," in *KI-94: Advances in AI*, vol. 861 of Lecture Notes in Computer Science, Springer-Verlag, 1994, pp. 202–213.
31. J. Loeckx and K. Sieber, *The Foundations of Program Verification*, Wiley-Teubner Series in Computer Science, John Wiley and Sons and B.G. Teubner: Stuttgart, 1984.
32. P. Maes, R. Guttman, and A. Moukas, "Agent that buy and sell," *Communications of the ACM*, vol. 42, no. 3, pp. 81–91, 1999.
33. B. Mayoh, *Constraint Programming*, Springer-Verlag: Berlin, 1994.
34. J. Misra and K. Chandy, "Proofs of networks of processes," *IEEE Transactions on Software Engineering*, vol. 7, no. 4, pp. 417–426, 1981.
35. P. Naur, "Proof of algorithms by general snapshots," *Nordisk tidskrift for informationsbehandling*, vol. 6, no. 4, pp. 310–316, 1966.
36. S. Owicki and D. Gries, "An axiomatic proof technique for parallel programs," *Acta Informatica*, vol. 6, pp. 319–340, 1976.
37. J. Pitt and A. Mamdani, "Some remarks on the semantics of FIPA's agent communication language," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 4, pp. 333–356, 1999.
38. G. Plotkin, "A structured approach to operational semantics," Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.
39. H. Prakken, "On dialogue systems with speech acts, arguments, and counterarguments," in M. Ojeda-Aciego, I. d. Guzman, G. Brewka, and L. Pereira (eds.), *Proceedings of the 7th European Workshop on Logics in Artificial Inteligence (JELIA 2000)*, vol. 1919 of Lecture Notes in Artificial Intelligence, Springer-Verlag: Heidelberg, 2000.
40. A. Rao, "AgentSpeak(L): BDI agents speak out in a logical computable language," in W. v. d. Velde and J. Perram (eds.), *Agents Breaking Away*, vol. 1038 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 1996, pp. 42–55.
41. V. Saraswat, *Concurrent Constraint Programming*, The MIT Press: Cambridge, MA, 1993.

42. V. Saraswat, M. Rinard, and P. Panangaden, "Semantic foundations of concurrent constraint programming," in *Proceedings of the 18th ACM Symposium on Principles of Programming Languages (POPL'91)*, 1991, pp. 333–352.
43. Y. Shoham, "Agent-oriented programming," *Artificial Intelligence*, vol. 60, pp. 51–92, 1993.
44. C. Sierra, N. Jennings, P. Noriega, and S. Parsons, "A framework for argumentation-based negotiation," in M. Singh, A. Rao, and M. Wooldridge (eds.), *Intelligent Agents IV—Agent Theories, Architectures, and Languages (ATAL'97)*, vol. 1365 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 1998, pp. 177–192.
45. M. Singh, "Agent communication languages: Rethinking the principles," *IEEE Computer*, vol. 31, no. 12, pp. 40–47, 1998.
46. S. Thomas, "PLACA, an agent oriented programming language," Ph.D. thesis, Computer Science Department, Stanford University, Stanford, CA, 1993.
47. A. Turing, "Checking a large routine," in *Report on a Conference on High-Speed Automatic Calculating Machines*, University Mathematical Laboratory: Cambridge, 1949, pp. 67–69.
48. M. Wooldridge, "Semantic issues in the verification of agent communication," *Autonomous Agents and Multi-Agent Systems*, vol. 3, no. 1, pp. 9–31, 2000.
49. J. Zwiers, *Compositionality, Concurrency and Partial Correctness*, vol. 321 of Lecture Notes in Computer Science, Springer-Verlag, 1989.
50. J. Zwiers, A. d. Bruin, and W.-P. d. Roever, "A proof system for partial correctness of Dynamic Networks of Processes," in *Proceedings of the Conference on Logics of Programs*, vol. 164 of Lecture Notes in Computer Science, Springer-Verlag, 1983, pp. 513–527.