Failure Semantics for the Exchange of Information in Multi-Agent Systems

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

Utrecht University, Department of Computer Science
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{ frankb, rogier, wiebe, jj}@cs.uu.nl

Abstract. In this paper, we present a semantic theory for the exchange of information in multi-agent systems. We define a concurrent programming language for systems of agents that maintain their own private stores of information and that interact with each other by means of a synchronous communication mechanism that allows for 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.

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 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 [15]. 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 language KQML [8]. However none of these communication languages have been given a fully formal account of their semantics [14]. The main contribution of this paper is the introduction of a formal semantic theory for the exchange of information in multi-agent systems.

Concurrent Programming

We introduce a concurrent programming language that concentrates on the information processing aspects of agents. The underlying computational model of the language has already been introduced in [5]. The basic operations of the language for the processing of information are the *ask* and *tell* operations of Concurrent Constraint Programming (CCP) [13]. 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 our programming language, however, the global store of Concurrent Constraint Programming 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.

Fully Abstract Semantics

The main result of this paper is a compositional semantics for the multi-agent language 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 [4]. 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* [2]. 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.

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 the language KQML [8].

Other approaches that relate to our programming language include the work of Réty on distributed concurrent constraint programming [12]. One of the differences with our approach is that in this framework 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 [3]. 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.

Overview

The remainder of this paper is organised as follows. In Section 2, we give the syntax of our multi-agent programming language, 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, which like Concurrent Constraint Programming is parameterised by a constraint system that is used to represent information.

Definition 1 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 [13] 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 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 ($\exists_x \varphi \sqcup \psi$ 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. $d_{xx} = \mathit{true}$,
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.

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

Definition 3 *(Basic actions)* Given a cylindrical constraint system **C** the basic actions of the programming language are defined as follows:

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

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 $\mathsf{ask}(\varphi)$ by an agent consists of checking whether the private store of the agent entails φ. On the other hand, the execution of $\mathsf{tell}(\varphi)$ consist of adding φ to the private store.

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 $\bar{x}$ denotes a sequence of variables which constitute the actual parameters of the call.

Definition 5 *(Multi-agent systems)* A multi-agent system A is defined by

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

A basic agent in a multi-agent system is represented by a tuple $\langle D, S, \varphi \rangle$ consisting first of all of a set D of procedure declarations of the form $p(\bar{x}) :- S$, where $\bar{x}$ denote the formal parameters of p and S denotes its body. 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 $\mathsf{ask}(\psi)$ and $\mathsf{tell}(\psi)$. 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.

Finally, the encapsulation operator δ_H with $H \subseteq \mathit{Chan}$, which stems from the process algebra ACP, is used to define local communication channels [1]. 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 $\mathsf{ask}(\varphi)$ or $\mathsf{tell}(\varphi)$, 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 *(The local transition system)* We have the following transitions for the basic actions.

- $\langle c!\varphi, \psi \rangle \xrightarrow{c!\varphi} \langle E, \psi \rangle \quad$ if $\varphi \sqsubseteq \psi$
- $\langle c?\varphi, \psi \rangle \xrightarrow{c?\varphi} \langle E, \psi \sqcup \varphi \rangle$
- $\langle \mathsf{ask}(\varphi), \psi \rangle \xrightarrow{\tau} \langle E, \psi \rangle \quad$ if $\varphi \sqsubseteq \psi$
- $\langle \mathsf{tell}(\varphi), \psi \rangle \xrightarrow{\tau} \langle E, \psi \sqcup \varphi \rangle$

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 $\mathsf{tell}(\varphi)$. However, for technical convenience only we have adopted in this paper the first approach.

The actions $\mathsf{ask}(\varphi)$ and $\mathsf{tell}(\varphi)$ 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, recursive procedure calls and the programming constructs for non-deterministic choice and parallel composition. Here we list only the following generalisation of the rule for local variables as defined in [5].

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

where the label l' equals τ in case $l = \tau$, and $l' = c!\exists_x\varphi$ ($l' = c?\exists_x\varphi$) in case $l = c!\varphi$ ($l = c?\varphi$), for all c and φ.

The syntax of the language is thus 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 $\exists_x^{true} S$, denoting that the local constraints on x are initially empty.

Thus, in our framework we have a *global* store that is distributed over the agents, which is formally defined below. Each agent has direct access only to its *private* store. Information in the private store about the *global* variables can be communicated to the other agents. The *local* variables of an agent, which are introduced by the hiding operator $\exists_x$, however, cannot be referred to in communications. This explains why in the context of $\exists_x^{\varphi} S$ the constraints on the local variable x in a communication are existentially quantified.

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

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

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

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

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

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

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:

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

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

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

Definition 8 *(Global store)*
We define *store*(A) by induction on the structure of the agent system A:

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

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.

Definition 9 *(Operational Semantics)*
In defining the operational semantics we make use of several auxiliary notions:

– The notation $A \Longrightarrow B$ indicates the existence of a terminating computation that consists of internal computation steps only: $A \xrightarrow{\tau} \cdots \xrightarrow{\tau} B$. Additionally, $B \overset{\tau}{\not\longrightarrow}$ 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 $A \Longrightarrow \Omega$ 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*.

Then we define:

$$\begin{aligned} &\mathcal{O}(A) = \\ &\{\mathit{store}(B) \mid A \Longrightarrow B \overset{\tau}{\not\longrightarrow}\} \,\cup \\ &\{\varphi \mid \varphi \in \mathcal{C},\ A \Longrightarrow \Omega \text{ or } A \Longrightarrow B,\ \text{with } \mathit{store}(B) = \mathit{false}\} \end{aligned}$$

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

$$\begin{aligned} &\mathcal{O}(A) = \\ &\{\mathit{store}(A) \mid A \overset{\tau}{\not\longrightarrow}\} \,\cup \\ &\{\varphi \mid \varphi \in \mathcal{O}(B),\ \text{with } A \xrightarrow{\tau} B,\ \text{or } \mathit{store}(A) = \mathit{false}\} \end{aligned}$$

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

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 described notion of observables, we introduce a failure semantics that records for each sequence of communications of an agent system the corresponding failure set and the corresponding private store.

Definition 10 *(Failure semantics)*
First, we introduce the following notions:

- The transition relation $A \overset{w}{\Longrightarrow} B$, 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:

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

- We denote by $\overline{X}$, with X a set of communication actions, the set of communication actions $c?\varphi$ ($c!\varphi$) such that there does not exist a complementary $c!\psi \in X$ ($c?\psi \in X$) with $\varphi \sqsubseteq \psi$ ($\psi \sqsubseteq \varphi$).

The failure semantics $\mathcal{F}$ is then defined as follows:

$$\begin{array}{l} \mathcal{F}(A) = \\ \{\langle w, (\mathit{store}(B), F)\rangle \mid A \overset{w}{\Longrightarrow} B,\ \tau \notin I(B),\ F \subseteq \overline{I(B)}\} \cup \\ \{\langle w \cdot w', \bot\rangle \mid A \overset{w}{\Longrightarrow} B, \text{ with } B \Longrightarrow \Omega \text{ or } \mathit{store}(B) = \mathit{false}\}. \end{array}$$

Note that w' ranges over all sequences of communication actions. Alternatively, we have the following *recursive* definition of $\mathcal{F}$:

$$\begin{array}{l} \mathcal{F}(A) = \\ \{\langle \epsilon, (\mathit{store}(A), F)\rangle \mid \tau \notin I(A),\ F \subseteq \overline{I(A)}\} \cup \\ \{\langle l \cdot w, t\rangle \mid A \xrightarrow{l} B, \text{ with } \langle w, t\rangle \in \mathcal{F}(B) \text{ or } \mathit{store}(B) = \mathit{false}\} \end{array}$$

where l denotes a communication action. The above non-recursive definition then can be shown to correspond with the *greatest fixpoint* of this recursive equation.

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

This definition of the failure sets differs from the usual one which is simply defined as a subset of the *complement* of the set of initial actions. The standard failure sets as such indicate the actions which the process itself cannot perform. However, a communication action which an agent cannot perform is not necessarily an action which it refuses for synchronisation, as shown in the following example.

Example 11 Consider the agents

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

The set of initial actions of A and B are clearly different. However, it is easy to see that $\overline{\overline{X}} = \overline{\overline{Y}}$, for $X = \{c!(\varphi \sqcup \psi)\}$ and $Y = \{c!(\varphi \sqcup \psi), c!\varphi\}$. In fact, in general we have that $\overline{\overline{X}}$, for any set of communication actions X, consists of exactly those communication actions $c?\varphi$ $(c!\varphi)$ such that $c?\psi \in X$ $(c!\psi \in X)$, with $\psi \sqsubseteq \varphi$ $(\varphi \sqsubseteq \psi)$. That is, $\overline{\overline{X}}$ contains all the communication actions which derive from X by *asking more* or *telling less*.

Of particular interest is to observe here that the failure sets can be *infinite* (but denumerable) sets of communication actions (that is, when the underlying constraint system contains infinitely but denumerable many constraints).

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

Theorem 12 *(Correctness of $\mathcal{F}$)*

$$\varphi \in \mathcal{O}(A) \text{ iff } \langle \epsilon, (\varphi, F) \rangle \in \mathcal{F}(A), \text{ for some } F, \text{ or } \langle \epsilon, \perp \rangle \in \mathcal{F}(A)$$

Moreover, we have the following compositionality result.

Theorem 13 *(Compositionality of $\mathcal{F}$)*
In order to define the semantics of the parallel composition of agents we first introduce the parallel composition of sequences of communication actions. Given such sequences w_1 and w_2 we define $w_1 \parallel w_2$ as the following set of communication sequences [1]. First, we define $\epsilon \parallel \epsilon$ to be $\{\epsilon\}$, and for the other cases:

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

Additionally, let $X \mid Y$, where X and Y are sets of communication actions, indicate that there exists $c?\varphi \in X$ and $c!\psi \in Y$ (or vice versa) such that $\varphi \sqsubseteq \psi$.

Finally, for the compositional modelling of chaos introduced by internally diverging computations and inconsistent constraint stores, we need the following notions.

- $\mathcal{F}^\omega(A)$ denotes all the *infinite* sequences w of communication actions such that for every prefix w' of w, we have $\langle w', t\rangle \in \mathcal{F}(A)$, for some t
- the predicate $v_1 \uparrow v_2$ satisfies:

$$(c!\psi \cdot v) \uparrow (c?\varphi \cdot v') = (\varphi \sqsubseteq \psi \text{ and } v \uparrow v')$$

- Finally, the composition of termination modes is defined as follows:
  - Provided that $\overline{F_1} \not| \overline{F_2}$ and $\varphi_1 \sqcup \varphi_2 \neq \mathit{false}$ 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 $\varphi_1 \sqcup \varphi_2 = \mathit{false}$ 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. Additionally, note that in the first equation we make explicit use of *infinite* failure sets since, given an infinite underlying constraint system, for any *finite* failure sets F_1 and F_2 we have that $\overline{F_1} \mid \overline{F_2}$: choose a constraint φ such that neither $\varphi \sqsubseteq \psi$ nor $\psi \sqsubseteq \varphi$, for any constraint ψ occurring in F_1 or F_2. Then we have for any channel c that, for example, $c!\varphi \in \overline{F_1}$ and $c?\varphi \in \overline{F_2}$. We then have:

$$\begin{array}{l}\mathcal{F}(A_1 \parallel A_2) = \\ \{\langle w, t\rangle \mid w \in (w_1 \parallel w_2),\ t \in (t_1 \parallel t_2),\ \langle w_i, t_i\rangle \in \mathcal{F}(A_i),\ i = 1, 2\} \cup \\ \{\langle w \cdot w', \bot\rangle \mid w \in (w_1 \parallel w_2) \mid (w_1 \cdot u) \in \mathcal{F}^\omega(A_1),\ (w_2 \cdot v) \in \mathcal{F}^\omega(A_2) \text{ and } u \uparrow v\}\end{array}$$

and additionally:

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

Divergence of the parallel composition $A_1 \parallel A_2$ thus stems from the fact that A_1 or A_2 *themselves* diverge, or that each of them generates an infinite sequence of communication actions such that each individual communication action in one sequence matches with the communication action in the other sequence. The failure semantics $\mathcal{F}$ however still distinguishes too many agent systems, that is, it is not fully abstract with respect to the observables $\mathcal{O}$, as shown by the following example.

Example 14 Consider again the agents

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

The failure semantics distinguishes these two agents as we have:

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

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

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

Definition 15 *(Abstraction operator)*
For every set W of traces we denote by $\alpha(W)$ the smallest set V that contains W and additionally satisfies:

- $w_1 \cdot c?\varphi \cdot w_2 \in V \Rightarrow w_1 \cdot c?\psi \cdot w_2 \in V$, provided that $\varphi \sqsubseteq \psi$
- $w_1 \cdot c!\varphi \cdot w_2 \in V \Rightarrow w_1 \cdot c!\psi \cdot w_2 \in V$, provided that $\psi \sqsubseteq \varphi$.

Definition 16 *(The semantics $\mathcal{F}_\alpha$)*
We define the semantics $\mathcal{F}_\alpha$ as follows:

$$\mathcal{F}_\alpha(A) = \alpha(\mathcal{F}(A))$$

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

The semantics $\mathcal{F}_\alpha$ are correct and compositional with respect to the observable $\mathcal{O}$. The compositionality is established in the following theorem.

Theorem 17 *(Compositionality of $\mathcal{F}_\alpha$)*
We have:

$$\begin{aligned}
&\mathcal{F}_\alpha(A_1 \parallel A_2) = \\
&\{\langle w, t\rangle \mid w \in (w_1 \parallel w_2),\ t \in (t_1 \parallel t_2),\ \langle w_i, t_i\rangle \in \mathcal{F}_\alpha(A_i),\ i = 1,2\} \cup \\
&\{\langle w \cdot w', \perp\rangle \mid w \in (w_1 \parallel w_2) \mid (w_1 \cdot u) \in \mathcal{F}^\omega_\alpha(A_1),\ (w_2 \cdot v) \in \mathcal{F}^\omega_\alpha(A_2) \text{ and } u \uparrow v\}
\end{aligned}$$

where $\mathcal{F}^\omega_\alpha(A)$ consists of all the *infinite* sequences w of communication actions such that for every prefix w' of w, we have $\langle w', t\rangle \in \mathcal{F}_\alpha(A)$, for some t. Additionally, we have

$$\begin{aligned}
&\mathcal{F}_\alpha(\delta_H(A)) = \\
&\{\langle w, (\varphi, F)\rangle \mid \langle w, (\varphi, F')\rangle \in \mathcal{F}_\alpha(A), \\
&\mathit{chan}(w) \cap H = \emptyset \text{ and } F \subseteq F' \cup \{c!\psi, c?\psi \mid c \in H\}\} \cup \\
&\{\langle w, \perp\rangle \mid \langle w, \perp\rangle \in \mathcal{F}_\alpha(A),\ \mathit{chan}(w) \cap H = \emptyset\}
\end{aligned}$$

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

In order to prove full abstractness of the semantics $\mathcal{F}_\alpha$ we need the following *compactness* property.

Theorem 18 *(Compactness of the failure semantics $\mathcal{F}_\alpha$)*
We have that if $\langle w, (\varphi, F')\rangle \in \mathcal{F}_\alpha(A)$, for every *finite* subset F' of a given set of communication actions F, then also $\langle w, (\varphi, F)\rangle \in \mathcal{F}_\alpha(A)$.

Proof. Suppose that for all finite $F' \subseteq F$ we have $\langle w, (\varphi, F')\rangle \in \mathcal{F}_\alpha(A)$. 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 following collection $\mathbf{F}$ of subsets of F:

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

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 language gives rise to only finitely branching computation trees, we conclude via Königs Lemma, which states that any finitely branching tree with an infinite number of nodes has an infinite path, 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 $\langle w, (\varphi, F)\rangle \in \mathcal{F}_\alpha(A)$.

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 therefor 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 finite sets in $\mathbf{F}$. We claim that p also refuses F.

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 $j \geq i+1$ 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 which says that any subset of a failure set is also a failure set, we obtain $\langle w, (\varphi, F)\rangle \in \mathcal{F}_\alpha(A)$.

Next, we show the full abstractness for $\mathcal{F}_\alpha$.

Theorem 19 *(Full abstractness for $\mathcal{F}_\alpha$)*
For any two agents A and B:

$$\mathcal{F}_\alpha(A) = \mathcal{F}_\alpha(B) \Leftrightarrow \mathcal{O}(C[A]) = \mathcal{O}(C[B]), \quad \text{for any context } C[\cdot]$$

Proof. The implication from left to right follows from the correctness of $\mathcal{F}_\alpha$ and its compositionality. For the reverse implication we proceed as follows. Suppose $\mathcal{F}_\alpha(A) \neq \mathcal{F}_\alpha(B)$ then w.l.o.g. there must exist $\langle w, \perp\rangle$ or $\langle w, (\varphi, F)\rangle \in \mathcal{F}_\alpha(A) - \mathcal{F}_\alpha(B)$, for some *finite* set F (according to the compactness property established in Theorem 18).

The idea is then to define a context $C[\cdot]$ such that there exists a constraint ψ with $\psi \in \mathcal{O}(C[A])$ and $\psi \notin \mathcal{O}(C[B])$. In order to achieve this, we define the complement $\widetilde{l}$ of a communication action l as follows: $\widetilde{c!\varphi} = c?\varphi$ and $\widetilde{c?\varphi} = (\mathsf{tell}(\varphi) \cdot c!\varphi)$, for all c and φ. Let w be given by $l_1 l_2 \cdots l_n$.

First, we consider the case $\langle w, \perp\rangle$. Consider the context $C[\cdot]$ that is defined by:

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

It is easy to see that $\mathcal{O}(C[A])$ contains all constraints. We claim that this does not hold for $\mathcal{O}(C[B])$. For, suppose $\mathcal{O}(C[B])$ contains all constraints then there are two possibilities. It could be the case that $B \overset{u}{\Longrightarrow} B' \overset{\tau}{\Longrightarrow} \Omega$, for some prefix u of w (modulo asking more and telling less) or it could be the case that $B \overset{u}{\Longrightarrow} B'$ with $\mathit{store}(B') = \mathit{false}$ for some prefix u of w. Then by the definition of $\mathcal{F}_\alpha$ in which divergence and inconsistency gives rise to chaos, we conclude $\langle u, \bot \rangle \in \mathcal{F}_\alpha(A)$ and hence, as u is a prefix of w we have: $\langle w, \bot \rangle \in \mathcal{F}_\alpha(A)$. This yields a contradiction.

Next, we consider the case $\langle w, (\varphi, F) \rangle$ (where $\varphi \neq \mathit{false}$). Consider the following context $C[\cdot]$:

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

where ok_1 and ok_2 denote 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. 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 $(\varphi \sqcup ok_1) \in \mathcal{O}(C[A])$.

We claim that $(\varphi \sqcup ok_1) \notin \mathcal{O}(C[B])$. For, otherwise it would be the case that $B \overset{u}{\Longrightarrow} B'$ 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 $\mathcal{F}_\alpha$ which says that any subset of a failure set is also a failure set, we conclude $\langle w, (\varphi, F) \rangle \in \mathcal{F}_\alpha(B)$. This yields a contradiction and hence we obtain $\mathcal{O}(C[A]) \neq \mathcal{O}(C[B])$.

5 Conclusions and Future Research

In this paper, we have outlined a concurrent programming language for multi-agent systems that concentrates on the information-processing aspects of agents. The language is given a compositional semantics, 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 our language which incorporates agent *signatures* such that communication of information additionally involves the *translation* of information from the signature of the sender to that of the receiving agent, as outlined in [6]. Furthermore, we aim to study the incorporation of non-monotonically increasing information stores as described in [7].

References

1. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

2. F. van Breughel. Failures, finiteness and full abstraction. In S. Brookes and M. Mislove, editors, *Proceedings of the Thirteenth Conference on the Mathematical Foundations of Programming Semantics*, volume 6 of Electronic Notes in Theoretical Computer Science. Elsevier, 1997.
3. L. Brim, D. Gilbert, J.-M. Jacquet, and M. Křetínský. A process algebra for synchronous concurrent constraint programming. In M. Hanus and M. Rodriquez-Artalejo, editors, *Proceedings of the fifth Conference on Algebraic and Logic Programming*, volume 1139 of *LNCS*, pages 165–178. Springer-Verlag, 1996.
4. S.D. Brookes, C.A.R. Hoare, and W. Roscoe. A theory of communicating sequential processes. *Journal of ACM*, 31:499–560, 1984.
5. R.M. van Eijk, F.S. de Boer, W. van der Hoek, and J.-J.Ch. Meyer. A language for modular information-passing agents. In K. R. Apt, editor, *CWI Quarterly, Special issue on Constraint Programming*, volume 11, pages 273–297. CWI, Amsterdam, 1998.
6. R.M. van Eijk, F.S. de Boer, W. van der Hoek, and J.-J.Ch. Meyer. Systems of communicating agents. In Henri Prade, editor, *Proceedings of the 13th biennial European Conference on Artificial Intelligence (ECAI-98)*, pages 293–297, Brighton, UK, 1998. John Wiley & Sons, Ltd.
7. R.M. van Eijk, F.S. de Boer, W. van der Hoek, and J.-J.Ch. Meyer. Information-passing and belief revision in multi-agent systems. In J. P. M. Müller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V — Proceedings of 5th International Workshop on Agent Theories, Architectures, and Languages (ATAL'98)*, volume 1555 of *Lecture Notes in Artificial Intelligence*, pages 29–45. Springer-Verlag, Heidelberg, 1999.
8. T. Finin, D. McKay, R. Fritzson, and R. McEntire. KQML: An Information and Knowledge Exchange Protocol. In Kazuhiro Fuchi and Toshio Yokoi, editors, *Knowledge Building and Knowledge Sharing*. Ohmsha and IOS Press, 1994.
9. J.Y. Halpern and Y. Moses. A guide to the completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.
10. L. Henkin, J.D. Monk, and A. Tarski. *Cylindric Algebras (Part I)*. North-Holland Publishing, Amsterdam, 1971.
11. C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.
12. Jean-Hugues Réty. *Langages concurrents avec contraintes, communication par messages et distribution*. PhD thesis, University of Orleans, France, 1997.
13. V.A. Saraswat. *Concurrent Constraint Programming*. The MIT Press, Cambridge, Massachusetts, 1993.
14. M. Wooldridge. Verifiable semantics for agent communication languages. In *Proceedings 3rd International Conference on Multi-Agent Systems (ICMAS'98)*, pages 349–356, Los Alamitos, California, 1998. IEEE Computer Society.
15. M. Wooldridge and N. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.