Electronic Notes in Theoretical Computer Science 68 No. 3 (2003)
URL: `http://www.elsevier.nl/locate/entcs/volume68.html` 26 pages

On Modeling Coordination via Asynchronous Communication and Enhanced Matching

Antonio Brogi [1]

Dipartimento di Informatica,
Università di Pisa
Pisa, Italy

Jean-Marie Jacquet [2]

Institute of Informatics
University of Namur
Namur, Belgium

Isabelle Linden [3]

Institute of Informatics
University of Namur
Namur, Belgium

Abstract

The paper proposes a theoretical study of a coordination language embodying Linda's asynchronous communication primitive with a refined matching mechanism based on pairs composed of attribute names associated with their values. Computations in this language are described by means of an operational semantics, reporting the whole traces of executions. The non-compositionality of this intuitive operational semantics motivates the design of a compositional and fully abstract denotational semantics, which is then exploited for studying program equivalence in this setting.

1 Introduction

Modern computer systems consist of large numbers of software components that interact one another. The process of software construction is more and

[1] Email: `brogi@di.unipi.it`
[2] Email: `jmj@info.fundp.ac.be`
[3] Email: `ili@info.fundp.ac.be`

more centered on the composition of generic existing packages to construct complex systems. Moreover, the rapid expansion of computer networks is highlighting the need for integrating and coordinating hetereogeneous components, which rely on different computational models and are physically distributed on the net.

Carrero and Gelernter [4] first pointed out the relevance of defining *coordination* models and languages for combining separate computational activities into asynchronous ensembles and for supporting communication among them. Linda [3] was the first coordination language, presented as a set of inter-agent communication primitives which can be virtually added to any programming language.

The interest for coordination models and languages is rapidly growing, and several coordination-based systems are under development or already available [5]. The study of models and languages for coordinating separate activities of software components is therefore of primary importance in this scenario. The scope of this paper is to contribute to setting the theoretical foundations of the growing field of coordination languages.

In this paper we will consider a simple language $\mathcal{L}_\Psi$ that embodies the essential features of coordination languages based on generative communication à la Linda [3]. The language includes Linda's `out`, `in`, and `rd` primitives for adding, deleting and checking the presence of a tuple in a shared dataspace. Furthermore, the matching mechanism of Linda is extended to allow tuples to be added with partial information and selected on partial information. Following the concurrency tradition, the language also includes sequential and parallel composition operators, as well as a choice operator in the style of CCS [8].

We will first describe the operational semantics of the language in the SOS style [10]. We will consider as observable behaviour of programs, for all possible computations, the sequences of states generated by computations. Each such state is composed of the contents of the shared dataspace and of the values assigned to variables. We will then define a denotational semantics and study the properties of compositionality and full abstraction with respect to the operational model of the language.

The definition of the denotational model is inspired by the denotational models for *nonuniform* concurrent languages proposed by Horita, de Bakker and Rutten in [7]. However, whereas in [7] the state of the computation is represented by the values of individual variables and is changed by assigning variables, in our setting the state of the computation is represented both by the content of the shared dataspace, which is changed by `out` and `in` primitives, and by the values of communication variables, which are affected by the `out`, `in`, and `rd` primitives. In both cases the meaning of a process needs to represent the possible interactions between the process itself and the environment. Simply recording the initial and final states of computation sequences does not provide a compositional description of processes. We will therefore employ sequences of states which contain *gaps* between steps to represent possible interactions with the environment.

This paper is also an extension of previous work published as [1]. However, the usual Linda matching mechanism was employed in that paper. As will be appreciated by the reader, the introduction of a refined matching mechanisms leads to the introduction of communication variables modified by the communication primitives which open new technical problems.

The denotational model of the language is defined in a *compositional* way. First the denotation of atomic agents (viz., communication actions) is given. Then the denotation of sequential, parallel and choice compositions of agents is defined by means of homomorphic operations on the denotations of agents. As the denotational semantics preserves the observational equivalence of agents, we have that denotationally equivalent programs are indistinguishable, that is, they exhibit the same observable operational behaviour in any possible context. We then show that the obtained denotational model is also *fully abstract* with respect to the operational model of the language. Intuitively speaking, this means that operationally indistinguishable programs are equivalent in the chosen denotational semantics.

The properties of compositionality and full abstraction of the denotational semantics establish firm foundations for reasoning about programs and program transformations. Let A be an agent which is part of a larger system, or context, denoted by $\mathcal{C}[A]$. Suppose that A' is an alternative, possibly more efficient, version of A obtained for instance by applying some program transformation technique to A. If A' is denotationally equivalent to A, that is if $\mathcal{D}(A) = \mathcal{D}(A')$, then the property of compositionality ensures that the substitution of A' for A does not affect the observable behavior of the whole system, that is $\mathcal{O}(\mathcal{C}[A]) = \mathcal{O}(\mathcal{C}[A'])$. While compositionality ensures that any pair of denotationally equivalent agents can be substituted one another without affecting the observational behavior of a system, full abstraction establishes that *only* denotationally equivalent programs satisfy such a property.

The paper is organized as follows. The language $\mathcal{L}_\Psi$ is introduced in Section 2, where syntax and operational semantics of the language are presented. A fully abstract denotational semantics for $\mathcal{L}_\Psi$ is defined in Section 3, while Section 4 contains some concluding remarks.

2 The $\mathcal{L}_\Psi$ language

2.1 Syntax

We shall consider a simple language $\mathcal{L}_\Psi$ embodying the basic Linda's `out`, `in`, and `rd` primitives, for putting a tuple on a shared space, getting it and checking its presence, respectively. $\mathcal{L}_\Psi$ also includes sequential and parallel composition operators as well as a choice operator in the style of CCS [8]. However, for simplicity purposes, only finite processes are treated here, under the observation that infinite processes can be handled by extending the results of this paper in the classical way, exemplified, for instance, in [7].

Special care is taken for the matching mechanism. We shall extend it so that tuples may specify partial information only. Technically, this leads to the introduction of Ψ-terms, defined as follows.

Definition 2.1 Let *Scvar* be a denumerably infinite set of communication variables and *Sf* a denumerably infinite set of functor names, each one coupled to an arity. Such an association is typically written as f/n where f is the functor name and n is its associated arity. Assume that the sets *Scvar* and *Sf* are disjoint.

As usual, functors of arity 0 are called constants. Their set is subsequently denoted as *Sconst*.

Definition 2.2 A Ψ-term is a construct of the form

$$f(item_1 = value_1, \cdots, item_m = value_m)$$

where

- f/n is a functor such that $m \leq n$
- the $item_i$'s are distinct constants
- $value_i$ denotes an integer, a string of characters, a Ψ-term or a communication variable
- any communication variable appears at most once in the Ψ-term.

A Ψ-term is said to be closed if it contains no communication variable. The sets of Ψ-terms is subsequently referred to as *Spterm*. The set of closed Ψ-terms is denoted as *Scpterm* in the following.

Matching will lead us subsequently to compare two Ψ-terms. To that end, we introduce the notion of correspondence.

Definition 2.3 Let $\Psi_1 = f(t_1 = v_1, \cdots, t_n = v_n)$ and $\Psi_2 = f'(t'_1 = v'_1, \cdots, t'_m = v'_m)$ be two Ψ-terms. We say that Ψ_1 corresponds to Ψ_2 iff

(i) f and f' are identical functors with same arities;

(ii) for any i such that v_i is an integer or a string of characters, if $t_i = t'_j$ then $v_i = v'_j$;

(iii) for any i such that v_i is a Ψ-term, if $t_i = t'_j$ then v_i corresponds to v'_j.

Note that, when Ψ_2 is closed, the correspondence of Ψ_1 with respect to Ψ_2 amounts to the existence of a set of values for the communication variables of Ψ_1 such that the two Ψ-terms become identical. When this is the case, this set of values for communication variables is subsequently denoted by $\Psi_1 \lhd \Psi_2$.

Definition 2.4 We shall denote by *Sbind* the set of all the possibly partial bindings of values to variables, namely the set of all the functions θ of the form

$$\theta : \mathit{Scvar} \rightarrow (N \cup \mathit{Sstring} \cup \mathit{Scpterm} \cup \{\perp\})$$

where N represents the set of integers, *Sstring* the set of string of characters and $\perp$ a special value indicating an undefined value. By abuse of notation, we shall denote by $\perp$ the binding which consists of leaving all the variables undefined.

A partial order can be defined on the set *Sbind* in the following way.

Definition 2.5 For any pair θ, φ of elements of *Sbind*, define $\theta \langle \varphi$ iff, for any variable x, one has $\theta(x) = \perp$ whenever $\varphi(x) = \perp$.

Moreover composition can be defined on *Sbind* as follows.

Definition 2.6 For any θ, φ of elements of *Sbind*, define $\theta\varphi$ as the following function of $\mathit{Scvar} \rightarrow (N \cup \mathit{Sstring} \cup \mathit{Scpterm} \cup \{\perp\})$: for any variable x,

$$\theta\mu(x) = \begin{cases} \mu(x) & \text{si } \mu(x) \neq \perp \\ \theta(x) & \text{si } \mu(x) = \perp \end{cases}$$

We are now in a position to define the language $\mathcal{L}_\Psi$. According to the philosophy of coordination languages, the $\mathcal{L}_\Psi$ language foccusses on interaction and communication. Other languages have to be used to specify the computations. Moreover, a means must be provided to interface the two aspects: interactions/communications and computations. This is subsequently achieved by communication variables used both in the communication primitives and the language used to code the computations. Hence, we shall subsequently assume the existence of a set of instructions *Sinstr* of some programming language. These instructions can be of any kind and of any paradigm (functional, imperative, logic, object-oriented); the only constraint is that they cannot directly interact with the shared space and cannot modify values assigned to variables.

The language $\mathcal{L}_\Psi$ is formally defined by the following grammar.

Definition 2.7 Let Ψ be a Ψ-term, Ψ_c be a closed Ψ-term and I denotes an instruction of *Sinstr*. The language $\mathcal{L}_\Psi$ is the set of agents A defined by the following grammar. On the point of terminology, the constructs c are subsequently called communication actions.

$$c ::= \mathit{tell}(\psi_c) \mid \mathit{nask}(\psi) \mid \mathit{ask}(\psi) \mid \mathit{get}(\psi) \mid I$$
$$A ::= c \mid A \ ; \ A \mid A \,||\, A \mid A \ + \ A$$

2.2 Operational semantics

2.2.1 Configurations.

$\mathcal{L}_\Psi$ computations may be modelled by the following transition system written in Plotkin's style. Following intuition, most of configurations consist of an agent to be solved together with a description of the contents of the shared space and of the bindings of the communication variables.

To easily express termination, we shall introduce particular configurations composed of a special terminating symbol E together with a shared space and a binding for communication variables. For uniformity purposes, we shall abuse language and qualify E as an agent. However, to meet the intuition, we shall always rewrite agents of the form $(E\ ;\ A)$, $(E \mid\mid A)$, and $(A \mid\mid E)$ as A. This is technically achieved by defining the extended set of agents as follows. In particular, simplifications are operated by imposing a bimonoid structure.

Definition 2.8 Define the extended set of agents *Seagent* by the following grammar

$$Ae ::= E \mid c \mid A\ ;\ A \mid A \mid\mid A \mid A\ +\ A$$

Moreover, we shall subsequently assert that the structure $(Seagent, E, \ ;\ , \mid\mid\ )$ is a bimonoid and simplify elements of $\mathcal{L}_\Psi$ accordingly.

Definition 2.9 Define *Sstore* as the set of finite multisets of closed Ψ-terms. Moreover, define the set of situations *Ssit* as the set $Sbind \times Sstore$.

Definition 2.10 Define the set of configurations *Sconf* as $Seagent \times Ssit$. Configurations are denoted as $\langle A \mid (\theta, \sigma)\rangle$, where A is an (extended) agent and (θ, σ) is a situation.

2.2.2 Transition rules.

The transition rules defining the operational semantics of the language are reported in Figure 1. They are based on the consideration of two points.

On the one hand, instructions of *Sinstr* may consult and use the values of communication variables. However, it is possible that their actual execution depends on these values. For instance, printing variables without values may be impossible as is the division by a variable whose value is 0. The predicate *executable* is introduced to model the possible non execution of instructions. For any $I \in Sinstr$ and any $\theta \in Sbind$, $executable(I, \theta)$ is true iff I can be executed on θ. Note that, in view of the coordination philosophy, the execution of I cannot depend on the contents of the shared space since it cannot access to it. Moreover, the result of the execution cannot modify the contents of this shared space and of the bindings reported by θ. Hence, the execution of any instruction is of no consequence for our semantics.

On the other hand, in order to keep the communication between agents in a pure form through the shared space and thus in order to avoid a side communication of agents by means of communication variables, we impose that agents obtained by parallel composition are formed from components which do not share communication variables. The following Var function helps to grasp this idea technically.

Definition 2.11 Define the function $Var : \mathcal{L}_\psi \rightarrow \mathcal{P}(Scvar)$ inductively as follows:

$$
\begin{aligned}
Var(tell(\Psi_c)) &= \emptyset \\
Var(nask(\Psi)) &= \emptyset \\
Var(I) &= \emptyset \\
Var(ask(\Psi)) &= \{v : v \text{ variable appearing in } \Psi\} \\
Var(get(\Psi)) &= \{v : v \text{ variable appearing in } \Psi\} \\
Var(A;B) &= Var(A) \cup Var(B) \\
Var(A+B) &= Var(A) \cup Var(B) \\
Var(A \,||\, B) &= \begin{cases} \emptyset & \text{if } Var(A) \cap Var(B) \neq \emptyset \\ Var(A) \cup Var(B) & \text{if } Var(A) \cap Var(B) = \emptyset. \end{cases}
\end{aligned}
$$

Rule **(T)** states that an atomic agent $tell(\Psi_c)$ can be executed in any situation (θ, σ), and that its execution results in adding the closed Ψ-term Ψ_c to the store σ. Rule **(A)** states that an atomic agent $ask(\Psi)$ can be executed in a situation (θ, σ) provided that the store σ contains a Ψ-term Ψ_c that corresponds to Ψ. In that case, new values may be computed for the communication variables, which results in updating the binding θ to $\theta\mu$. Rule **(G)** is similar except that Ψ_c is removed from the store. Rule **(N)** is dual: it succeeds if such a corresponding Ψ-term cannot be found on the current store. Note that no update results from its execution. Rule **(I)** specifies that an instruction I is computed if it is executable and, in this case, without affecting the current situation. Finally, rules **(S)**, **(P)**, and **(C)** describe the operational meaning of sequential, parallel and choice operators in the standard way [8].

2.2.3 Observables.

A reasonable notion of observables consists of reporting, for all possible computations, the sequence of situations the computations produce. It is defined subsequently as the semantics $\mathcal{O}_h$ to stress this notion of history of situation. Another reasonable notion of observable would consists in focusing on the results of the computations only. It is however considered to be outside the scope of this paper and will be studied in future work.

We will use the following notation to represent sequences of elements and their concatenation. Given a set A, the set of finite sequences of elements of A is denoted by $A^{<\omega}$. The concatenation of two sequences q_1 and q_2 is denoted by $q_1.q_2$. Also if S_1 and S_2 are two sets of (finite) sequences we put:

$$S_1.S_2 \;=\; \{q_1.q_2 \mid q_1 \in S_1 \wedge q_2 \in S_2\}.$$

(T) $$\langle \mathit{tell}(\Psi_c) \mid (\theta,\sigma)\rangle \longrightarrow \langle E \mid (\theta, \sigma \cup \{\Psi_c\}\rangle$$

(A) $$\frac{\Psi \lhd \Psi_c = \mu}{\langle \mathit{ask}(\Psi) \mid (\theta, \sigma \cup \{\Psi_c\})\rangle \longrightarrow \langle E \mid (\theta\mu, \sigma \cup \{\Psi_c\})\rangle}$$

(N) $$\frac{\not\exists \Psi_c : \Psi_c \in \sigma, \Psi \text{ corresponds to } \Psi_c}{\langle \mathit{nask}(\Psi) \mid (\theta,\sigma)\rangle \longrightarrow \langle E \mid (\theta,\sigma)\rangle}$$

(G) $$\frac{\Psi \lhd \Psi_c = \mu}{\langle \mathit{get}(\Psi) \mid (\theta, \sigma \cup \{\Psi_c\})\rangle \longrightarrow \langle E \mid (\theta\mu, \sigma)\rangle}$$

(I) $$\frac{\mathit{executable}(I,\theta)}{\langle I \mid (\theta,\sigma)\rangle \longrightarrow \langle E \mid (\theta,\sigma)\rangle}$$

(S) $$\frac{\langle A \mid (\theta,\sigma)\rangle \longrightarrow \langle A' \mid (\theta',\sigma')\rangle}{\langle A\ ;\ B \mid (\theta,\sigma)\rangle \longrightarrow \langle A'\ ;\ B \mid (\theta',\sigma')\rangle}$$

(P) $$\frac{\begin{array}{c}\langle A \mid (\theta,\sigma)\rangle \longrightarrow \langle A' \mid (\theta',\sigma')\rangle \\ Var(A) \cap Var(B) = \emptyset\end{array}}{\begin{array}{c}\langle A \mid\mid B \mid (\theta,\sigma)\rangle \longrightarrow \langle A' \mid\mid B \mid (\theta',\sigma')\rangle \\ \langle B \mid\mid A \mid (\theta,\sigma)\rangle \longrightarrow \langle B \mid\mid A' \mid (\theta'\sigma')\rangle\end{array}}$$

(C) $$\frac{\langle A \mid (\theta,\sigma)\rangle \longrightarrow \langle A' \mid (\theta',\sigma')\rangle}{\begin{array}{c}\langle A\ +\ B \mid (\theta,\sigma)\rangle \longrightarrow \langle A' \mid (\theta',\sigma')\rangle \\ \langle B\ +\ A \mid (\theta,\sigma)\rangle \longrightarrow \langle A' \mid (\theta',\sigma')\rangle\end{array}}$$

Fig. 1. The transition rules.

Definition 2.12

(i) Let δ^+ and δ^- be two fresh symbols denoting respectively success and failure. Define the set of histories *Shist* as the set of finite sequences ending by one termination mark: $Ssit^{<\omega}.\{\delta^+, \delta^-\}$.

(ii) Define the *"history semantics"* $\mathcal{O}_h : \mathcal{L}_\Psi \to \mathcal{P}(Shist)$ as the following function: For any agent A,

$$\begin{aligned}
&\mathcal{O}_h(A) = \\
&\{(\theta_0,\sigma_0).\cdots.(\theta_n,\sigma_n).\delta^+ : \langle A_0 \mid (\theta_0,\sigma_0)\rangle \to \cdots \langle A_n \mid (\theta_n,\sigma_n)\rangle, \\
&\qquad\qquad A_0 = A, \theta_0 = \perp, \sigma_0 = \emptyset, A_n = E, n \geq 0\} \\
&\cup \\
&\{(\theta_0,\sigma_0).\cdots.(\theta_n,\sigma_n).\delta^- : \langle A_0 \mid (\theta_0,\sigma_0)\rangle \to \cdots \langle A_n \mid (\theta_n,\sigma_n)\rangle \not\to, \\
&\qquad\qquad A_0 = A, \theta_0 = \perp, \sigma_0 = \emptyset, A_n \neq E, n \geq 0\}
\end{aligned}$$

2.2.4 On compositionality.
It is here worth noting that the operational semantics $\mathcal{O}_h$ is not compositional. For instance, taking Ψ_1 and Ψ_2 two distinct and closed Ψ-terms, we have that:

$$\mathcal{O}_h(get(\Psi_1)) \;=\; \mathcal{O}_h(get(\Psi_2)) \;=\; \{(\perp,\emptyset).\delta^-\}$$

whereas

$$\begin{aligned}
&\mathcal{O}_h(tell(\Psi_1) \mid\mid get(\Psi_1)) = \{(\perp,\emptyset).(\perp,\{\Psi_1\}).(\perp,\emptyset).\delta^+\} \\
&\mathcal{O}_h(tell(\Psi_1) \mid\mid get(\Psi_2)) = \{(\perp,\emptyset).(\perp,\{\Psi_2\}).\delta^-\}
\end{aligned}$$

Hence, $\mathcal{O}_h$ is not compositional.

The purpose of the next section is precisely to define a compositional semantics for $\mathcal{L}_\Psi$ which is correct with respect to the history operational semantics but which also contains a "minimal" amount of information to be compositional. In other words, we shall try to define a fully abstract (compositional) semantics.

3 Fully abstract compositional semantics for histories

There are two main reasons why the history semantics $\mathcal{O}_h$ is not compositional. First, the execution of a computation step produces a store and a binding which are not necessarily empty. A compositional semantics should therefore account for initial stores of any content and bindings of any values. Second, as shown from the transition system, the computation of the agent $A \mid\mid B$ amounts to interleaving execution steps of A and B. A compositional semantics should thus allow for transition steps made by the environment.

Following [7], we shall model transition steps in the form of pairs of input and output stores and take as semantic domain sets of sequences of such pairs. These sequences possibly contain gaps, accounting for actions of the environment. Moreover, they will start in any store, allowing previous steps to result in a possibly non-empty store.

3.1 Notation

Before proceeding, it is convenient to introduce some notations.

Definition 3.1 Define *Shhist* as the following set

$$\begin{aligned} \mathit{Shhist} = \{h = ((\theta_1,\sigma_1),(\kappa_1,\tau_1)).\cdots.((\theta_{n-1},\sigma_{n-1}),(\kappa_{n-1},\tau_{n-1})).((\theta_n,\sigma_n),\delta) :& \\ \theta_i,\kappa_i \in \mathit{Saffect}, \sigma_i,\tau_i \in \mathit{Setat} \text{ for } i=1,\ldots,n, \delta \in \{\delta^+,\delta^-\}& \\ \text{and } \kappa_i \prec \theta_{i+1} \text{ for } i=1,\ldots,n-1\}& \end{aligned}$$

Notation 3.2

(i) *Let S be a set of histories of Shhist and p be a sequence of $(Ssit \times Ssit)^{<\omega}$. Then*

$$S[p] = \{h : p.h \in S\}$$

(ii) *Let S be a set of histories of Shhist. Then,*

$$\begin{aligned} S^a &= \{h : h = (s,t).h' \in S\} \\ S^+ &= \{h : h = (s,\delta^+) \in S\} \\ S^- &= \{h : h = (s,\delta^-) \in S\} \end{aligned}$$

(iii) *Let h be an history of Shhist. Then*

$$\mathit{init}(h) = \begin{cases} s & \text{if } h = (s,t).h' \\ s & \text{if } h = (s,\delta), \delta \in \{\delta^+,\delta^-\} \end{cases}$$

(iv) *For $n \geq 0$ and $\delta \in \{\delta^+,\delta^-\}$, let*

$$h = ((\theta_1,\sigma_1),(\kappa_1,\tau_1)).\cdots.((\theta_{n-1},\sigma_{n-1}),(\kappa_{n-1},\tau_{n-1})).((\theta_n,\sigma_n),\delta)$$

be an history of Shhist. Then

$$\mathit{diff}(h) = (\sigma_1 \setminus \tau_1) \cup (\tau_1 \setminus \sigma_1) \cup \cdots \cup (\sigma_{n-1} \setminus \tau_{n-1}) \cup (\tau_{n-1} \setminus \sigma_{n-1})$$

where $\cup$ and $\setminus$ denote, respectively, multiset union and difference. Abusing notations, we shall lift diff to sets of histories in the natural way: For any set S of histories of Shhist,

$$\mathit{diff}(S) = \bigcup\{\mathit{diff}(h) : h \in S\}$$

(v) *Let $h = ((\theta_1,\sigma_1),(\kappa_1,\tau_1)).\cdots.((\theta_{n-1},\sigma_{n-1}),(\kappa_{n-1},\tau_{n-1})).((\theta_n,\sigma_n),\delta)$ be an history of Shhist. Then*

$$\begin{aligned} \mathit{Vars}(h) &= \{x \in \mathit{Svar} : \exists i \in \{1,\ldots,n-1\} : \kappa_i(x) \neq \theta_i(x)\} \\ \mathit{Ext}(h) &= \{x \in \mathit{Svar} : \exists i \in \{1,\ldots,n-1\} : \theta_{i+1}(x) \neq \kappa_i(x)\} \end{aligned}$$

By extension, we shall lift these notations to sets of histories: for any $S \subseteq$ Shhist,

$$\begin{aligned} \mathit{Var}(S) &= \bigcup_{h \in S} \mathit{Vars}(h) \\ \mathit{Ext}(S) &= \bigcup_{h \in S} \mathit{Ext}(h) \end{aligned}$$

Definition 3.3 An history $h \in Shhist$ is *continuous* iff it has the form

$$(s_0, s_1).(s_1, s_2). \cdots .(s_{n-1}, s_n).(s_n, \delta)$$

with $\delta \in \{\delta^+, \delta^-\}$. In that case, $\overline{h}$ denotes the following sequence of stores

$$\overline{h} = s_0.s_1. \cdots .s_n.\delta$$

3.2 Denotational semantics

Defining a compositional semantics consists of, on the one hand, specifying the meaning of elementary statements and, on the other hand, providing an operator at the semantic level for each syntactic operator. We start with this second task in the following section. A compositional semantics, called denotational in view of the compositionality property, is defined next. It is then proved correct with respect to the history operational semantics $\mathcal{O}_h$, and finally it is established to be fully abstract.

3.2.1 Property.

Before going into the details of the definitions, it is worth observing that the variable non-sharing constraint we have requested from the parallel composition of agents imposes that an agent cannot expect from its environment to modify the values of variables that it modifies. Technically speaking, for any agent A and history h of the denotational semantics $\mathcal{D}_h(A)$ of A, the following equality should thus hold:

$$Vars(h) \cap Ext(h) = \emptyset.$$

This property is even more general: none of the histories associated with an agent may expect the environment to modify a variable modified by the history. In other terms,

$$Vars(\mathcal{D}_h(A)) \cap Ext(\mathcal{D}_h(A)) = \emptyset.$$

3.2.2 Semantic operators.

There are three syntactic operators to combine elementary agents: Sequential composition, parallel composition, and choice. Let us examine each of them in turn.

Sequential composition. Since semantic histories may include gaps and begin with any input store and any binding, composing the meaning of two agents which are sequentially composed amounts to concatenating their histories. However, some of the resulting histories may violate the values increasing condition $\kappa_i \prec \theta_{i+1}$ of definition 3.1. An intersection with *Shhist* is introduced for that purpose. Moreover, care is taken in the expected manner for the termination marks. This leads to the following operator.

Definition 3.4 Define $\tilde{;} : \mathcal{P}(Shhist) \times \mathcal{P}(Shhist) \rightarrow \mathcal{P}(Shhist)$ as the following function: For any subset S_1, S_2 of *Shhist*,

$$\begin{aligned} S_1 \tilde{;} S_2 = {} & \{h_1.h_2 : h_1.(\sigma, \delta^+) \in S_1, h_2 \in S_2, \\ & \qquad Ext(h) \cap (Vars(S_1) \cup Vars(S_2)) = \emptyset\} \cap Shhist \\ & \cup \{h_1.(\sigma, \delta^-) : h_1.(\sigma, \delta^-) \in S_1, \\ & \qquad Ext(h) \cap (Vars(S_1) \cup Vars(S_2)) = \emptyset\} \end{aligned}$$

Parallel composition. Parallel composition is modelled in an interleaving fashion. Consequently, composing in parallel two semantic histories amounts to take their merge. Again care has to be taken to termination marks and on the value increasing constraint, as formalized below.

Definition 3.5 Define the parallel composition of two histories as the function $\widetilde{\|}_h : Shhist \times Shhist \rightarrow \mathcal{P}(Shhist)$ defined inductively by the following equalities, where δ stands either for δ^+ or δ^-.

$$\begin{aligned} (s_1, t_1).h_1 \ \widetilde{\|}_h \ (s_2, t_2).h_2 = {} & (\{(s_1, t_1).h : h \in h_1 \ \widetilde{\|}_h \ (s_2, t_2).h_2\} \cap Shhist) \\ & \cup (\{(s_2, t_2).h : h \in (s_1, t_1).h_1 \ \widetilde{\|}_h \ h_2\} \cap Shhist) \end{aligned}$$

$$\begin{aligned} (s_1, t_1).h_1 \ \widetilde{\|}_h \ (s_2, \delta_2) & = (s_2, \delta_2) \ \widetilde{\|}_h \ (s_1, t_1).h_1 \\ & = \{(s_1, t_1).h : h \in h_1 \ \widetilde{\|}_h \ (s_2, \delta_2)\} \cap Shhist \end{aligned}$$

$$(s_1, \delta_1) \ \widetilde{\|}_h \ (s_2, \delta_2) = \begin{cases} \{(s_1, \delta^+)\}, \text{ if } s_1 = s_2 \text{ and } \delta_1 = \delta_2 = \delta^+ \\ \{(s_1, \delta^-)\}, \text{ if } 1) \ s_1 = s_2 \text{ and} \\ \qquad\qquad 2) \ \delta_1 = \delta^- \text{ or } \delta_2 = \delta^- \\ \emptyset, \text{ if } s_1 \neq s_2 \end{cases}$$

Definition 3.6 Define the parallel composition of two sets of histories as the natural lifting of function $\widetilde{\|}_h$, namely as the function $\widetilde{\|} : \mathcal{P}(Shhist) \times \mathcal{P}(Shhist) \rightarrow \mathcal{P}(Shhist)$ defined as follows: for any subset S_1, S_2 of *Shhist*,

$$S_1 \ \widetilde{\|} \ S_2 = \begin{cases} \bigcup \{h_1 \ \widetilde{\|}_h \ h_2 : h_1 \in S_1, h_2 \in S_2, \\ \qquad Ext(h) \cap (Vars(S_1) \cup Vars(S_2)) = \emptyset\}, \\ \qquad \text{if } Vars(S_1) \cap Vars(S_2) = \emptyset \\ \emptyset, \text{ otherwise} \end{cases}$$

Note that the composition of histories has been made possible only if they do not affect common variables.

Choice. Choice is modelled as an internal choice, namely an agent formed from the choice of two agents can proceed as any of its components. As before care has to be taken for termination marks. The composed agent fails if the two components do so; it succeeds if at least one of the two components does.

Definition 3.7 Define $\tilde{+}$: $\mathcal{P}(Shhist) \times \mathcal{P}(Shhist) \to \mathcal{P}(Shhist)$ as the following function: for any subset S_1, S_2 of *Shhist*,

$$\begin{aligned} S_1 \tilde{+} S_2 = \{ & h \in S_1^a \cup S_2^a : Ext(h) \cap (Vars(S_1) \cup Vars(S_2)) = \emptyset\} \\ & \cup S_1^+ \cup S_2^+ \cup (S_1^- \cap S_2^-) \end{aligned}$$

3.2.3 Definition.
Given the operators $\tilde{;}$, $\tilde{\|}$, and $\tilde{+}$, defining the denotational semantics amounts to specifying the semantics of the basic constructs *tell*, *ask*, *nask*, *get* and of the instructions, typically denoted *instr*. This is achieved according to the intuition given by their operational behavior.

Definition 3.8 Define the denotational semantics as the following function $\mathcal{D}_h$: $Sagent \to \mathcal{P}(Shhist)$: for any Ψ-term Ψ, for any instruction *instr*, for any agents A_1, A_2,

$$\begin{aligned} \mathcal{D}_h(tell(\psi)) = \{ & ((\theta,\sigma),(\theta,\sigma \cup \{\psi\})).((\kappa,\tau),\delta^+) : \sigma,\tau \in Sstore, \\ & \theta,\kappa \in Sbin, \theta \prec \kappa\} \end{aligned}$$

$$\begin{aligned} \mathcal{D}_h(ask(\psi)) = & \{((\theta,\sigma),(\theta\mu,\sigma)).((\kappa,\tau),\delta^+) : \sigma,\tau \in Sstore, \psi_c \in \sigma, \\ & \psi \lhd \psi_c = \mu, \theta,\kappa \in Sbind, \theta\mu \prec \kappa, \kappa\mu = \kappa\} \\ & \cup \{((\theta,\sigma),\delta^-) : \sigma \in Sstore, \theta \in Sbind, \\ & \nexists \psi_c \in \sigma : \psi \text{ corresponds to } \psi_c\} \end{aligned}$$

$$\begin{aligned} \mathcal{D}_h(nask(\psi)) = & \{((\theta,\sigma),(\theta,\sigma)).((\kappa,\tau),\delta^+) : \sigma,\tau \in Sbind, \\ & \nexists \psi_c \in \sigma : \psi \text{ corresponds to } \psi_c, \theta,\kappa \in Saffect, \theta \prec \kappa\} \\ & \cup \{((\theta,\sigma),\delta^-) : \sigma \in Setat, \theta \in Sbind, \\ & \exists \psi_c \in \sigma : \psi \text{ corresponds to } \psi_c\} \end{aligned}$$

$$\begin{aligned} \mathcal{D}_h(get(\psi)) = & \{((\theta,\sigma),(\theta\mu,\sigma \setminus \{\psi_c\})).((\kappa,\tau),\delta^+) : \sigma,\tau \in Sstore, \\ & \psi_c \in \sigma, \psi \lhd \psi_c = \mu\theta, \kappa \in Sbind, \theta\mu \prec \kappa, \kappa\mu = \kappa\} \\ & \cup \{((\theta,\sigma),\delta^-) : \sigma \in Sstore, \theta \in Sbind, \\ & \nexists \psi_c \in \sigma\psi \text{ corresponds to } \psi_c\} \end{aligned}$$

$$\begin{aligned}\mathcal{D}_h(instr) = \{((\theta,\sigma),(\theta,\sigma)).((\kappa,\tau),\delta^+) : \sigma,\tau \in Sstore, \theta,\kappa \in Sbind,\\ executable(instr,\theta), \theta \prec \kappa\}\\ \cup \{((\theta,\sigma),\delta^-) : \sigma \in Sstore, \neg executable(instr,\theta)\}\\ \mathcal{D}_h(A_1 \ ; \ A_2) = \mathcal{D}_h(A_1) \,\tilde{;}\, \mathcal{D}_h(A_2)\\ \mathcal{D}_h(A_1 \ || \ A_2) = \mathcal{D}_h(A_1) \,\widetilde{\|}\, \mathcal{D}_h(A_2)\\ \mathcal{D}_h(A_1 \ + \ A_2) = \mathcal{D}_h(A_1) \,\tilde{+}\, \mathcal{D}_h(A_2)\end{aligned}$$

3.2.4 Properties.
It is easy to observe that the semantics $\mathcal{D}_h$ is compositional by construction. It is also correct with respect to the semantics $\mathcal{O}_h$ in the sense that the latter can be obtained from $\mathcal{D}_h$. Indeed, it is sufficient to take the continous histories from $\mathcal{D}_h$ starting in the empty store and the empty binding to get those produced by $\mathcal{O}_h$.

Proposition 3.9 *Let $\alpha : \mathcal{P}(Shhist) \to \mathcal{P}(Shhist)$ be defined as follows: for any subset $S \subseteq Shhist$,*

$$\alpha(S) = \{\overline{h} : h \in S, h \ continuous, init(h) = (\bot, \emptyset)\}.$$

Then

$$\mathcal{O}_h = \alpha \circ \mathcal{D}_h.$$

Proof By structural reasoning. ■

Proposition 3.9 establishes that if two agents A_1 and A_2 have same denotation, viz. $\mathcal{D}_h(A_1) = \mathcal{D}_h(A_2)$, then they are indistinguishable in any context.

Note that, as a corollary of this proposition an operational history $s_1. \cdots .s_n.\delta$ corresponds biunivoquely to a continuous denotational history starting in the empty binding and empty store: $((\bot,\emptyset), s_1). \cdots .(s_n, \delta)$.

The denotational semantics can also be characterized in terms of the operational semantics as follows.

Proposition 3.10 *Extend the denotational semantics to the empty agent E as follows:*

$$\mathcal{D}_h(E) = \{(s,\delta^+) : s \in Ssit\}.$$

Let A be an agent and (θ,σ), (κ,τ) be situations such that $\mathcal{D}_h(A)[((\theta,\sigma),(\kappa,\tau))] \neq \emptyset$. Moreover, let B_1, ..., B_m be all the agents such that

$$\langle A \mid (\theta,\sigma)\rangle \to \langle B \mid (\kappa,\tau)\rangle$$

Then,

$$\mathcal{D}_h(A)[((\theta,\sigma),(\kappa,\tau))] = \mathcal{D}_h(B_1) \cup \cdots \cup \mathcal{D}_h(B_m).$$

Proof By structural reasoning. ■

Note that $\mathcal{D}_h(B_1) \cup \cdots \cup \mathcal{D}_h(B_m)$ is almost $\mathcal{D}_h(B_1 + \cdots + B_m)$. They actually differ by the treatment of immediately failing computations: All of them are registered in $\mathcal{D}_h(B_1) \cup \cdots \cup \mathcal{D}_h(B_m)$ while only those common to B_1, $\ldots$, B_m appear in the denotational semantics of $B_1 + \cdots + B_m$.

The next property to ask is whether $\mathcal{D}_h$ contains the least information necessary to be compositional and correct. That corresponds to a full abstraction result. This result is so involved that it deserves a complete section, which is done in the next section.

As a preliminary result, it is interesting to observe that for any agent A, the denotational semantics $\mathcal{D}_h(A)$ is *extensible* in the following sense.

Proposition 3.11 *For any agent A, any situations s, s_1, $\ldots$, s_n,*

(i) *if $\mathcal{D}_h(A) \neq \emptyset$, there is a continuous history in $\mathcal{D}_h(A)$ starting in s*

(ii) *if $\mathcal{D}_h(A)[(s_1, s_2). \cdots .(s_{n-1}, s_n)] \neq \emptyset$ then there is a continuous history in $\mathcal{D}_h(A)$ of the form $(s_1, s_2). \cdots .(s_{n-1}, s_n).h'$*

Proof By structural reasoning. ■

3.3 Full abstraction

3.3.1 Definitions.

Definition 3.12 Let $\Box$ be a fresh symbol. Define the set of contexts *Scontext* by the following rules, where t is a token, A is an agent and c denotes a communication action.

$$C ::= \Box \mid A \mid C \; ; \; A \mid A \; ; \; C \mid C \, || \, A \mid A \, || \, C \mid C \; + \; A \mid A \; + \; C$$

The application of a context C to an agent A is defined as the new agent obtained by replacing the place holder $\Box$ in C, if any, by A. This is subsequently denoted as $C[A]$.

Definition 3.13 The semantics $\mathcal{D}_h$ is fully abstract with respect to the semantics $\mathcal{O}_h$ iff the following property holds: for any agents A_1, A_2, the following assertions are equivalent

i) for any context C, $\mathcal{O}_h(C[A_1]) = \mathcal{O}_h(C[A_2])$;

ii) $\mathcal{D}_h(A_1) = \mathcal{D}_h(A_2)$.

3.3.2 Intuition.

The compositional property of $\mathcal{D}_h$ together with proposition 3.9 establish the implication $(ii) \Rightarrow (i)$. It thus remains to prove the converse $(i) \Rightarrow (ii)$. To that end, we shall proceed by contraposition. Given two agents A_1, A_2 such that

$$\mathcal{D}_h(A_1) \neq \mathcal{D}_h(A_2)$$

we shall construct a context C such that

$\mathcal{O}_h(C[A_1]) \neq \mathcal{O}_h(C[A_2])$

The two semantics reporting sets, the construction amounts to constructing from a denotational history h of one agent, say A_1, which is not in the denotation of the other A_2, a context C and an operational history of $C[A_1]$ not of $C[A_2]$. In view of the relation between $\mathcal{O}_h$ and $\mathcal{D}_h$ as shown by α in proposition 3.9, this amounts to establishing the existence of a continuous denotational history, starting in $(\perp, \emptyset)$, which is in $\mathcal{D}_h(C[A_1])$ and not in $\mathcal{D}_h(C[A_2])$. To that end, following [7], we shall construct from h a new history h' and an agent T such that h' is in the denotation of $A_1 \mid\mid T$ and not in the denotation of $A_2 \mid\mid T$.

The proof basically proceeds by induction on the length of h.

In the base case, h takes the form $((\theta, \sigma), \delta)$ with δ being either δ^+ or δ^-. The tester T then essentially constructs a continuous sequence yielding (θ, σ) from the initial situation $(\perp, \emptyset)$ in a way that, on the one hand, prevents A_1 and A_2 to do any intermediary step, and, on the other hand, forces A_1 and A_2 to do the last step $((\theta, \sigma), \delta)$. By hypothesis, this is possible for A_1 and not for A_2.

In the inductive case, h takes the form $((\theta, \sigma), (\kappa, \tau)).h^*$ for some history h^*. Two cases are possible: either there is no history starting by $((\theta, \sigma), (\kappa, \tau))$ in $\mathcal{D}_h(A_2)$ or those which start by $((\theta, \sigma), (\kappa, \tau))$ cannot end by h^*. In the first case, the proof proceeds as in the base case. In the second case, the proof uses induction. However, the induction should be applied for h^* in $\mathcal{D}_h(A_1)[((\theta, \sigma), (\kappa, \tau))]$ and not in $\mathcal{D}_h(A_2)[((\theta, \sigma), (\kappa, \tau))]$. As stated by proposition 3.10, these sets turned out to be basically but not exactly the denotations $\mathcal{D}_h(A_1')$ and $\mathcal{D}_h(A_2')$, of some agents A_1' and A_2'. We shall consequently generalize a bit the induction to sets of denotational histories. This extension being discarded here for the sake of simplicity, we thus apply the induction hypothesis for h^*, A_1' and A_2'. It points out a tester T' and an history h'' which is in $\mathcal{D}_h(A_1' \mid\mid T')$ and not in $\mathcal{D}_h(A_2' \mid\mid T')$. From there we should construct a tester T and an history h''' in $\mathcal{D}_h(A_1 \mid\mid T)$ and not in $\mathcal{D}_h(A_2 \mid\mid T)$. Basically, the step $((\theta, \sigma), (\kappa, \tau))$ has to be done before h'' and since h''' needs to be continuous, h'' has to start in a possibly non empty situation. Hence, we have to generalize the theorem and construct in general from h an history h' which start in any binding and any initial store. Given this generalization, the tester T basically consists of first making the steps necessary to produce (θ, σ) from the given initial store, then of making an auxiliary transition from τ to some τ' chosen so as to ensure that A_1 and A_2 have to do the step $((\theta, \sigma), (\kappa, \tau))$, modify the bindings to the variables to make them identical to those of the initial situation of h'' and finally consists of T'.

3.3.3 Auxiliary concepts.

The above intuition points out two auxiliary tasks. The first one consists of making by an auxiliary agent the steps necessary to modify a binding θ in a binding $\kappa \succ \theta$ while preserving a store σ. This is the purpose of the agent

$B^W_{(\theta,\sigma)\to(\kappa,\sigma)}$. The second task is to produce a given target store τ from a given initial store σ while preserving a binding θ. These steps are subsequently achieved by means of the following agent $Ag^V_{(\theta,\sigma)\to(\theta,\tau)}$.

Definition 3.14 Let θ, κ be two bindings such that $\theta \prec \kappa$. Let σ be a store and W be a set of Ψ-terms. Then the agent $B^W_{(\theta,\sigma)\to(\kappa,\sigma)}$ is defined as follows. Let us denote by μ the binding defined by

$$\mu(x) = \begin{cases} \kappa(x) & \text{if } \kappa(x) \neq \theta(x) \\ \perp & \text{if } \kappa(x) = \theta(x). \end{cases}$$

Note that it is such that $\kappa = \theta\mu$. Moreover let us denote by $x_1, \ldots, x_n$ the variables such that $\mu(x_i) \neq \perp$. Let now f be a functor of arity n which appears in none of the Ψ-terms in σ, nor in W. Construct then the Ψ-terms

$$\Psi_c = f(t_1 = \mu(x_1), \cdots, t_n = \mu(x_n))$$
$$\Psi = f(t_1 = x_1, \cdots, t_n = x_n)$$

Define $B^W_{(\theta,\sigma)\to(\kappa,\sigma)}$ as the agent

$$\mathit{tell}(\Psi_c); \mathit{get}(\Psi).$$

Obviously, the history

$$((\theta,\sigma),(\theta,\sigma\cup\{\psi\})).((\theta,\sigma\cup\{\psi\}),(\theta\mu,\sigma)).((\theta\mu,\sigma),\delta^+)$$

is an history of $\mathcal{D}_h(B^W_{(\theta,\sigma)\to(\kappa,\sigma)})$. We shall denote it by $\Gamma^W_{(\theta,\sigma)\to(\kappa,\sigma)}$.

It is here worth stressing that we will not be able later to put the agent $B^W_{(\theta,\sigma)\to(\kappa,\sigma)}$ in parallel with other agents manipulating common variables. By doing so, we would then end up with an empty denotational semantics.

Definition 3.15 Let V be a finite set of Ψ-terms, σ and τ be two stores, and θ be a binding. Let

$$\sigma \setminus \tau = \{g_1, \cdots, g_m\}$$
$$\tau \setminus \sigma = \{t_1, \cdots, t_n\}$$

with $m, n \geq 0$. Let $a_1, \ldots, a_{m+n}$ be the closes Ψ-terms not in V, σ, and τ. Abusing language by forgetting in the notation about these a_i's, we denote by $Ag^V_{(\theta,\sigma)\to(\theta,\tau)}$, the agent

$$\begin{gathered} \mathit{get}(g_1); \mathit{tell}(a_1); \\ \cdots \\ \mathit{get}(g_m); \mathit{tell}(a_m); \\ \mathit{tell}(t_1); \mathit{tell}(a_{m+1}); \\ \cdots \\ \mathit{tell}(t_n); \mathit{tell}(a_{m+n}); \\ \mathit{get}(a_1); \cdots; \mathit{get}(a_{m+n}) \end{gathered}$$

Note that, as all the get primitives have a closed Ψ-term as argument, they cannot modify the binding of any variables. Hence $Vars(\mathcal{D}_h(Ag^V_{(\theta,\sigma)\to(\theta,\tau)})) = \emptyset$ holds. Moreover, we note by $\Sigma^V_{(\theta,\sigma)\to(\theta,\tau)}$ the associated sequence of states

$$\begin{gathered}
((\theta,\xi_0),(\theta,\gamma_1)).((\theta,\gamma_1),(\theta,\xi_1)).\\
\cdots\\
((\theta,\xi_{m-1}),(\theta,\gamma_m)).((\theta,\gamma_m),(\theta,\xi_m)).\\
((\theta,\xi_m),(\theta,\tau_1)).((\theta,\tau_1),(\theta,\xi_{m+1})).\\
\cdots\\
((\theta,\xi_{m+n-1}),(\theta,\tau_n)).((\theta,\tau_n),(\theta,\xi_{m+n})).\\
((\theta,\rho_0),(\theta,\rho_1)).\cdots.((\theta,\rho_{m+n-1}),(\theta,\rho_{m+n}))
\end{gathered}$$

where

$$\begin{array}{ll}
\xi_0 = \sigma & \\
\rho_0 = \xi_{m+n} & \\
\rho_{m+n} = \tau & \\
\gamma_i = \xi_{i-1} \setminus \{g_i\} & (1 \le i \le m)\\
\xi_i = \gamma_i \cup \{a_i\} & (1 \le i \le m)\\
\tau_j = \xi_{m+j-1} \cup \{t_j\} & (1 \le j \le n)\\
\xi_{m+j} = \tau_j \cup \{a_{m+j}\} & (1 \le j \le n)\\
\rho_k = \rho_{k-1} \setminus \{a_k\} & (1 \le k \le m+n)
\end{array}$$

Obviously, $Ag^V_{(\theta,\sigma)\to(\theta,\tau)}$ can generate histories of the form $\Sigma^V_{(\theta,\sigma)\to(\theta,\tau)}.((\kappa,\gamma),\delta^+)$ for any binding κ such that $\theta \prec \kappa$ and any store γ. If V is suitably chosen, it also has the property of being responsible for making the steps of $\Sigma^V_{(\theta,\sigma)\to(\theta,\tau)}$ when placed in parallel with another agent.

Proposition 3.16 *Let σ and τ be two stores. Let θ be a binding. Let A be an agent and let V contain the Ψ-terms present in the tell, get, ask and nask communication primitives of A.*

(i) *Any history $h = \Sigma^V_{(\theta,\sigma)\to(\theta,\tau)}.h'$ of $\mathcal{D}_h(Ag^V_{(\theta,\sigma)\to(\theta,\tau)} \parallel A)$ is of the set $\Sigma^V_{(\theta,\sigma)\to(\theta,\tau)}.((\kappa,\gamma),\delta^+)\ \widetilde{\|}_h\ h_a$ for some store γ, some binding $\kappa \succ \theta$ and some history $h_a \in \mathcal{D}_h(A)$.*

(ii) *For any agent B, any history $h = \Sigma^V_{(\theta,\sigma)\to(\theta,\tau)}.h'$ of $\mathcal{D}_h((Ag^V_{(\theta,\sigma)\to(\theta,\tau)}\ ;\ B) \parallel A)$ is of the set $\Sigma^V_{(\theta,\sigma)\to(\theta,\tau)}.h_b\ \widetilde{\|}_h\ h_a$ for some histories $h_a \in \mathcal{D}_h(A)$ and $h_b \in \mathcal{D}_h(B)$.*

Proof Let us establish the first part of the proposition, the proof of the other part being similar.

By definition 3.8, if h is in $\mathcal{D}_h(Ag^V_{(\theta,\sigma)\to(\theta,\tau)} \;||\; A)$, there are $h_1 \in \mathcal{D}_h(Ag^V_{(\theta,\sigma)\to(\theta,\tau)})$ and $h_2 \in \mathcal{D}_h(A)$ such that $h \in h_1 \;\widetilde{\|}_h\; h_2$. Moreover, in view of $Ag^V_{(\theta,\sigma)\to(\theta,\tau)}$, h_1 is necessarily of the following form:

$$\begin{gathered}((\theta_1,\alpha_1),(\theta_1,\alpha_1\setminus\{g_1\})).((\theta_2,\beta_1),(\theta_2,\beta_1\cup\{a_1\})).\\ \cdots.\\ ((\theta_{2m-1},\alpha_m),(\theta_{2m-1},\alpha_m\setminus\{g_m\})).((\theta_{2m},\beta_m),(\theta_{2m},\beta_m\cup\{a_m\})).\\ ((\theta_{2m+1},\alpha_{m+1}),(\theta_{2m+1},\alpha_{m+1}\cup\{t_1\})).\\ ((\theta_{2m+2},\beta_{m+1}),(\theta_{2m+2},\beta_{m+1}\cup\{a_{m+1}\})).\\ \cdots\\ ((\theta_{2m+2n-1},\alpha_{m+n}),(\theta_{2m+2n-1},\alpha_{m+n}\cup\{t_n\})).\\ ((\theta_{2m+2n},\beta_{m+n}),(\theta_{2m+2n},\beta_{m+n}\cup\{a_{m+n}\})).\\ ((\theta_{2m+2n+1},\pi_1),(\theta_{2m+2n+1},\pi_1\setminus\{a_1\})).\\ \cdots\\ ((\theta_{3m+3n},\pi_{m+n}),(\theta_{3m+3n},\pi_{m+n}\setminus\{a_{m+n}\}))\end{gathered}$$

Let us first progressively establish that $h_1 = \Sigma^V_{(\theta,\sigma)\to(\theta,\tau)}.h'_1$ for some history h'_1.

Using the notations of definition 3.15, we first observe that h_2 cannot be of the form $((\theta,\xi_0),(\theta,\gamma_1)).h'_2$. Indeed, if this was the case, then, in view of the merge operator $\widetilde{\|}_h$, either $h_1 = ((\theta,\gamma_1),(\theta,\xi_1)).h'_1$ or $h'_2 = ((\theta,\gamma_1),(\theta,\xi_1)).h''_2$. However, both cases are impossible. In the first case, in view of the above form of h_1, one would have $\xi_1 = \gamma_1 \setminus \{g_1\}$ and thus $a_1 \notin \xi_1$ since $a_1 \notin \gamma_1$ whereas by definition of ξ_1, $a_1 \in \xi_1$. In the second case, since a_1 cannot be told by A by choice of a_1, then again $a_1 \notin \xi_1$ whereas by definition of ξ_1, $a_1 \in \xi_1$. Hence,

$$h_1 = ((\theta,\xi_0),(\theta,\gamma_1)).r_1$$

Moreover, as just explained, by its choice, a_1 cannot be told by A and consequently, h_2 cannot be of the form $h_2 = ((\theta,\gamma_1),(\theta,\xi_1)).h'_2$, f or some h'_2. It follows that

$$h_1 = ((\theta,\xi_0),(\theta,\gamma_1)).((\theta,\gamma_1),(\theta,\xi_1)).r_2$$

By similar reasoning, h_1 can be proved to be of the form

$$\begin{aligned}h_1 = {} & ((\theta,\xi_0),(\theta,\gamma_1)).((\theta,\gamma_1),(\theta,\xi_1)).\cdots.\\ & ((\theta,\xi_{m-1}),(\theta,\gamma_m)).((\theta,\gamma_m),(\theta,\xi_m)).\\ & ((\theta,\xi_m),(\theta,\tau_1)).((\theta,\tau_1),(\theta,\xi_{m+1})).\cdots\\ & ((\theta,\xi_{m+n-1}),(\theta,\tau_n)).((\theta,\tau_n),(\theta,\xi_{m+n})).r_3\end{aligned}$$

Now, since by definition A cannot get any a_i, h_1 must further be of the form

$$((\theta,\xi_0),(\theta,\gamma_1)).((\theta,\gamma_1),(\theta,\xi_1)).$$
$$\dots.$$
$$((\theta,\xi_{m-1}),(\theta,\gamma_m)).((\theta,\gamma_m),(\theta,\xi_m)).$$
$$((\theta,\xi_m),(\theta,\tau_1)).((\theta,\tau_1),(\theta,\xi_{m+1})).$$
$$\dots$$
$$((\theta,\xi_{m+n-1}),(\theta,\tau_n)).((\theta,\tau_n),(\theta,\xi_{m+n})).$$
$$((\theta,\rho_0),(\theta,\rho_1)).$$
$$\dots$$
$$((\theta,\rho_{m+n-1}),(\theta,\rho_{m+n})).r_4$$

Summing up, the agent $Ag^V_{(\theta,\sigma)\to(\theta,\tau)}$ has made all the 3*(m+n) steps and thus has reached completion successfully. It follows that r_4 should be $((\kappa,\gamma),\delta^+)$, for some store γ and some binding $\kappa \succ \theta$. ∎

Proposition 3.16 can be extended to more general sets of denotational histories.

Definition 3.17 A set of denotational histories is called *coherent* if it is extensible (in the sense of proposition 3.11), if it its set *diff*(S) is finite and if it satisfies $Vars(S) \cap Ext(S) = \emptyset$.

Proposition 3.18 *Let σ and τ be two stores. Let S be a coherent subset of Shhist and let V contain diff(S).*

(i) *Any history $h = \Sigma^V_{(\theta,\sigma)\to(\theta,\tau)}.h'$ of $\mathcal{D}_h(Ag^V_{(\theta,\sigma)\to(\theta,\tau)}) \mathbin{\widetilde{\|}} S$ is of the set $\Sigma^V_{(\theta,\sigma)\to(\theta,\tau)}.((\kappa,\gamma),\delta^+) \mathbin{\|_h} h_s$ for some store γ, some binding $\kappa \succ \theta$, and some history $h_s \in S$.*

(ii) *For any agent B, any history $h = \Sigma^V_{(\theta,\sigma)\to(\theta,\tau)}.h'$ of $\mathcal{D}_h(Ag^V_{(\theta,\sigma)\to(\theta,\tau)}\ ;\ B) \mathbin{\widetilde{\|}} S$ is of the set $\Sigma^V_{(\theta,\sigma)\to(\theta,\tau)}.h_b \mathbin{\|_h} h_s$ for some histories $h_s \in S$ and $h_b \in \mathcal{D}_h(B)$.*

Proof Similar to that of proposition 3.16. ∎

3.3.4 Key proposition.

Theorem 3.19 *Let S_1, S_2 be two coherent subsets of Shhist such that $S_1 \setminus S_2 \neq \emptyset$. Let $g \in S_1 \setminus S_2$ of minimal length and $init(g) = (\theta,\sigma)$. Then, for any store α, there is an agent T such that $Vars(\mathcal{D}_h(T)) \subseteq Ext(S_1)$ and a continuous history $h \in (S_1 \mathbin{\widetilde{\|}} \mathcal{D}_h(T)) \setminus (S_2 \mathbin{\widetilde{\|}} \mathcal{D}_h(T))$ which starts in (θ,α).*

Proof The proof is conducted by induction on the minimum Lg of the length of the histories which are in S_1 and not in S_2.

Case I: $Lg = 1$.

Then $g \in S_1 \setminus S_2$ is of the form $((\theta,\sigma),\delta^+)$ or of the form $((\theta,\sigma),\delta^-)$.

Subcase i: $g = ((\theta,\sigma),\delta^+)$. Let us first examine the case where $g = ((\theta,\sigma),\delta^+)$. By hypothesis, $((\theta,\sigma),\delta^+) \notin S_2$. Let V be the set *diff* (S_2). Consider $T = Ag^V_{(\theta,\alpha)\rightarrow(\theta,\sigma)}$. Obviously, $Vars(\mathcal{D}_h(T)) = \emptyset$ and thus $Vars(\mathcal{D}_h(T)) \subseteq Ext(S_1)$. Moreover, $h = \Sigma^V_{(\theta,\alpha)\rightarrow(\theta,\sigma)}.((\theta,\sigma),\delta^+)$ is a continuous history belonging to $S_1 \widetilde{\|} \mathcal{D}_h(T)$. To conclude in this case, let us prove that it does not belong to $S_2 \widetilde{\|} \mathcal{D}_h(T)$. Indeed, if so, by proposition 3.18, h should come from the following merge:

$$h \in \Sigma^V_{(\theta,\alpha)\rightarrow(\theta,\sigma)}.((\kappa,\gamma),\delta^+) \ \widetilde{\|}_h \ h_s$$

for some store γ, some binding $\kappa \succ \theta$ and some history $h_s \in S_2$. Moreover, since h ends after $\Sigma^V_{(\theta,\alpha)\rightarrow(\theta,\sigma)}$ by $((\theta,\sigma),\delta^+)$, one should have, by definition of the merge (see definition 3.5), $\gamma = \sigma$, $\kappa = \theta$, and $h_s = (\sigma,\delta^+)$. Therefore, $((\theta,\sigma),\delta^+)$ should belong to S_2, which contradicts the hypothesis on A_2.

Subcase ii: $g = ((\theta,\sigma),\delta^-)$. The case where $g = ((\theta,\sigma),\delta^-)$ can be treated similarly with the proof ending by noting that $h_s = ((\theta,\sigma),\delta^-)$ should belong to S_2, which contradicts the hypothesis.

Case II: $Lg > 1$.

Let us now consider the case where the minimum of the lengths of the histories of $S_1 \setminus S_2$ is greater than 1. In that case, g is of the form $g = ((\theta,\sigma),(\kappa,\tau)).h'$ f or some stores σ,τ, some bindings θ,κ, and some history h'. There are two cases to be considered: either $S_2[((\theta,\sigma),(\kappa,\tau))] = \emptyset$ or $S_2[((\theta,\sigma),(\kappa,\tau))] \neq \emptyset$ but $h' \notin S_2[((\theta,\sigma),(\kappa,\tau))]$.

Subcase i: $S_2[((\theta,\sigma),(\kappa,\tau))] = \emptyset$. If $S_2[((\theta,\sigma),(\kappa,\tau))] = \emptyset$, then let us first observe, by proposition 3.11, that there is a continuous history of the form $((\theta,\sigma),(\kappa,\tau)).h_r$ in S_1. Let us then consider V and $T = Ag^V_{(\theta,\alpha)\rightarrow(\theta,\sigma)}$ as above. Obviously, $h^* = \Sigma^V_{(\theta,\alpha)\rightarrow(\theta,\sigma)}.((\theta,\sigma),(\kappa,\tau)).h_r$ is a continuous history starting in (θ,α). Moreover, if $((\rho,\omega),\delta)$ is the last pair of the sequence h_r, then h^* belongs to the merge of the histories $\Sigma^V_{(\theta,\alpha)\rightarrow(\theta,\sigma)}.((\rho,\omega),\delta^+)$ and $((\theta,\sigma),(\kappa,\tau)).h_r$, and consequently to $S_1 \widetilde{\|} \mathcal{D}_h(T)$. To conclude, let us establish that it does not belong to $S_2 \widetilde{\|} \mathcal{D}_h(T)$. Indeed, otherwise, following proposition 3.18, h^* should then come from the merge of two histories of the form $\Sigma^V_{(\theta,\alpha)\rightarrow(\theta,\sigma)}.((\phi,\gamma),\delta^+)$ and h_s with $h_s \in S_2$. According to the definition of the merge operation, one would then have $h_s = ((\theta,\sigma),(\kappa,\tau)).h_r$ and thus $S_2[((\theta,\sigma),(\kappa,\tau))]$ would be non-empty, which contradicts the hypothesis.

Subcase ii: $S_2[((\theta,\sigma),(\kappa,\tau))] \neq \emptyset$ *but* $g' \notin S_2[((\theta,\sigma),(\kappa,\tau))]$. In that case, by hypothesis $g' \in S_1[((\theta,\sigma),(\kappa,\tau))] \setminus S_2[((\theta,\sigma),(\kappa,\tau))]$ and the minimum of the length of those histories which are in $S_1[((\theta,\sigma),(\kappa,\tau))]$ and not in $S_2[((\theta,\sigma),(\kappa,\tau))]$ is strictly less than Lg. We are thus in the position of applying the induction hypothesis. Let $init(g') = (\theta',\sigma')$. Note that $Ext(S_1[((\theta,\sigma),(\kappa,\tau))]) \subseteq Ext(S_1)$ and $Vars(S_1[((\theta,\sigma),(\kappa,\tau))]) \subseteq Vars(S_1)$. Applying the induction hypothesis delivers, for an arbitrarily given store α' — to be specified in a moment — a tester T' such that $Vars(\mathcal{D}_h(T')) \subseteq Ext(S_1[((\theta,\sigma),(\kappa,\tau))])$ and a continuous history h_r^* starting in (θ',α') and which is in $(S_1[((\theta,\sigma),(\kappa,\tau))] \mathbin{\widetilde{\|}} \mathcal{D}_h(T')) \setminus (S_2[((\theta,\sigma),(\kappa,\tau))] \mathbin{\widetilde{\|}} \mathcal{D}_h(T'))$. The proof then consists of prefixing T' by some actions, yielding T, and h_r^* by a suitable sequence, yielding h^*, such that h^* starts, as required, in (θ,α), is continuous, and is in $S_1 \mathbin{\widetilde{\|}} \mathcal{D}_h(T)$ and not in $S_2 \mathbin{\widetilde{\|}} \mathcal{D}_h(T)$. Applying the previous technique, T should start by $Ag^V_{(\theta,\alpha)\to(\theta,\sigma)}$ to bring the situation (θ,α) to (θ,σ), then leave S_1 and S_2 do the step $((\theta,\sigma),(\kappa,\tau))$, follow by $B^W_{(\kappa,\alpha')\to(\theta',\alpha')}$ where W is taken as $diff(S_2)$ and finally resume by doing T'. In order to force the S_i's to do so, we need a trick which basically consists of adding in h^* after $((\theta,\sigma),(\kappa,\tau))$ a step that can only be made by T. Hence, let t be a fresh closed Ψ-term not appearing in $diff(S_1)$, $diff(S_2)$, and in the Ψ-terms used by $Ag^V_{(\theta,\alpha)\to(\theta,\sigma)}$ and let α' be $\tau \cup \{t\}$. Moreover, let us take

$$T = Ag^V_{(\theta,\alpha)\to(\theta,\sigma)} \ ; \ tell(t) \ ; \ B^W_{(\kappa,\alpha')\to(\theta',\alpha')} \ ; \ T'$$

and

$$h^* = \Sigma^V_{(\theta,\alpha)\to(\theta,\sigma)}.((\theta,\sigma),(\kappa,\tau)).((\kappa,\tau),(\kappa,\alpha')).\Gamma^W_{(\kappa,\alpha')\to(\theta',\alpha')}.h_r^*.$$

Note that $Ag^V_{(\theta,\alpha)\to(\theta,\sigma)}$ and $tell(t)$ interfere with no variables. Moreover, $B^W_{(\kappa,\alpha')\to(\theta',\alpha')}$ is such that

$$Vars(B^W_{(\kappa,\alpha')\to(\theta',\alpha')}) \subseteq Ext(g) \subseteq Ext(S1)$$

and the agent T' affects only the variables of $Ext(S_1[((\theta,\sigma),(\kappa,\tau))]) \subseteq Ext(S_1)$. The agent T thus does not alter the variables of $Ext(S_1)$. Finally, the history h^* is in $(S_1 \mathbin{\widetilde{\|}} \mathcal{D}_h(T))$. Indeed, as h_r^* is in $(S_1[((\theta,\sigma),(\kappa,\tau))] \mathbin{\widetilde{\|}} \mathcal{D}_h(T'))$, there are $h_1 \in S_1[((\theta,\sigma),(\kappa,\tau))]$ and $h_t \in \mathcal{D}_h(T')$ such that $h_r^* \in h_1 \mathbin{\widetilde{\|}_h} h_t$. Consequently, $((\theta,\sigma),(\kappa,\tau)).h_1 \in S_1$ and $\Sigma^V_{(\theta,\alpha)\to(\theta,\sigma)}.((\kappa,\tau),(\kappa,\alpha')).\Gamma^W_{(\kappa,\alpha')\to(\theta',\alpha')}.h_t \in \mathcal{D}_h(T)$. Summming up,

$$h^* \in ((\theta,\sigma),(\kappa,\tau)).h_1 \mathbin{\widetilde{\|}_h} \Sigma^V_{(\theta,\alpha)\to(\theta,\sigma)}.((\kappa,\tau),(\kappa,\alpha')).((\kappa,\alpha'),(\theta',\alpha')).h_t$$

and thus h^* is in $S_1 \mathbin{\widetilde{\|}} \mathcal{D}_h(T)$.

To conclude, it remains to be established that $h^* \notin S_2 \mathbin{\widetilde{\|}} \mathcal{D}_h(T)$. Two cases need to be discussed according as T has variables in common with $Vars(S_2)$.

Case a. $Vars(S_2) \cap Vars(\mathcal{D}_h(T)) \neq \emptyset$. In this case, $S_2 \mathbin{\widetilde{\|}} \mathcal{D}_h(T) = \emptyset$ and the proof is concluded.

Case b. $Var(S_2) \cap Vars(\mathcal{D}_h(T)) = \emptyset$. Here we shall proceed by contradiction as before. Otherwise, in view of proposition 3.18, h^* should be in the set $\Sigma^V_{(\theta,\sigma)\to(\kappa,\tau)}.h_t \mathbin{\widetilde{\|}_h} h_s$ for some histories $h_t \in \mathcal{D}_h(tell(t) \ ; \ B^W_{(\kappa,\alpha')\to(\theta',\alpha')} \ ; \ T')$ and $h_s \in S_2$. Moreover, T cannot be responsible for the step $((\theta,\sigma),(\kappa,\tau))$ ie, restated in formal terms, h_t cannot be of the form $h_t = ((\theta,\sigma),(\kappa,\tau)).h_t'$. Indeed, if this was the case, then $\tau = \sigma \cup \{t\}$, whereas by definition $t \notin \tau$. Hence, $h_s = ((\theta, \sigma), (\kappa, \tau)).h'_s$ for some history h'_s. Note that, since $h_s \in S_2$, $h'_s \in S_2[((\theta, \sigma), (\kappa, \tau))]$. Moving one step further in h^*, again, thanks to the choice of t, S_2 cannot perform the step $((\kappa, \tau), (\kappa, \alpha'))$ ie h'_s cannot rewrite as $h'_s = ((\kappa, \tau), (\kappa, \alpha')).h''_s$. In view of the choice of W, none of the steps of $\Gamma^W_{(\kappa,\alpha')\to(\theta',\alpha')}$ can be performed by h'_s. Therefore, $h_t = ((\kappa, \tau), (\kappa, \alpha')).\Gamma^W_{(\kappa,\alpha')\to(\theta',\alpha')}.h'_t$.

Summing up, $h^*_r \in h'_t \mathbin{\widetilde{\|}_h} h'_s$ for some histories $h'_t \in \mathcal{D}_h(T')$ and $h'_s \in S_2[((\theta, \sigma), (\kappa, \tau))]$ and, consequently, $h^*_r \in S_2[((\theta, \sigma), (\kappa, \tau))] \mathbin{\widetilde{\|}} \mathcal{D}_h(T')$, which contradicts the fact that by construction h^*_r is in $(S_1[((\theta, \sigma), (\kappa, \tau))] \mathbin{\widetilde{\|}} \mathcal{D}_h(T')) \setminus (S_2[((\theta, \sigma), (\kappa, \tau))] \mathbin{\widetilde{\|}} \mathcal{D}_h(T'))$. ■

3.3.5 Proof of the full abstraction property.

We are now in a position to establish the full abstraction property.

Theorem 3.20 *The semantics $\mathcal{D}_h$ is fully abstract with respect to the semantics $\mathcal{O}_h$.*

Proof Following definition 3.13, two following properties should be established equivalent:

i) for any context C, $\mathcal{O}_h(C[A_1]) = \mathcal{O}_h(C[A_2])$;

ii) $\mathcal{D}_h(A_1) = \mathcal{D}_h(A_2)$.

The implication ii) $\Rightarrow$ i) follows directly from proposition 3.9. The other implication i) $\Rightarrow$ ii) is proved by contraposition. Assume $\mathcal{D}_h(A_1) \neq \mathcal{D}_h(A_2)$. Then, since both $\mathcal{D}_h(A_1)$ and $\mathcal{D}_h(A_2)$ are sets, there is an history h which is in one set and not in the other one. Without lost of generality, we may assume that $h \in \mathcal{D}_h(A_1)$ and $h \notin \mathcal{D}_h(A_2)$. Moreover, let $\mathit{init}(h) = (\theta, \sigma)$. Then $\mathcal{D}_h(A_1) \setminus \mathcal{D}_h(A_2) \neq \emptyset$ and, consequently taking as coherent sets $S_1 = \mathcal{D}_h(A_1)$, $S_2 = \mathcal{D}_h(A_2)$, and $\emptyset$ as initial store α, theorem 3.19 establishes that there is an agent T and a continuous history $g \in (\mathcal{D}_h(A_1) \mathbin{\widetilde{\|}} \mathcal{D}_h(T)) \setminus (\mathcal{D}_h(A_2) \mathbin{\widetilde{\|}} \mathcal{D}_h(T))$ which starts in $(\theta, \emptyset)$. Furthermore, $\mathit{Vars}(\mathcal{D}_h(T)) \subseteq \mathit{Ext}(\mathcal{D}_h(A_1))$ ie $\mathit{Vars}(\mathcal{D}_h(T)) \cap \mathit{Vars}(\mathcal{D}_h(A_1)) = \emptyset$. Note that, by definition 3.8, $g \in \mathcal{D}_h(A_1 \mid\mid T) \setminus \mathcal{D}_h(A_2 \mid\mid T)$. Consider now $S = B^{\emptyset}_{(\bot,\emptyset)\to(\theta,\emptyset)}$ and the history $((\bot, \emptyset), (\bot, \{\psi\})).((\bot, \{\psi\}), (\theta, \emptyset)).((\kappa, \tau), \delta^+) \in \mathcal{D}_h(B_{(\bot,\emptyset)\to(\theta,\emptyset)})$. The history $k = ((\bot, \emptyset), (\bot, \{\psi\})).((\bot, \{\psi\}), (\theta, \emptyset)).g$ is continuous, starts in $(\bot, \emptyset)$, belongs to $\mathcal{D}_h(S) \mathbin{\tilde{;}} (\mathcal{D}_h(A_1) \mathbin{\widetilde{\|}} \mathcal{D}_h(T))$ but not to $\mathcal{D}_h(S) \mathbin{\tilde{;}} (\mathcal{D}_h(A_2) \mathbin{\widetilde{\|}} \mathcal{D}_h(T))$. Therefore, by proposition 3.9, $\overline{k}$ is an operational history of $\mathcal{O}_h(S; (A_1 \mid\mid T))$ which is not in $\mathcal{O}_h(S; (A_2 \mid\mid T))$. There is thus a context $C = S; (\square \mid\mid T)$, such that $\mathcal{O}_h(C[A_1]) \neq \mathcal{O}_h(C[A_2])$, which concludes the proof.

■

4 Concluding Remarks

We have considered a simple language that embodies the essential features of coordination languages based on generative communication à la Linda as well as an improved form of matching based on Ψ-terms.

The definition of a fully abstract compositional semantics for the language sets a firm foundation for reasoning about agents and agent compositions in a coordination-oriented setting. For instance, the denotational semantics induces a number of semantic equalities among agents. The following proposition reports a few classical ones.

Proposition 4.1

$$\begin{array}{ll}
(C1) & X + X = X \\
(C2) & X + Y = Y + X \\
(C3) & X + (Y + Z) = (X + Y) + Z \\
 & \\
(P1) & X \,||\, Y = Y \,||\, X \\
(P2) & X \,||\, (Y \,||\, Z) = (X \,||\, Y) \,||\, Z \\
 & \\
(SC) & (X + Y)\, ;\, Z = (X\, ;\, Z) + (Y\, ;\, Z) \\
 & \\
(E1) & X\, ;\, E = X \\
(E2) & E\, ;\, X = X \\
(E3) & E \,||\, X = X
\end{array}$$

Proof Direct from definitions 3.4, 3.5, 3.6, 3.7, and 3.8. ■

It is worth noting that, on the other hand, the equality

$$X\, ;\, (Y + Z) = (X\, ;\, Y) + (X\, ;\, Z)$$

does not hold.

Future work will be devoted to develop and analyse semantics-preserving program transformation techniques in this setting.

As we already pointed out in the Introduction, our work is an extension of previous work on the denotational semantics of Linda-like languages ([1]). It extends it so as to manipulate not just tokens but richer structures which may be partially defined. These structures, called Ψ-terms, introduce the need for novel treatments, notably for a proper handling of values generated by primitives acting on the shared dataspace.

As a result of our previous work, the denotational semantics proposed in this paper takes also inspiration from the denotational models for concurrent languages proposed by Horita, de Bakker and Rutten in [7]. In [7] the problem of defining fully abstract denotational semantics for concurrent languages is studied in the context of an concurrent imperative setting based on assignments of variables and if-then-else constructs as basic operations. Our contribution has been to show that whereas the concurrent language $\mathcal{L}_\Psi$ differs substantially from this setting, the denotational model proposed in [7] can be applied to characterize $\mathcal{L}_\Psi$. Namely, the idea of employing *gaps* in state sequences to represent possible interactions of agents with the state and the testing technique allow us to obtain a fully abstract compositional denotational semantics for $\mathcal{L}_\Psi$.

A general framework embodying a variety of concurrent languages all based on asynchronous communication is studied in [6]. Among others, a full abstraction result similar to ours is claimed however without much details. Our contribution was to show how the peculiarities of the $\mathcal{L}_\Psi$ language can be used to actually establish the claim in a restricted setting.

Full abstraction for a shared variable parallel imperative language is also studied in [2]. However, the observables are composed of the final states of the computation coupled to termination marks. They thus substantially differ from what is returned by the semantics $\mathcal{O}_h$ and hence, so are the denotational semantics although traces are also used in [2].

De Nicola and Pugliese defined in [9] a testing scenario for a process algebra based on Linda's primitives. The language considered in [9] is richer than the language $\mathcal{L}_\Psi$ considered in this paper, in that the former includes several other composition operators and allows infinite processes. The work by De Nicola and Pugliese however differs from ours in the type of description chosen to model Linda-based coordination languages (testing vs. denotational semantics).

5 Acknowledgment

The authors thank K. Apt, M. Bonsangue, J.W. de Bakker, F. de Boer, E. de Vinck, E. Horita, J. Kok, U. Montanari, J. Rutten for stimulating discussions on the semantics of coordination languages.

References

[1] A. Brogi and J.-M. Jacquet. Modeling Coordination via Asynchronous Communication. In D. Garlan and D. Le Métayer, editors, *Proceedings of the Second International Conference on Coordination Languages and Models*, volume 1282 of *Lecture Notes in Computer Science*, pages 238–255, Berlin, Germany, 1997.

[2] S. Brookes. Full Abstraction for a Shared-Variable Parallel Language. In *Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 98–109, Montreal, Canada, June 1993. IEEE Computer Society Press.

[3] N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32(4):444–458, 1989.

[4] N. Carriero and D. Gelernter. Coordination Languages and Their Significance. *Communications of the ACM*, 35(2):97–107, 1992.

[5] P. Ciancarini and C. Hankin, editors. *Proceedings of The First International Conference on Coordination Models and Languages*, number 1061 in LNCS. Springer-Verlag, 1996.

[6] F.S. de Boer, J.N. Kok, C. Palamidessi, and J.J.M.M. Rutten. The Failure of Failures in a Paradigm of Asynchronous Communication. In J.C.M. Baeten and J.F. Groote, editors, *Proc. 2^{nd} Int. Conf. on Concurrency Theory (Concur'91)*, volume 527 of *Lecture Notes in Computer Science*, pages 111–126, Amsterdam, The Netherlands, 1991. Springer-Verlag.

[7] E. Horita, J.W. de Bakker, and J.J.M.M. Rutten. Fully abstract denotational models for nonuniform concurrent languages. *Information and computation*, 115(1):125–178, 1994.

[8] R. Milner. *A Calculus of communicating systems.* Springer-Verlag, 1980.

[9] R. De Nicola and R. Pugliese. A process algebra based on Linda. In P. Ciancarini and C. Hankin, editors, *COORDINATION 96*, number 1061 in LNCS. Springer-Verlag, 1996.

[10] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN-19, Aarhus University, 1981.