ELSEVIER Theoretical Computer Science 290 (2003) 1753–1773

Theoretical Computer Science

www.elsevier.com/locate/tcs

A fully abstract model for the exchange of information in multi-agent systems

Frank S. de Boer*, Rogier M. van Eijk, Wiebe van der Hoek, John-Jules Ch. Meyer

Department of Computer Science, Institute of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, Netherlands

Received 8 November 2000; received in revised form 6 August 2001; accepted 29 October 2001
Communicate by U. Montanari

Abstract

In this paper,[1] we present a semantic theory for the exchange of information in multi-agent systems. We consider the multi-agent programming language agent communication programming language, which integrates the paradigms of concurrent constraint programming and communicating sequential processes (CSP). The constraint programming techniques are used to represent and process information, whereas the synchronous communication mechanism from CSP is generalised to enable the exchange of information. The semantics of the language, which is based on a generalisation of traditional failure semantics, is shown to be fully abstract with respect to observing of each terminating computation its final global store of information.

Keywords: Exchange of information; Agent communication languages; Failure semantics; Full abstractness; Constraint programming

1. Introduction

Multi-agent systems are the subject of a very active and rapidly growing research field in both artificial intelligence and computer science. Although there is no formal definition of an *agent* (in fact this also holds for the notion of an *object*, which nevertheless has proven to be a very successful concept for the design of a new

[1] This paper is an extended version of [7].

* Corresponding author.

E-mail addresses: frankb@cs.uu.nl (F.S. de Boer), rogier@cs.uu.nl (R.M. van Eijk), wiebe@cs.uu.nl (W. van der Hoek), jj@cs.uu.nl (J.-J. Ch. Meyer).

0304-3975/02/$ - see front matter
PII: S0304-3975(02)00078-6

generation of programming languages), generally speaking, one could say that a *multi-agent system* constitutes a system composed of several autonomous agents that operate in a (distributed) environment which they can perceive, reason about as well as can affect by performing actions [24]. In the current research on multi-agent systems, a major topic is the development of a standardised *agent communication language* for the exchange of information. Recently, several agent communication languages have been proposed in the literature, like for instance the languages KQML [8] and FIPA-ACL [12]. However none of these communication languages have been given a fully formal account of their semantics [23]. The main contribution of this paper is the introduction of a formal semantic theory for the exchange of information in multi-agent systems.

1.1. Concurrent programming

We introduce the multi-agent programming language agent communication programming language (ACPL), which models the information processing aspects of agents. The underlying computational model of the language has already been introduced in [19–22]. The basic operations of the language for the processing of information are the *ask* and *tell* operations of concurrent constraint programming (CCP) [15,16]. This programming paradigm derives from traditional programming by replacing the *store-as-valuation* concept of von Neumann computing by the *store-as-constraint* model. This computational model is based on a global *store*, represented by a constraint, that expresses partial information on the values of the variables that are involved in computations. The different concurrently operating processes in CCP refine this partial information by adding (*telling*) new constraints to the store. Additionally, communication and synchronisation are achieved by allowing processes to test (*ask*) if the store entails a particular constraint before they proceed in their computation. These basic operations of asking and telling are defined in terms of the logical notions of conjunction and entailment, which are supported by a given underlying constraint system.

In the language ACPL, however, the global store of CCP is *distributed* among the agents of the system. That is, the above described ask and tell operations of CCP are used by an agent to maintain its own *private* store of information. More precisely, these operations are performed by concurrently executing *threads* within the agent. The agent itself, however, has no direct access to the parts of the global store that are distributed among the other agents in the system. Instead, the agents can only obtain information from each other by means of a synchronous communication mechanism.

This communication mechanism is based on a generalisation of the communication scheme of (imperative) concurrent languages like communicating sequential processes (CSP) [11], where the generalisation consists of the exchange of *information*, i.e. constraints, instead of the communication of simple *values*. Abstractly, communication between two agents comprises the supply of an *answer* of one agent to a posed *question* of another agent, and as such presents the basics of a *dialogue*. In particular, posing a question amounts to asking the other agent whether some information holds, while the answering agent in turn provides information from its own private constraint store that is logically strong enough to entail the question. In general, our programming language thus can be viewed upon as a particular model of the concept of *distributed*

knowledge as introduced in [9]. The above described communication mechanism then provides a way in which the distributed knowledge of a multi-agent system can become shared among the agents.

1.2. Fully abstract semantics

The main result of this paper is a compositional semantics for the multi-agent language ACPL that is *fully abstract* with respect to observing the final (global) stores of terminating computations. This semantics is based on a generalisation of the *failure semantics* as developed for CSP, in which failure sets are employed to give a semantic account of (possible) *deadlock* behaviour [5]. However, whereas in CSP a failure set is simply given by a subset of the complement of all the *initial* actions of a process, in our framework, these failure sets are defined in terms of the information that is *logically* irrelevant to the specific question or answer of the agent. Moreover, for CSP-like languages the failure sets can without loss of generality be assumed to be *finite* [18]. This assumption, which plays an essential role in the full abstractness proof, fails in the context of the exchange of information. However, we show that our notion of failure semantics, which includes *infinite* failure sets, satisfies a *compactness property* that roughly amounts to the following: if every *finite* subset of a given set of answers or questions is logically irrelevant (with respect to a particular question/answer of a given agent) then this entire set is irrelevant. Additionally, the logical nature of the communication mechanism requires an abstraction of what is actually communicated. This abstraction corresponds to the principle of 'asking more and telling less' that forms the basis of the fully abstract model of CCP [6,16].

1.3. Related work

To the best of our knowledge, this paper presents a first formal semantic account of the exchange of information in multi-agent systems. This semantics, we believe, provides a general basis for the semantics of agent communication languages in general as introduced in artificial intelligence, like for instance KQML [8].

Other approaches that relate to our programming language include the work of Réty on distributed concurrent constraint programming [14]. One of the differences with our approach is that in the framework of Réty, distributed processes do not share any variables. In particular, communication between processes proceeds by means of a form of constraint abstraction: during the exchange of a constraint, the variables of the sender that occur in the constraint are replaced by the variables of the receiving processes.

Additionally, there is the research on synchronous concurrent constraint programming, which is a version of CCP that in addition to the standard ask and tell operations, covers a synchronous communication mechanism in which a constraint is told to the constraint store only if there is another process asking for it [4]. The main difference with our approach is that in this framework both the synchronous and asynchronous form of communication proceed via a global constraint store.

1.4. Overview

The remainder of this paper is organised as follows. In Section 2, we give the syntax of the multi-agent programming language ACPL, which combines the language of concurrent constraint programming with synchronous communication primitives for the exchange of information. The structural operational semantics of this language is subsequently defined in Section 3 in terms of a local and global transition system. Additionally, in Section 4, we define a failure semantics which is shown to be fully abstract with respect to observing the final information store of terminating computations. Finally, in Section 5 we round off by suggesting several directions for future research.

2. Syntax

In this section, we introduce the syntax of our agent language ACPL, which like CCP is parameterised by a constraint system that is used to represent information.

Definition 1 (Constraint systems). A *constraint system* $\mathbf{C}$ is a tuple $(C, \sqsubseteq, \sqcup, \mathit{true}, \mathit{false})$, where C (the set of constraints, with typical element φ) is a set ordered with respect to $\sqsubseteq$, $\sqcup$ is the least upperbound operation, and *true*, *false* are the least and greatest elements of C, respectively.

The interpretation of $\varphi \sqsubseteq \psi$ is that φ contains less information than ψ, while $\varphi \sqcup \psi$ denotes the conjunction of φ and ψ.

In order to model *hiding* of local variables and *parameter passing* in constraint programming, in [16] the notion of constraint system is enriched with *cylindrification operators* and *diagonal elements*, which are concepts borrowed from the theory of cylindric algebras [10].

Definition 2 (Cylindric constraint systems). Given a (denumerable) set of variables *Var* with typical elements $x, y, z, \ldots$, we introduce a family of operators $\{\exists_x \mid x \in \mathit{Var}\}$ (cylindrification operators) and of constants $\{d_{xy} \mid x, y \in \mathit{Var}\}$ (diagonal elements).

Starting from a constraint system $\mathbf{C}$, we define a *cylindric constraint system* $\mathbf{C}'$ as the constraint system whose support set C' is the smallest such that

$$C' = C \cup \{\exists_x \varphi \mid x \in \mathit{Var}, \varphi \in C'\} \cup \{d_{xy} \mid x, y \in \mathit{Var}\}$$

modulo the identities and with the additional relations derived by the following axioms, where $\exists_x \varphi \sqcup \psi$ stands for $(\exists_x \varphi) \sqcup \psi$:

A1. $\exists_x \varphi \sqsubseteq \varphi$,
A2. if $\varphi \sqsubseteq \psi$ then $\exists_x \varphi \sqsubseteq \exists_x \psi$,
A3. $\exists_x(\varphi \sqcup \exists_x \psi) = \exists_x \varphi \sqcup \exists_x \psi$,
A4. $\exists_x \exists_y \varphi = \exists_y \exists_x \varphi$,
A5. $d_{xx} = \mathrm{true}$,

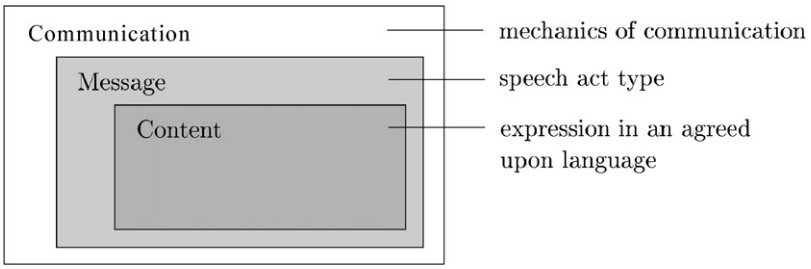

Fig. 1. Layers of a KQML expression.

A6. if $z \neq x, y$ then $d_{xy} = \exists_z(d_{xz} \sqcup d_{zy})$,
A7. if $x \neq y$ then $\varphi \sqsubseteq d_{xy} \sqcup \exists_x(\varphi \sqcup d_{xy})$.

The above laws give to $\exists_x$ the flavour of a first-order *existential quantifier*, as the notation also suggests. The constraint d_{xy} can be interpreted as the equality between x and y. Cylindrification and diagonal elements allow us to model the variable renaming of a formula φ; in fact, by the above axioms, the formula $\exists_x(d_{xy} \sqcup \varphi)$ can be interpreted as the formula $\varphi[y/x]$, namely the formula obtained from φ by replacing all the free occurrences of x by y. We also assume the generalisation $\varphi[\vec{y}/\vec{x}]$ to sequences of variables.

Agent communication languages, like KQML [8], can be thought of being divided in different layers (see Fig. 1). 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 [1,17] 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`, which allows to derive negative information in terms of the closed world assumption [13], is represented by the expression $\texttt{untell}(\varphi)$. These operators can be defined by an extension of the information ordering of the constraint system. For instance, given the constraints ψ and φ, we define

$$\varphi \not\sqsubseteq \psi \quad \Leftrightarrow \quad \texttt{untell}(\varphi) \sqsubseteq \psi.$$

Assuming that ψ represents the belief base of an agent, this rule formalises the closed world assumption. The anti-monotonicity property of the `untell` operator is expressed by

$$\texttt{untell}(\varphi) \sqsubseteq \texttt{untell}(\psi) \quad \Leftrightarrow \quad \psi \sqsubseteq \varphi.$$

In this paper, we will assume that the content layer and message layer together form a constraint system. This assumption is justified because the semantics of the communication mechanism itself can be fully described in terms of the underlying information ordering.

The communication mechanism is described in KQML by its communication layer and involves basic concepts like sender, recipient, communication channel and synchronicity.

The main objective of the programming language defined below is to provide a generic framework for the exchange of information in multi-agent systems, which abstracts from the specific nature of the underlying information system.

In the following definition, we assume a given set *Chan* of communication channels, with typical element c.

Definition 3 (Basic actions). Given a cylindric constraint system $\mathbf{C}$ the *basic actions* of the programming language are defined as follows:

$$a ::= c!\varphi \mid c?\varphi \mid \mathtt{ask}(\varphi) \mid \mathtt{tell}(\varphi).$$

The execution of the output action $c!\varphi$ consists of sending the information φ along the channel c, which has to synchronise with a corresponding input $c?\psi$, for some ψ with $\psi \sqsubseteq \varphi$. In other words, the information φ can be sent along a channel c only if some information entailed by φ is requested. The execution of an input action $c?\psi$, which consists of receiving the information φ along the channel c, also has to synchronise with a corresponding output $c!\varphi$, for some φ with $\psi \sqsubseteq \varphi$. The execution of a basic action $\mathtt{ask}(\varphi)$ by an agent consists of checking whether the private store of the agent entails φ. On the other hand, the execution of $\mathtt{tell}(\varphi)$ consist of adding φ to the private store.

In the following definition, we assume a given set *Proc* of procedure identifiers, with typical element p.

Definition 4 (Statements). The behaviour of an agent is then described by a *statement* S:

$$S ::= a \cdot S \mid S_1 + S_2 \mid S_1 \ \& \ S_2 \mid \exists_x S \mid p(\bar{x}).$$

Statements are thus built up from the basic actions using the following standard programming constructs: action prefixing, which is denoted by $\cdot$; non-deterministic choice, denoted by $+$; internal parallelism, denoted by &; local variables, denoted by $\exists_x S$, which indicates that x is a local variable in S; and (recursive) procedure calls of the form $p(\bar{x})$, where $p \in Proc$ constitutes the name of the procedure and $\bar{x}$ denotes a sequence of variables which constitute the actual parameters of the call. Since, no information on a local variable x can be communicated we additionally require that in $\exists_x S$ the variable x does not occur free in a communication of S; that is, $\exists_x \varphi = \varphi$ for every communication action $c?\varphi$ or $c!\varphi$ of S.

Definition 5 (Multi-agent systems). A *multi-agent system* A is defined as follows:

$$A ::= \langle D, S, \varphi \rangle \mid A_1 \parallel A_2 \mid \delta_H(A).$$

A basic agent in a multi-agent system is represented by a tuple $\langle D, S, \varphi \rangle$. The set D consists of procedure declarations of the form $p(\bar{x}) :- S$, where $\bar{x}$ denote the formal

parameters of p and S denotes its body. We assume that D satisfies the following property:

$$\text{if } p(\bar{x}) : -S \in D \text{ then } p(\bar{y}) : -S[\bar{y}/\bar{x}] \in D$$

for all $\bar{x}$ and $\bar{y}$, where $S[\bar{y}/\bar{x}]$ denotes the statement S in which each constraint φ is replaced by $\varphi[\bar{y}/\bar{x}]$. The statement S in $\langle D, S, \varphi\rangle$ describes the behaviour of the agent with respect to its *private* store φ. The threads of S, i.e. the concurrently executing substatements of S, interact with each other via the private store of the basic agent by means of the actions $\mathtt{ask}(\psi)$ and $\mathtt{tell}(\psi)$. As in the operational semantics below the set D of procedure declarations will not change, we usually omit it from notation and simply write $\langle S, \varphi\rangle$ instead of $\langle D, S, \varphi\rangle$.

Additionally, a multi-agent system itself consists of a collection of concurrently operating agents that interact with each other only via a synchronous information-passing mechanism by means of the communication actions $c!\psi$ and $c?\psi$.

For technical convenience only we restrict to the parallel composition of agent systems: the semantic treatment of the sequential composition of multi-agent systems and the non-deterministic choice between agent systems is standard. Moreover, due to our focus on the semantic treatment of the communication mechanism we do not consider recursion at the level of multi-agent systems.

Finally, the encapsulation operator δ_H with $H \subseteq Chan$, which stems from the process algebra ACP, is used to define local communication channels [2]. That is, $\delta_H(A)$ denotes a multi-agent system in which the communication channels in H are local and hence, cannot be used for communication with agents outside the system.

3. Operational semantics

The structural operational semantics of the programming language is defined by means of a *local* and a *global* transition system. Given a set of declarations D, a local transition is of the form

$$\langle S, \varphi\rangle \xrightarrow{l} \langle S', \psi\rangle$$

where either l equals τ in case of an *internal* computation step, that is, a computation step which consists of the execution of a basic action of the form $\mathtt{ask}(\varphi)$ or $\mathtt{tell}(\varphi)$, or l is of the form $c!\varphi$ or $c?\varphi$, in case of a communication step. We employ the symbol E to denote successful termination.

Definition 6 (Transitions for basic actions).

$$\langle c!\varphi, \psi\rangle \xrightarrow{c!\varphi} \langle E, \psi\rangle \qquad \text{if } \varphi \sqsubseteq \psi$$

$$\langle c?\varphi, \psi\rangle \xrightarrow{c?\varphi} \langle E, \psi \sqcup \varphi\rangle$$

$$\langle \mathtt{ask}(\varphi), \psi\rangle \xrightarrow{\tau} \langle E, \psi\rangle \qquad \text{if } \varphi \sqsubseteq \psi$$

$$\langle \mathtt{tell}(\varphi), \psi\rangle \xrightarrow{\tau} \langle E, \psi \sqcup \varphi\rangle$$

An output action $c!\varphi$ can only take place in case the information φ to be communicated is entailed by the private store ψ. In other words, the agents are assumed to be *truthful*. On the other hand, the information φ received by an input action $c?\varphi$ is added to the private store. It is worthwhile to remark here that alternatively we could have defined input actions semantically by $\langle c?\varphi,\psi\rangle \xrightarrow{c?\varphi} \langle E,\psi\rangle$. Whether or not the private store is correspondingly updated can be controlled by the agent itself by means of the action $\texttt{tell}(\varphi)$. However, for technical convenience only we have adopted in this paper the first approach.

The actions $\texttt{ask}(\varphi)$ and $\texttt{tell}(\varphi)$ are the familiar operations from CCP which allow an agent to inspect and update its private store.

Furthermore, we have the usual rules for action prefixing, procedure calls and the programming constructs for non-deterministic choice and parallel composition, which is modelled by interleaving.

Definition 7 (Transitions for statements).

$$\frac{\langle a,\psi\rangle \xrightarrow{l} \langle E,\psi'\rangle}{\langle a\cdot S,\psi\rangle \xrightarrow{l} \langle S,\psi'\rangle}$$

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

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

$$\frac{\langle S,\exists_x\psi\sqcup\varphi\rangle \xrightarrow{l} \langle S',\psi'\rangle}{\langle \exists_x^{\varphi} S,\psi\rangle \xrightarrow{l} \langle \exists_x^{\psi'} S',\psi\sqcup\exists_x\psi'\rangle}$$

$$\langle p(\bar{y}),\psi\rangle \xrightarrow{\tau} \langle S,\psi\rangle \quad \text{where } p(\bar{y}) :\!- S\in D$$

Note that the syntax of the language is extended with a construct of the form $\exists_x^{\varphi} S$ denoting that in the statement S the variable x is a local variable, where the constraint φ collects the information on the local variable x. In this notation, the statement $\exists_x S$ is written as $\exists_x^{\text{true}} S$, denoting that the local constraints on x are initially empty. Note that no information on the local variable x can be communicated, because by definition x does not occur free in φ in case l is of the form $c?\varphi$ or $c!\varphi$.

A *global* transition is of the form $A \xrightarrow{l} A'$, where l indicates whether the transition involves an internal computation step, that is, $l=\tau$, or a communication, that is, $l=c!\varphi$ or $l=c?\varphi$.

Definition 8 (Transitions for multi-agent systems). The following rule describes parallel composition by interleaving of the basic actions:

$$\frac{A_1 \xrightarrow{l} A_1'}{\begin{array}{l} A_1 \parallel A_2 \xrightarrow{l} A_1' \parallel A_2 \\ A_2 \parallel A_1 \xrightarrow{l} A_2 \parallel A_1' \end{array}}$$

In order to describe the synchronisation between agents we introduce a synchronisation predicate $|$, which is defined as follows. For all $c \in Chan$ and $\varphi, \psi \in \mathbf{C}$, if $\psi \sqsubseteq \varphi$ then

$$(c!\varphi \mid c?\psi) \quad \text{and} \quad (c?\psi \mid c!\varphi).$$

In all other cases, the predicate $|$ yields the boolean value false. We then have the following synchronisation rule:

$$\frac{A_1 \xrightarrow{l_1} A_1' \quad A_2 \xrightarrow{l_2} A_2'}{A_1 \parallel A_2 \xrightarrow{\tau} A_1' \parallel A_2'} \quad \text{if } l_1 \mid l_2$$

This rule shows that an action of the form $c?\psi$ only matches with an action of the form $c!\varphi$ in case ψ is entailed by φ. In all other cases, the predicate $|$ yields false and therefore no communication can take place.

Finally, *encapsulation* of communications along a set of channels H is described by the rule:

$$\frac{A \xrightarrow{l} A'}{\delta_H(A) \xrightarrow{l} \delta_H(A')} \quad \text{if } chan(l) \cap H = \emptyset$$

where *chan* is defined by $chan(c!\varphi) = chan(c?\varphi) = \{c\}$ and $chan(\tau) = \emptyset$.

For any multi-agent system A we use the notation $store(A)$ to denote its constraint store.

Definition 9 (Global store). We define $store(A)$ by induction on the structure of the agent system A:

$$\begin{aligned} store(\langle D, S, \varphi \rangle) &= \varphi \\ store(A_1 \| A_2) &= store(A_1) \sqcup store(A_2) \\ store(\delta_H(A)) &= store(A) \end{aligned}$$

For any multi-agent system A, $store(A)$ thus denotes the *global* constraint store that is distributed among its (sub-)agents. In fact, this amounts to what is known as *distributed knowledge* in the research on distributed systems, referring to the knowledge that would result if the knowledge of all agents in a distributed system is taken together [9].

We want to observe the behaviour of a multi-agent system when it runs on its own, that is, as a *closed* system without interaction with an environment. In the definition

below the *observable* behaviour of a multi-agent system is therefore defined as the set of final stores of *terminating* computations that consist of internal computation steps only. Moreover, the existence of an internally diverging computation or the generation of an inconsistent global store will be considered as a fatal *error* and as such they will give rise to *chaos*, which amounts to a situation in which simply anything can be observed.

In defining the operational semantics we make use of several auxiliary notions:

- The notation $A \Rightarrow B$ indicates the existence of a computation that consists of internal computation steps only: $A \xrightarrow{\tau} \cdots \xrightarrow{\tau} B$. Additionally, $B \not\xrightarrow{\tau}$ indicates that starting from B there is no further τ-transition possible; that is, B is either terminated or represents a *deadlock* situation.
- Furthermore, the construct $A \Rightarrow \Omega$ indicates the existence of an infinite computation that consists of internal computation steps only: $A \xrightarrow{\tau} \cdots \xrightarrow{\tau} A_i \xrightarrow{\tau} \cdots$. Such computations are called *divergent*.

The operational semantics is then defined as follows.

Definition 10 (Operational semantics).

$$\begin{aligned}\mathcal{O}(A) = {} & \{store(B) \mid A \Rightarrow B \not\xrightarrow{\tau}\} \cup \\ & \{\varphi \in \mathbf{C} \mid A \Rightarrow B, store(B) = false\} \cup \\ & \{\varphi \in \mathbf{C} \mid A \Rightarrow \Omega\}.\end{aligned}$$

Thus, the observable behaviour $\mathcal{O}(A)$ of an agent system A consists of the output stores that the system produces. However, in case the system produces an output store that is inconsistent or gives rise to an internally diverging computation, a fatal error has occurred. In these circumstances, the observable behaviour comprises all possible output stores.

Note that the treatment of (internally) diverging computations and inconsistent stores can be mathematically justified in terms of the following *recursive* definition of $\mathcal{O}$:

$$\begin{aligned}\mathcal{O}(A) = {} & \{store(A) \mid A \not\xrightarrow{\tau}\} \cup \\ & \{\varphi \in \mathbf{C} \mid store(A) = false\} \cup \\ & \{\varphi \in \mathcal{O}(B) \mid A \xrightarrow{\tau} B\}.\end{aligned}$$

The above non-recursive definition then can be shown to correspond to the *greatest fixpoint* of this recursive equation with respect to the pointwise extension of the subset ordering.

We observe that internal divergent computations may be *unfair*: if A gives rise to an internally divergent computation so will $A \parallel B$, for *any* B. However, the results of this paper can be easily extended to a semantics of the parallel composition operator which is *weakly* fair in the following sense: every parallel agent which is enabled will eventually be executed.

4. Failure semantics

In order to obtain a compositional and fully abstract characterisation of the above defined notion of observables, we introduce a refined semantics that besides containing the produced information store, also records the sequence of communication actions that the system executes as well as a *failure set* that contains the actions that, when offered by the environment, lead to a deadlock situation.

Definition 11 (Failure semantics).

- The transition relation $A \stackrel{w}{\Rightarrow} B$, where w is a sequence of communication actions, indicates that the agent system A can evolve itself into B by executing a sequence of actions w' such that w can be obtained from w' by deleting all τ-moves.
- We let I be the function that associates with each multi-agent system A the collection of initial actions it can perform, which is defined as follows:

$$I(A) = \{l \mid A \stackrel{l}{\rightarrow} B, \text{ for some } B\}.$$

- Given a set X of communication actions, the complement $\bar{X}$ is defined by

$$\bar{X} = \{c?\psi \mid \text{there does not exist } c!\varphi \in X \text{ with } \psi \sqsubseteq \varphi\} \cup \{c!\varphi \mid \text{there does not exist } c?\psi \in X \text{ with } \psi \sqsubseteq \varphi\}.$$

  The set $\bar{X}$ thus contains precisely the communication actions that cannot synchronise with a communication action in X.
- The failure semantics $\mathscr{F}$ is then defined as follows:

$$\mathscr{F}(A) = \{\langle w, (store(B), F)\rangle \mid A \stackrel{w}{\Rightarrow} B,\ \tau \notin I(B),\ F \subseteq \overline{I(B)}\} \cup \{\langle w \cdot w', \bot\rangle \mid A \stackrel{w}{\Rightarrow} B \text{ and } store(B) = false\} \cup \{\langle w \cdot w', \bot\rangle \mid A \stackrel{w}{\Rightarrow} B \text{ and } B \Rightarrow \Omega\}.$$

The above failure semantics associates with each multi-agent system A a set of *failure traces*, which record the sequence of communication actions that are generated by a computation of A, the corresponding resulting store and a failure set of communication actions that are *refused*. Divergence and inconsistency are represented by the symbol $\bot$, denoting a situation of chaos in which everything is possible.

The above definition of a failure set differs from the standard one, which is simply given by a subset of the *complement* of the set of initial actions [3,5]. The standard failure sets as such indicate the actions which the process itself cannot perform. However, the communication actions that an agent *cannot perform* are not necessarily given by the actions that it *refuses* for synchronisation.

Example 12 (Failure sets). Consider the following two agents A and B:

$$A = \langle c!(\varphi \sqcup \psi), \varphi \sqcup \psi\rangle$$
$$B = \langle (c!(\varphi \sqcup \psi) + c!\varphi), \varphi \sqcup \psi\rangle.$$

The set of initial actions of A and B are different:

$$I(A) = \{c!(\varphi \sqcup \psi)\}$$
$$I(B) = \{c!(\varphi \sqcup \psi),\ c!\varphi\}.$$

Hence, the actions that A and B cannot execute are also different: the agent A *cannot* perform the action $c!\varphi$, while this does not hold for the agent B. However, the actions that are *refused* for synchronisation by the agents are the same, that is, for their failure sets we have

$$\overline{I(A)} = \overline{I(B)}.$$

In fact, there is the following more general result, which states that $\bar{\bar{X}}$ contains all the communication actions that derive from X by *asking more* and *telling less*.

Lemma 13 (Refusals of refusals). *For any set X of communication actions, we have*

$$\bar{\bar{X}} = \{c?\psi \mid c?\varphi \in X \text{ and } \varphi \sqsubseteq \psi\} \cup \{c!\varphi \mid c!\psi \in X \text{ and } \varphi \sqsubseteq \psi\}.$$

Proof. We have the following sequence of equivalences:

$$\begin{array}{ll} c!\varphi \in \bar{\bar{X}} & \Leftrightarrow \\ \forall \psi\ (\psi \sqsubseteq \varphi \Rightarrow c?\psi \notin \bar{X}) & \Leftrightarrow \\ \forall \psi\ (\psi \sqsubseteq \varphi \Rightarrow \exists \varphi'\ (c!\varphi' \in X \text{ and } \psi \sqsubseteq \varphi')) & \Leftrightarrow \\ \exists \varphi'\ (c!\varphi' \in X \text{ and } \varphi \sqsubseteq \varphi') & \Leftrightarrow \\ c!\varphi \in \{c!\varphi \mid c!\varphi' \in X \text{ and } \varphi \sqsubseteq \varphi'\}. & \end{array}$$

To see that the third equivalence holds, for the implication from left to right take φ for ψ. Conversely, consider ψ with $\psi \sqsubseteq \varphi$. If we assume that there exists a constraint φ' with $\varphi \sqsubseteq \varphi'$ that satisfies $c!\varphi' \in X$, we can also conclude that there exists φ' with $\psi \sqsubseteq \varphi'$ such that $c!\varphi' \in X$.

The same line of reasoning can be applied to formulae $c?\psi \in \bar{\bar{X}}$. □

Of particular interest is to observe that failure sets can be *infinite*, but denumerable sets of communication actions; i.e. when the underlying constraint system contains infinitely but denumerable many constraints. This is in contrast with the traditional approaches in which the assumption is made that failure sets are finite [18].

The following theorem states the correctness of the above failure semantics.

Lemma 14 (Correctness of $\mathscr{F}$). *For all agent systems A and formula φ the following holds*:

$$\varphi \in \mathscr{O}(A) \text{ iff } \langle \varepsilon, (\varphi, F)\rangle \in \mathscr{F}(A) \text{ for some } F \text{ or } \langle \varepsilon, \bot\rangle \in \mathscr{F}(A),$$

where ε denotes the empty trace.

Proof. If $\varphi \in \mathcal{O}(A)$ then there are three possibilities: either there exists an agent system B with $A \Rightarrow B$ and $store(B) = \varphi$, or a fatal error has occurred, which means that $A \Rightarrow \Omega$ or for some B we have $A \Rightarrow B$ with $store(B) = false$. In the first situation, we have $\langle \varepsilon, (store(B), F)\rangle \in \mathscr{F}(A)$, for some particular failure set F. In the other situation, which amounts to chaos, we have $\langle \varepsilon, \perp\rangle \in \mathscr{F}(A)$. For the converse, the same line of reasoning can be applied. □

4.1. Compositionality of failure semantics

In order to establish the compositionality of the failure semantics, we first introduce the parallel composition of sequences of communication actions.

Definition 15 (Compositionality of communication traces). Given sequences w_1 and w_2 of communication actions, we define $w_1 \| w_2$ to be the following set of communication sequences [2]:

- $\varepsilon \parallel \varepsilon$ is equal to $\{\varepsilon\}$, and for the other cases:
- $w_1 \parallel w_2 = (w_1 \lfloor\!\lfloor w_2) \cup (w_2 \lfloor\!\lfloor w_1) \cup (w_1 \mid w_2)$, where the leftmerge operator $\lfloor\!\lfloor$ and the synchronisation merge $\mid$ are (recursively) defined by:
- $(a \cdot w) \lfloor\!\lfloor w' = a \cdot (w \parallel w')$ and
- $(c?\psi \cdot w) \mid (c!\varphi \cdot w') = (w \parallel w')$ provided that $\psi \sqsubseteq \varphi$. In all other cases we have: $w_1 \mid w_2 = \emptyset$.

Thus, $w_1 \parallel w_2$ consists of all possible interleavings of the traces w_1 and w_2 and additionally takes into account all possible synchronisations between them.

Definition 16 (The predicate $X \mid Y$). Given two sets X and Y of communication actions, we use the notation $X \mid Y$ to indicate that there exists $c!\varphi \in X$ and $c?\psi \in Y$ (or vice versa) such that $\psi \sqsubseteq \varphi$.

So if, X denotes a set of actions that an agent A can execute and Y a set of actions that an agent B can execute then $X \mid Y$ implies that A and B can communicate with each other. Finally, for the compositional modelling of chaos introduced by internally diverging computations and inconsistent constraint stores, we need the following notions.

Definition 17 (Infinite communication traces).

- The set $\mathscr{F}^{\omega}(A)$ denotes all the *infinite* sequences w of communication actions such that for every prefix w' of w, we have $\langle w', t\rangle \in \mathscr{F}(A)$, for some t.
- The predicate $v_1 \uparrow v_2$ is recursively defined: $(c!\varphi \cdot v) \uparrow (c?\psi \cdot v')$ if and only if $(\psi \sqsubseteq \varphi$ and $v \uparrow v')$.

The composition of termination modes is defined as follows.

Definition 18 (Termination modes).

- Provided that $\overline{F_1} \nmid \overline{F_2}$ and $\varphi_1 \sqcup \varphi_2 \neq$ false we define: $(\varphi_1, F_1) \parallel (\varphi_2, F_2) = \{(\varphi_1 \sqcup \varphi_2, F) \mid F \subseteq F_1 \cap F_2\}$

- Provided that $\varphi_1 \sqcup \varphi_2 = \text{false}$ we define:
  $(\varphi_1, F_1) \parallel (\varphi_2, F_2) = \{\bot\}$
- $(\varphi, F) \parallel \bot = \{\bot\}$ and $\bot \parallel (\varphi, F) = \{\bot\}$.

Note that the first two equations for the composition of termination modes are partially defined; in the other cases, the result is the empty set. So, if one agent system ends with a store φ_1 and failure set F_1 and another agent system with a store φ_2 and failure set F_2, then their parallel composition produces a store $\varphi_1 \sqcup \varphi_2$ and refuses a set F of actions that are refused by both of the agent systems, hence $F \subseteq F_1 \cap F_2$. Moreover, it is required that the systems cannot communicate with each other, that is, $\overline{F_1} \not\downarrow \overline{F_2}$. The second equation handles the case in which the composition of φ_1 and φ_2 yields an inconsistent store. This fatal error is represented by the symbol $\bot$. Finally, the third equation deals with the situation in which an error has occurred in one of the two agent systems; this error is propagated to the parallel composition.

Next, we consider the compositionality of the failure semantics.

Theorem 19 (Compositionality of $\mathscr{F}$). *The compositionality of the parallel operator is given by*

$$
\begin{array}{l}
\mathscr{F}(A_1 \parallel A_2) \\
= \\
\{\langle w, t\rangle \mid w \in (w_1 \parallel w_2),\ t \in (t_1 \parallel t_2),\ \langle w_1, t_1\rangle \in \mathscr{F}(A_1), \\
\langle w_2, t_2\rangle \in \mathscr{F}(A_2)\} \\
\cup \\
\{\langle w \cdot w', \bot\rangle \mid w \in (w_1 \parallel w_2),\ (w_1 \cdot u) \in \mathscr{F}^{\omega}(A_1),\ (w_2 \cdot v) \in \mathscr{F}^{\omega}(A_2), \\
u \uparrow v\}.
\end{array}
$$

The compositionality of the encapsulation operator is phrased as follows, where we use the notation chan(w) to denote the set of channels that occur in the sequence w:

$$
\begin{array}{l}
\mathscr{F}(\delta_H(A)) \\
= \\
\{\langle w, (\varphi, F)\rangle \mid \langle w, (\varphi, F')\rangle \in \mathscr{F}(A),\ chan(w) \cap H = \emptyset, \\
F \subseteq F' \cup \{c!\psi, c?\psi \mid c \in H, \psi \in \mathbf{C}\}\} \\
\cup \\
\{\langle w, \bot\rangle \mid \langle w, \bot\rangle \in \mathscr{F}(A),\ chan(w) \cap H\ \ = \emptyset\}.
\end{array}
$$

Divergence of the parallel composition $A_1 \parallel A_2$ thus stems from the divergence of one of the individual agent systems A_1 or A_2, or from the generation of an infinite sequence u of communication actions by A_1 and an infinite sequence v by A_2 such that each of the actions in u matches the corresponding action in v, in which case the proposition $u \uparrow v$ is true.

The proof of this theorem can be found in the appendix. In this proof as well as in the sequel we frequently make use of the following well-known result.

Lemma 20 (König's lemma). *Every finitely branching tree with an infinite number of nodes has an infinite branch.*

4.2. Full abstraction of failure semantics

The failure semantics $\mathscr{F}$ still distinguishes too many multi-agent systems, that is, it is not fully abstract with respect to the observables $\mathscr{O}$. This is shown in the following example.

Example 21. Consider again the agents

$$A = \langle c!(\varphi \sqcup \psi), \varphi \sqcup \psi \rangle$$
$$B = \langle (c!(\varphi \sqcup \psi) + c!\varphi), \varphi \sqcup \psi \rangle.$$

The failure semantics distinguishes these two agents since we have

$$\langle c!\varphi, (\varphi \sqcup \psi, F) \rangle \in \mathscr{F}(B) - \mathscr{F}(A),$$

for all F. However, intuitively, there is no context C such that $\mathscr{O}(C[A]) \neq \mathscr{O}(C[B])$. (Formally, this can be shown via the correctness and compositionality of the operator $\mathscr{F}_\alpha$ that is defined below.)

From this example we conclude that in order to obtain a fully abstract semantics we should account for the fact that asking for a constraint includes asking for all stronger information and that the communication of a constraint includes the communication of all weaker information. We therefore introduce an abstraction of $\mathscr{F}$ that incorporates these properties of *asking more* and *telling less*.

Definition 22 (Abstraction operator). For every set W of traces we denote by $\alpha(W)$ the smallest set V that contains W and additionally satisfies:
- $w_1 \cdot c?\psi \cdot w_2 \in V \Rightarrow w_1 \cdot c?\varphi \cdot w_2 \in V$, provided that $\psi \sqsubseteq \varphi$
- $w_1 \cdot c!\varphi \cdot w_2 \in V \Rightarrow w_1 \cdot c!\psi \cdot w_2 \in V$, provided that $\psi \sqsubseteq \varphi$.

The operator α saturates a set of traces with all traces that derive via asking more and telling less information.

Definition 23 (The semantics $\mathscr{F}_\alpha$). The semantics $\mathscr{F}_\alpha$ is obtained from $\mathscr{F}$ as follows:

$$\mathscr{F}_\alpha(A) = \alpha(\mathscr{F}(A)),$$

where $\alpha(\mathscr{F}(A))$ denotes the extension of α to sets of failure traces.

The failure semantics $\mathscr{F}_\alpha$ is correct and compositional with respect to the observable $\mathscr{O}$. The correctness of $\mathscr{F}_\alpha$ follows from the correctness of $\mathscr{F}$.

Theorem 24 (Compositionality of $\mathscr{F}_\alpha$). *The compositionality of the parallel operator is phrased as follows, where $\mathscr{F}_\alpha^\omega(A)$ consists of all the infinite sequences w of*

communication actions such that for every prefix w' of w, we have $\langle w', t\rangle \in \mathscr{F}_\alpha(A)$, for some t:

$$
\begin{array}{l}
\mathscr{F}_\alpha(A_1 \parallel A_2) \\
= \\
\{\langle w, t\rangle \mid w \in (w_1 \parallel w_2), t \in (t_1 \parallel t_2),\ \langle w_1, t_1\rangle \in \mathscr{F}_\alpha(A_1), \\
\langle w_2, t_2\rangle \in \mathscr{F}_\alpha(A_2)\} \\
\cup \\
\{\langle w \cdot w', \bot\rangle \mid w \in (w_1 \parallel w_2),\ (w_1 \cdot u) \in \mathscr{F}^\omega_\alpha(A_1),\ (w_2 \cdot v) \in \mathscr{F}^\omega_\alpha(A_2), \\
u \uparrow v\}.
\end{array}
$$

Additionally, the compositionality of the encapsulation operator is given by

$$
\begin{array}{l}
\mathscr{F}_\alpha(\delta_H(A)) \\
= \\
\{\langle w, (\varphi, F)\rangle \mid \langle w, (\varphi, F')\rangle \in \mathscr{F}_\alpha(A),\ chan(w) \cap H = \emptyset, \\
F \subseteq F' \cup \{c!\psi, c?\psi \mid c \in H, \psi \in \mathbf{C}\}\} \\
\cup \\
\{\langle w, \bot\rangle \mid \langle w, \bot\rangle \in \mathscr{F}_\alpha(A),\ chan(w) \cap H = \emptyset\}.
\end{array}
$$

Proof. The proof of this theorem is a slight modification of the proof of Theorem 19, where we make use of the following property: $\alpha(w_1 \parallel w_2) = \alpha(w_1) \parallel \alpha(w_2)$, for all traces w_1 and w_2. □

The abstraction operator α thus simply distributes over the semantic counterparts of the operators of parallel composition and encapsulation.

As mentioned before, failure sets are typically comprised of an infinite number of refused communication actions. In order to prove full abstraction of the semantics $\mathscr{F}_\alpha$, we therefore need a *compactness* property, which is proved in the following theorem.

Theorem 25 (Compactness of the failure semantics $\mathscr{F}_\alpha$). *If* $\langle w, (\varphi, F')\rangle \in \mathscr{F}_\alpha(A)$, *for every* finite *subset F' of a given set of communication actions F, then also* $\langle w, (\varphi, F)\rangle \in \mathscr{F}_\alpha(A)$.

Proof. Suppose that for all finite $F' \subseteq F$ we have $\langle w, (\varphi, F')\rangle \in \mathscr{F}_\alpha(A)$. We assume that F is infinite, as the finite case is trivial. Let $l_1, l_2, \ldots$ be an enumeration of the elements of F. Consider the collection $\mathbf{F}$ of subsets of F:

$$
\begin{array}{ll}
F_0 & = \emptyset \\
F_{j+1} & = F_j \cup \{l_j\}
\end{array}
$$

Consider next the collection C of computations of A that generate the word w, yield the store φ and refuse a set $F_i \in \mathbf{F}$. We assume that there is a bound k on the number of successive τ-steps that can occur in each computation in C. For, if such a bound does not exist then from the fact that the operational semantics gives rise to only finitely branching computation trees, we conclude via König's lemma, which states that any finitely branching tree with an infinite number of nodes has an infinite branch

(Lemma 20), that there must be a computation in C that after having generated a prefix of w goes into an infinite loop of τ-steps. Then since such a diverging computation gives rise to chaos, we immediately obtain $\langle w,(\varphi,F)\rangle \in \mathscr{F}_\alpha(A)$.

Hence, we assume that such a bound k exists. As the computation tree is finitely branching, there also exists a bound on the length of the computations in C, and therefore C is finite. However, as there are infinitely many sets $F_i \in \mathbf{F}$, the pigeon-hole principle then tells us that there must be a computation p in C that refuses an infinite number of sets in $\mathbf{F}$.

We claim that p also refuses F. To see this consider an arbitrary element l_i of F. As $l_i \in F_{i+1}$ and p refuses an infinite number of failure sets of $\mathbf{F}$, there must be an index $j \geqslant i+1$ such that p refuses F_j. As this set F_j also includes the action l_i, we obtain that the computation p refuses l_i. As l_i was chosen arbitrarily from F, we conclude that p refuses all elements of F (or more). By the definition of the failure semantics $\mathscr{F}_\alpha$, which says that any subset of a failure set is also a failure set, we obtain $\langle w,(\varphi,F)\rangle \in \mathscr{F}_\alpha(A)$, which completes the proof. □

Finally, we are in the position to show our main result, namely that the semantics $\mathscr{F}_\alpha$ constitutes a fully abstract semantics with respect to our notion $\mathscr{O}$ of observable behaviour.

Theorem 26 (Full abstraction of $\mathscr{F}_\alpha$). *For any two agents A and B, the following holds*:

$$(\mathscr{O}(C[A]) = \mathscr{O}(C[B]),\ \textit{for all contexts}\ C) \Rightarrow \mathscr{F}_\alpha(A) = \mathscr{F}_\alpha(B).$$

Proof. The proof proceeds by contraposition. Suppose $\mathscr{F}_\alpha(A) \neq \mathscr{F}_\alpha(B)$ then without loss of generality there must exist a tuple $\langle w,\bot\rangle$ or $\langle w,(\varphi,F)\rangle \in \mathscr{F}_\alpha(A) - \mathscr{F}_\alpha(B)$, for some *finite* set F according to the compactness property that is established in Theorem 25.

The idea is then to define a context C such that $\mathscr{O}(C[A]) \neq \mathscr{O}(C[B])$. In order to achieve this, we define the complement $\tilde{l}$ of a communication action l as follows. For all $c \in Chan$ and $\varphi \in \mathbf{C}$:

$$\begin{aligned} \widetilde{c!\varphi} &= c?\varphi \\ \widetilde{c?\varphi} &= (\mathtt{tell}(\varphi) \cdot c!\varphi). \end{aligned}$$

Let w be given by $l_1 l_2 \cdots l_n$.

First, we consider the case $\langle w,\bot\rangle$. Consider the context C defined by

$$\langle \widetilde{l_1} \cdots \widetilde{l_n}, \mathit{true}\rangle \parallel \cdot$$

It is easy to see that $\mathscr{O}(C[A])$ contains all constraints. We claim that this does not hold for $\mathscr{O}(C[B])$. For, suppose $\mathscr{O}(C[B])$ contains all constraints, which means $\mathit{false} \in \mathscr{O}(C[B])$. There are two possibilities for generating *false*. The first is the case that $B \overset{u}{\Rightarrow}$

B' with $store(B') = false$ for some prefix u of w (note that because of the semantics of the communication actions, this also covers the case in which the constraints of w are inconsistent themselves). The second is the case that $B \stackrel{u}{\Rightarrow} B' \stackrel{\tau}{\Rightarrow} \Omega$, for some prefix u of w (modulo asking more and telling less). Then by the definition of $\mathscr{F}_\alpha$ in which divergence and inconsistency gives rise to chaos, we conclude $\langle u, \bot\rangle \in \mathscr{F}_\alpha(B)$ and hence, as u is a prefix of w we have: $\langle w, \bot\rangle \in \mathscr{F}_\alpha(B)$. This yields a contradiction.

Next, we consider the case $\langle w, (\varphi, F)\rangle$. Consider the following context C:

$$\langle \widetilde{l_1} \cdots \widetilde{l_n}\rangle \cdot \mathtt{tell}(ok_1) \cdot \Sigma_{l\in F}(l \cdot \mathtt{tell}(ok_2)), true\rangle \parallel \cdot$$

where ok_1 and ok_2 denote some constraints that do not occur in the multi-agent systems A and B and $\Sigma_{l\in F}$ denotes the non-deterministic choice between the actions in F. Note that here it is crucial that F is a *finite* set. The idea of this context is that it offers the complements of the actions of w then produces a signal ok_1, and finally, offers the actions in the failure set F. Additionally, it produces the signal ok_2 in case one of the elements in F is accepted. It is easy to see that $(\varphi \sqcup ok_1) \in \mathscr{O}(C[A])$.

We claim that $(\varphi \sqcup ok_1) \notin \mathscr{O}(C[B])$. For, otherwise it would be the case that $B \stackrel{u}{\Rightarrow} B'$ with $u = w$ modulo asking more and telling less, as the signal ok_1 has been produced, and B' refuses the set F or more, otherwise the signal ok_2 would have been produced. Then via the definition of $\mathscr{F}_\alpha$ which says that any subset of a failure set is also a failure set, we conclude $\langle w, (\varphi, F)\rangle \in \mathscr{F}_\alpha(B)$. This yields a contradiction and hence we obtain $\mathscr{O}(C[A]) \neq \mathscr{O}(C[B])$. □

This ends the construction of a compositional and fully abstract model for the multi-agent programming language ACPL.

5. Conclusions and future research

In this paper, we have developed a compositional semantics for the multi-agent programming language ACPL, based on a generalisation of traditional failure semantics, which is shown to be fully abstract with respect to observing the global information stores of terminating computations.

Our main goal is now to extend our failure semantics to more sophisticated agent communication languages. For example, currently, we are investigating an extension of the model which incorporates agent *ontologies* such that communication of information additionally involves the *translation* of information from the ontology of the sender to that of the receiving agent, as outlined in [22]. Furthermore, we aim to study the incorporation of non-monotonically increasing information stores as described in [20]. Another interesting extension of the framework concerns features that allow a dynamic reconfiguration of the communication network.

Acknowledgements

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

Appendix

We consider the compositionality of the failure semantics.

Proof of Theorem 19. We start with the parallel composition. Let us denote the right-hand side of the equation by $\mathscr{G}(A_1 \parallel A_2)$. We show $\mathscr{F}(A_1 \parallel A_2) \supseteq \mathscr{G}(A_1 \parallel A_2)$. There are four cases.

Case 1: Suppose $\langle w,(\varphi_1 \sqcup \varphi_2,F)\rangle \in \mathscr{G}(A_1 \parallel A_2)$, where we have $w \in (w_1 \parallel w_2)$, $\langle w_1,(\varphi_1,F_1)\rangle \in \mathscr{F}(A_1)$, that is,

$$A_1 \stackrel{w_1}{\Rightarrow} B_1,\ \tau \notin I(B_1),\ F_1 \subseteq \overline{I(B_1)},\ \varphi_1 = store(B_1),$$

and $\langle w_2,(\varphi_2,F_2)\rangle \in \mathscr{F}(A_2)$, that is,

$$A_2 \stackrel{w_2}{\Rightarrow} B_2,\ \tau \notin I(B_2),\ F_2 \subseteq \overline{I(B_2)},\ \varphi_2 = store(B_2),$$

such that $\varphi_1 \sqcup \varphi_2 \neq false$, $\overline{F_1} \frown \overline{F_2}$ and $F \subseteq F_1 \cap F_2$.

Note that the condition $\overline{F_1} \frown \overline{F_2}$ boils down to the requirement that B_1 and B_2 cannot communicate with each other, since $\overline{F_i}$ is a superset of the initial communication actions of A_i $(i=1,2)$. From this fact together with the assumptions $\tau \notin I(B_1)$ and $\tau \notin I(B_2)$ we obtain

$$\tau \notin I(B_1 \parallel B_2) \text{ (i)}.$$

Furthermore, since B_1 and B_2 cannot communicate, $F_1 \subseteq \overline{I(B_1)}$, $F_2 \subseteq \overline{I(B_2)}$ and $F \subseteq F_1 \cap F_2$ we derive

$$\begin{array}{ll} \overline{I(B_1)} \cap \overline{I(B_2)} = \overline{I(B_1 \parallel B_2)} & \Rightarrow \\ F_1 \cap F_2 \subseteq \overline{I(B_1 \parallel B_2)} & \Rightarrow \\ F \subseteq \overline{I(B_1 \parallel B_2)} & \text{(ii)}. \end{array}$$

Finally, as $store(B_1 \parallel B_2) = \varphi_1 \sqcup \varphi_2 \neq false$, we obtain via (i) and (ii) what was to be shown: $\langle w,(\varphi_1 \sqcup \varphi_2,F)\rangle \in \mathscr{F}(A_1 \parallel A_2)$. Note that here and in the sequel we make implicit use of the following property of the parallel composition: for all $w \in (w_1 \parallel w_2)$

$$(A_1 \stackrel{w_1}{\Rightarrow} B_1 \text{ and } A_2 \stackrel{w_2}{\Rightarrow} B_2) \ \Leftrightarrow\ (A_1 \parallel A_2 \stackrel{w}{\Rightarrow} B_1 \parallel B_2).$$

Case 2: Consider $\langle w,\perp\rangle \in \mathscr{G}(A_1 \parallel A_2)$, where $w \in (w_1 \parallel w_2)$, $\langle w_1,(\varphi_1,F_1)\rangle \in \mathscr{F}(A_1)$ and $\langle w_2,(\varphi_2,F_2)\rangle \in \mathscr{F}(A_2)$ such that $A_1 \stackrel{w_1}{\Rightarrow} B_1$, $A_2 \stackrel{w_2}{\Rightarrow} B_2$ as well as $store(B_1 \parallel B_2) = false$. We immediately obtain $\langle w,\perp\rangle \in \mathscr{F}(A_1 \parallel A_2)$.

Case 3: Consider $\langle w,\perp\rangle \in \mathscr{G}(A_1 \parallel A_2)$, where $w \in (w_1 \parallel w_2)$ and without loss of generality $\langle w_1,\perp\rangle \in \mathscr{F}(A_1)$ and $\langle w_2,t_2\rangle \in \mathscr{F}(A_2)$. Then either $A_1 \stackrel{w_1}{\Rightarrow} B_1$ with $store(B_1) = false$ or $B_1 \Rightarrow \Omega$. We immediately derive that $A_1 \parallel A_2$ also yields an inconsistent store or diverges. The latter follows from the following property:

$$A \Rightarrow \Omega \text{ implies } A \parallel B \Rightarrow \Omega \quad \text{for all } B.$$

Case 4: Finally, there is the case $\langle w \cdot w', \perp \rangle \in \mathscr{G}(A_1 \parallel A_2)$, where $A_1 \parallel A_2 \stackrel{w}{\Rightarrow} B_1 \parallel B_2$ and there exist infinite sequences u and v such that for all prefixes u' of u and v' of v we have

$$u' \uparrow v', \ B_1 \stackrel{u'}{\Rightarrow} C_1 \text{ and } B_2 \stackrel{v'}{\Rightarrow} C_2 \quad \text{for some } C_1 \text{ and } C_2.$$

Hence, there does not exist a bound on the number of successive internal steps in the computations of $B_1 \parallel B_2$. As the computation tree of $B_1 \parallel B_2$ is finitely branching, Lemma 20 then yields the existence of a computation with an infinite number of successive τ-steps. Hence, we have $B_1 \parallel B_2 \Rightarrow \Omega$, yielding $\langle w \cdot w', \perp \rangle \in \mathscr{F}(A_1 \parallel A_2)$.

For the converse inclusion $\mathscr{F}(A_1 \parallel A_2) \subseteq \mathscr{G}(A_1 \parallel A_2)$ the same line of reasoning can be applied.

Next, we consider encapsulation. Let us denote the right-hand side by $\mathscr{G}(\delta_H(A))$. We show $\mathscr{F}(\delta_H(A)) \supseteq \mathscr{G}(\delta_H(A))$. Suppose $\langle w, (store(B), F) \rangle \in \mathscr{G}(\delta_H(A))$, where $A \stackrel{w}{\Rightarrow} B$. Then as $chan(w) \cap H = \emptyset$, we also have

$$\delta_H(A) \stackrel{w}{\Rightarrow} \delta_H(B) \text{ (i)}.$$

Additionally, as $\tau \notin I(B)$, which implies $\tau \notin I(\delta_H(B))$, we also have

$$\tau \notin I(\delta_H(B)) \text{ (ii)}.$$

Finally, via the fact: $\overline{I(\delta_H(B))} = \overline{I(B)} \cup \{c!\psi, c?\psi \mid c \in H, \psi \in \mathbf{C}\}$, we obtain

$$F \subseteq \overline{I(\delta_H(B))} \text{ (iii)}.$$

Then from (i) to (iii) and the fact $store(\delta_H(B)) = store(B)$ we derive what was to be shown: $\langle w, (store(B), F) \rangle \in \mathscr{F}(\delta_H(A))$.

Secondly, suppose $\langle w \cdot w', \perp \rangle \in \mathscr{G}(\delta_H(A))$, where $chan(w) \cap H = \emptyset$ and $A \stackrel{w}{\Rightarrow} B$ with $B \Rightarrow \Omega$ or $store(B) = false$. Then also $\delta_H(A) \stackrel{w}{\Rightarrow} \delta_H(B)$ with $\delta_H(B) \Rightarrow \Omega$ or $store(\delta_H(B)) = false$, which yields $\langle w \cdot w', \perp \rangle \in \mathscr{F}(\delta_H(A))$. □

References

[1] J.L. Austin, How to do Things with Words, Oxford University Press, Oxford, 1962.

[2] J.A. Bergstra, J.W. Klop, Process algebra for synchronous communication, Inform. and Control 60 (1984) 109–137.

[3] J.A. Bergstra, J.W. Klop, E.-R. Olderog, Readies and failures in the algebra of communicating processes, SIAM J. Comput. 17 (1988) 1134–1177.

[4] L. Brim, D. Gilbert, J.-M. Jacquet, M. Křetínský , A process algebra for synchronous concurrent constraint programming, in: M. Hanus, M. Rodriquez-Artalejo (Eds.), Proc. 5th Conf. Algebraic and Logic Programming, Lecture Notes in Computer Science, Vol. 1139, Springer, Berlin, 1996, pp. 165–178.

[5] S.D. Brookes, C.A.R. Hoare, W. Roscoe, A theory of communicating sequential processes, J. ACM 31 (1984) 499–560.

[6] F.S. de Boer, C. Palamidessi, A fully abstract model for concurrent constraint programming, in: Proc. Fourth Internat. Joint Conf. Theory and Practice of Software Development (TAPSOFT), Lecture Notes in Computer Science, Vol. 493, Springer, Berlin, 1991, pp. 296–319.

[7] F.S. de Boer, R.M. van Eijk, W. van derHoek, J.-J.Ch. Meyer, Failure semantics for the exchange of information in multi-agent systems, in: C. Palamidessi (Ed.), Proc. 11th Internat. Conf. Concurrency Theory (CONCUR 2000), Lecture Notes in Computer Science, Vol. 1877, Springer, Heidelberg, 2000, pp. 214–228.
[8] T. Finin, D. McKay, R. Fritzson, R. McEntire, KQML: An information and knowledge exchange protocol, in: K. Fuchi, . Yokoi (Eds.), Knowledge Building and Knowledge Sharing, Ohmsha and IOS Press, Amsterdam, 1994.
[9] J.Y. Halpern, Y. Moses, A guide to the completeness and complexity for modal logics of knowledge and belief, Artificial Intelligence 54 (1992) 319–379.
[10] L. Henkin, J.D. Monk, A. Tarski, Cylindric Algebras (Part I), North-Holland, Amsterdam, 1971.
[11] C.A.R. Hoare, Communicating sequential processes, Commun. ACM 21 (8) (1978) 666–677.
[12] J. Pitt, A. Mamdani, Some remarks on the semantics of FIPA's agent communication language, Autonomous Agents and Multi-Agent Systems 2 (4) (1999) 333–356.
[13] R. Reiter, On closed world data bases, in: H. Gaillaire, J. Minker (Eds.), Logic and Data Bases, Plenum Press, New York, 1978, pp. 55–76.
[14] J.-H. Réty, Distributed concurrent constraint programming, Fund. Inform. 34 (3) (1998) 323–346.
[15] V.A. Saraswat, M. Rinard. Concurrent constraint programming, in: Proc. 17th ACM Symp. Principles of Programming Languages (POPL'90), 1990, pp. 232–245.
[16] V.A. Saraswat, M. Rinard, P. Panangaden, Semantic foundations of concurrent constraint programming, in: Proc. 18th ACM Symp. Principles of Programming Languages (POPL'91), 1991, pp. 333–352.
[17] J.R. Searle, Speech Acts: An Essay in the Philosophy of Language, Cambridge University Press, Cambridge, 1969.
[18] F. van Breughel, Failures, finiteness and full abstraction, in: S. Brookes, M. Mislove (Eds.), Proc. Thirteenth Conf. Mathematical Foundations of Programming Semantics, Electronic Notes in Theoretical Computer Science, Vol. 6, Elsevier, New York, 1997.
[19] R.M. van Eijk, F.S. de Boer, W. van derHoek, J.-J.Ch. Meyer, Systems of communicating agents, in: Henri Prade (Ed.), Proc. 13th Biennial European Conf. Artificial Intelligence (ECAI'98), Wiley, Chichester, United Kingdom, 1998, pp. 293–297.
[20] R.M. van Eijk, F.S. de Boer, W. van derHoek, J.-J.Ch. Meyer, Information-passing and belief revision in multi-agent systems, in: J.P.M. Müller, M.P. Singh, A.S. Rao (Eds.), Intelligent Agents V, Proc. 5th Internat. Workshop on Agent Theories, Architectures, and Languages (ATAL'98), Lecture Notes in Artificial Intelligence, Vol. 1555, Springer, Heidelberg, 1999.
[21] R.M. van Eijk, F.S. de Boer, W. van derHoek, J.-J.Ch. Meyer, Open multi-agent systems: Agent communication and integration, in: N.R. Jennings, Y. Lespèrance (Eds.), Intelligent Agents VI, Proc. 6th Internat. Workshop on Agent Theories, Architectures, and Languages (ATAL'99), Lecture Notes in Artificial Intelligence, Vol. 1757, Springer, Heidelberg, 2000, pp. 218–232.
[22] R.M. van Eijk, F.S. de Boer, W. van derHoek, J.-J.Ch. Meyer, On dynamically generated ontology translators in agent communication, Internat. J. Intell. Systems 16 (5) (2001) 587–607.
[23] M. Wooldridge, Verifiable semantics for agent communication languages, in: Proc. 3rd Internat. Conf. Multi-Agent Systems (ICMAS'98), IEEE Computer Society, Los Alamitos, California, 1998, pp. 349–356.
[24] M. Wooldridge, N. Jennings, Intelligent agents: theory and practice, The Knowledge Eng. Rev. 10 (2) (1995) 115–152.