Fully abstract models and refinements as tools to compare agents in timed coordination languages

Jean-Marie Jacquet, Isabelle Linden

Faculty of Informatics, FUND–University of Namur, Belgium

ARTICLE INFO
Keywords:
Coordination languages
Semantics
Refinement

ABSTRACT
Coordination languages and models promote the idea of separating computation and interaction aspects. As for traditional concurrency models, the question of safely replacing an agent by another one in any interacting context naturally appears. This paper proposes two tools to answer that question. On the one hand, a fully abstract semantics allows us to identify two processes which behave similarly in any context. On the other hand, a refinement theory allows us to compare processes that appear to be different in view of the fully abstract semantics but which satisfy the substitutability property: if the implementation $I$ refines the specification $S$ and if $C[S]$ is deadlock free, for some context $C$, then $C[I]$ is also deadlock free. Both theories are novel, are exposed in the context of our timed coordination languages but may actually be transposed in the context of almost any data-driven coordination language.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction
As motivated by the constant expansion of computer networks and illustrated by the development of distributed applications, the design of modern software systems centers on re-using and integrating software components. This induces a paradigm shift from stand-alone applications to interacting distributed systems, which, in turn, naturally calls for well-defined methodologies and tools aiming at integrating heterogeneous software components.

In this context, a clear separation between the interactional and the computational aspects of software components has been advocated by Gelernter and Carriero in [12]. Their claim has been supported by the design of a model, Linda [5], originally presented as a set of inter-agent communication primitives which may be added to almost any programming language. Besides process creation, this set includes primitives for adding, deleting, and testing the presence/absence of data in a shared dataspace.

A number of other models, now referred to as coordination models, have been proposed afterwards (see [26,27] for a comprehensive survey of many of them). One of the extensions, of interest for this paper, concerns the introduction of time. It is motivated both by industrial proposals such as JavaSpaces [11] and TSpaces [31] as well as by the coding of applications which evidence the fact that data and requests rarely have an eternal life. For instance, a request for information on the web has to be satisfied in a reasonable amount of time. Even more crucial is the request in a critical system which, not only has to be eventually answered, but within a critical period of time. The list could also be continued with software in the areas of air-traffic control, manufacturing plants and telecommunication switches, which are inherently reactive and, for which, interaction must occur in “real-time”.

* A preliminary partial version of this paper appeared in Coordination 2007.
* Corresponding address: Faculty of Informatics, FUND–University of Namur, rue Grandgagnage 21, 5000 Namur, Belgium. Tel.: +32 81 724987; fax: +32 81 724967.
E-mail addresses: jmj@info.fundp.ac.be (J.-M. Jacquet), ili@info.fundp.ac.be (I. Linden).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
In our recent work [16,20–22], we have proposed different ways of introducing time in coordination languages. For that purpose, we have used the classical two-phase functioning approach to real-time systems and have proved that this approach was effective for modeling coordination in reactive systems.

However, although the need for techniques and tools to reason about concurrent programs is widely recognized in the concurrency community, using process algebras such as CCS [23], CSP [14], π-calculus [24], and is met by a large body of work, little attention has been paid to programming methodologies in the coordination community. This lack is even more crucial in the context of real-time systems for which strict delays have to be guaranteed.

This paper aims at contributing to this effort. Work in other settings like the B-method [1], the FDR tool [28] and the concurrency workbench [7] have evidenced that fully abstract models and refinements are fundamental notions for reasoning. After having shown that the traditional fully abstract models and refinements do not transpose directly in a satisfactory way to Linda-like languages and, consequently, to our timed coordination framework, we shall propose a new fully abstract model and a new notion of refinement. The interest of the fully abstract model is that it allows one to determine which agents have identical behaviors under any context. Although very interesting, this might however be too strong to compare, for instance, a specification and an implementation, which, in general, offers less behaviors. To that end, we shall present a novel theory of refinement which satisfies the substitutability property: if the implementation I refines the specification S and if C[S] is deadlock free, for some context C, then C[I] is also deadlock free. This property is particularly crucial since it allows a compositional way of reasoning and thereby helps model checking to scale. Moreover, as Linda-like languages can be embedded in our time setting, our notion of refinement also applies to Linda-like languages and thus to a wide range of languages.

The rest of this paper is organized as follows. Section 2 introduces our timed coordination models. Section 3 explains why the classical fully abstract models and the usual notions of trace-refinement and failure-refinement are not suited for the coordination context. Section 4 presents our fully abstract model. Its study is complemented in Section 5 by an event based semantics, necessary to support the explanation of our refinement theory in Section 6. Section 7 compares the semantics and the refinement. Finally, Section 8 draws our conclusions and compares our work with related work.

2. Timed coordination languages

Our approach to the introduction of time in coordination languages follows the classical two-phase functioning approach to real-time systems illustrated by languages such as Lustre [6], Esterel [2]. Statecharts [13] and related with the notion of urgency in process calculi [25]. This approach may be described as follows. In a first phase, elementary actions of statements are executed. They are assumed to be atomic in the sense that they take no time. Similarly, composition operators are assumed to be executed at no cost. In a second phase, when no actions can be reduced or when all the components encounter a special timed action, time progresses by one unit.

In that context, four families of timed coordination languages have been introduced in [16]. They are obtained

(1) by introducing delays stating that a communication primitive should only be processed after some units of time;
(2) by stating that tuples on the tuple space are only valid for some units of time and that, similarly, requests for tuples are to be made during a period of time;
(3) by introducing delays for some specific points in time;
(4) by specifying absolute intervals of time in which actions should be processed and, dually, by associating such intervals with tuples.

The first two families are said to incorporate a relative notion of time since the delays and the validity of tuples and of operations are defined at execution time with respect to their moments of consideration. In contrast, the last two families are said to incorporate an absolute notion of time because they refer statically to specific instances of a clock.

The expressiveness of these families has been studied in [16,20–22]. Two interesting conclusions may be extracted from these papers. On the one hand, from a programming point of view, the language embodying relative delays and relative timed primitives (namely embodying the features of points 1 and 2 above) is the most expressive one. On the other hand, from an implementation point of view, as relative primitives can be easily translated into absolute times thanks to the current time of execution, the language incorporating absolute delays and absolute timed primitives (namely the features of points 3 and 4 above) is the most fundamental one. We shall thus use the former language is this paper. Formally it is defined as follows.

**Definition 1.** Let Stoken be a denumerable set, the elements of which are subsequently called tokens and are typically represented by the letters t and u. Let Stime be the set of time units or durations defined as the set composed by the positive integers. Elements of Stime are typically represented by the letter d. Moreover, let Stime0 be Stime without 0 (ie. the set of strictly positive durations). Let Sprocedure be a denumerable set disjoint with Stoken and Stime, the elements of which are typically denoted by X and are called procedure variables. Define the language L as the set of agents A generated by the following grammar

\[
C ::= \text{tell}_d(t) \mid \text{ask}_d(t) \mid \text{get}_d(t) \mid \text{nask}_d(t) \mid \text{delay}(d) \\
A ::= C \mid A \mid A \parallel A \mid A + A \mid X
\]

where the durations d in the subscripts are not null.
As easily noticed by the careful reader, the communication primitives of the language $L$ are basically the Linda primitives equipped with time. Indeed, the Linda primitives put, in and rd for, respectively, putting an object $t$ in a shared dataspace, getting it and checking for its presence are renamed as tell, get and ask for compatibility with the syntax used in our previous publications. Moreover, a primitive $\text{nask}(t)$ has been added to test the absence of $t$ on the shared dataspace.

These primitives are enriched with durations (syntactically denoted by subscripts) with the following intuition:

- the execution of $\text{tell}_{d}(t)$ adds $t$ to the dataspace but for $d$ units of time only,
- if the execution of the $\text{ask}_{d}(t)$, $\text{nask}_{d}(t)$, and $\text{get}_{d}(t)$ primitives need to suspend (because of the non availability of $t$ for the ask and get primitives or because of the availability of $t$ for the nask primitive), this may only occur during $d$ units of time, after which the primitives fail (i.e. terminates without allowing subsequent computations to continue).

To these primitives is added the primitive $\text{delay}(d)$ whose purpose is to force time to pass by $d$ units of time.

The composition operators are the traditional ones in concurrency theory: $\cdot$, $\parallel$ and $+$ are used to respectively denote sequential composition, parallel composition and external choice. Finally, following [JL87], the letter $X$ is used to denote an abstraction of procedure call and to allow recursion. These procedure calls are defined as guarded agents by means of declarations, as follows.

**Definition 2.** Define the set $\mathcal{G}$ of guarded agents as the set of the agents $G$ given by the following grammar:

$$
C ::\;\text{tell}_{d}(t) \mid \text{ask}_{d}(t) \mid \text{get}_{d}(t) \mid \text{nask}_{d}(t) \mid \text{delay}(d)
$$

$$
A ::\; C \mid A \mid A \parallel A \mid A + A \mid X
$$

$$
G ::\; C \mid G \mid A \mid G \parallel G \mid G + G
$$

where the durations in the subscripts are not null.

**Definition 3.** A declaration $D$ is a list of associations $\langle X, G \rangle$ between procedure variables and guarded agents. Such a list may be infinite, so that $D$ is formally regarded as a mapping from procedure variables to guarded agents. For the ease of reading, we shall also rewrite $\langle X, G \rangle$ as $X = G$.

To simplify the notations, we shall subsequently assume a declaration $D$ to be given and will omit it when no confusion is introduced.

As easily observed from the above definitions, the main property of guarded agents is that any call to a procedure variable $X$ is always preceded by at least one communication primitive $C$. This (classical) property ensures that equations of the form $X = G$ are well-defined. Note that we do not require that procedure calls are preceded by a tick clock. For instance, $X = \text{tell}_{1}(t) \parallel X$ is allowed. It corresponds to a process that infinitively produces occurrences of the token $t$ in the same unit of time. Such behaviors are named Zeno-behaviors (see eg. [JL87]). They will be treated in a companion paper.

**Example 4.** An example may help to understand the above concepts. Let us code a producer process which recursively produces items outdated after two units of time and which takes a rest of five units of time after each item is produced. Such a process may be coded as follows, where $\text{Prod}$ represents the producer and $i$ the item being produced:

$$
\text{Prod} = \text{tell}_{2}(i) ; \text{delay}(5) ; \text{Prod}.
$$

It is worth observing the contents of the shared dataspace during the execution of $\text{Prod}$. Started at time 1 with an empty dataspace, $\text{Prod}$ first executes the primitives $\text{tell}_{2}(i)$ which puts $i$ for two units of time, namely until time 3. The execution of the primitive itself takes no time so that $\text{Prod}$ then executes the $\text{delay}(5)$ operation which forces it to stay idle until time 6. As a result, assuming $\text{Prod}$ is the only process being executed, the dataspace is again empty between times 3 and 6. This is summarized in the following picture where the contents of the dataspace is drawn in each interval of time after the execution of the $\text{tell}(i)$ primitive and where the subscript associated with $i$ denotes the current life of the item.

<table>
<thead>
<tr>
<th>{i}</th>
<th>{i}</th>
<th>\emptyset</th>
<th>\emptyset</th>
<th>\emptyset</th>
<th>{i}</th>
<th>{i}</th>
<th>\emptyset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Similarly, a process which recursively requests the item $i$ for one unit of time and takes two units of time to actually consume it may be coded as follows:

$$
\text{Cons} = \text{get}_{1}(i) ; \text{delay}(2) ; \text{Cons}.
$$

The computation of $\text{Prod} \parallel \text{Cons}$ is then worth observing. As its first action is to get $i$, the agent $\text{Cons}$ has to wait that $\text{Prod}$ has produced $i$. It can then get it and then calls itself recursively after having waited two units of time, namely at time 3. At that moment, $\text{Cons}$ tries to get $i$ again but for one unit of time only. However, this is too short since $\text{Prod}$ will only put a new occurrence of $i$ at time 6. As a result, the agent $\text{Cons}$ is then blocked for the rest of the computation.
To easily express successful termination, we shall introduce particular configurations composed of a special terminating symbol $E$ together with a multi-set of tokens. For uniformity purposes, we shall misuse language and qualify $E$ as an agent. However, to meet the intuition, we shall always rewrite agents of the form $(E;A)$, $(E||A)$, and $(A||E)$ as $A$. This is technically achieved by defining the extended set of agents as follows, and by justifying the simplifications by imposing a bimonoid structure. Finally, in contrast to the agents of Definition 1, we shall allow communication primitives associated with null durations.

**Definition 5.** Define the set of extended agents $\mathcal{L}^e$ as the set of the agents $A$ defined by the following grammar

\[
C ::= \text{tell}_d(t) | \text{ask}_d(t) | \text{get}_d(t) | \text{nask}_d(t) | \text{delay}(d)
\]

\[
A ::= C | E | A;A | A||A | A + A | X.
\]

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

As just illustrated by Example 4, tokens and primitives get older as time evolves. This is formally captured by the following definitions.

**Definition 6.**

1. Given an agent $A \in \mathcal{L}$, we denote by $A^-$ the agent defined inductively as follows:

\[
\text{delay}(d)^- = \text{delay}(\max(0,d-1))
\]

\[
\text{tell}_d(t)^- = \text{tell}_d(t)
\]

\[
\text{ask}_d(t)^- = \text{ask}_{\max(0,d-1)}(t)
\]

\[
\text{nask}_d(t)^- = \text{nask}_{\max(0,d-1)}(t)
\]

\[
\text{get}_d(t)^- = \text{get}_{\max(0,d-1)}(t)
\]

We extend this notation to $\mathcal{L}^e$ by stating that $E^- = E$.

2. Define the set of timed stores\(^2\) $\text{Ststore}$ as the set of multisets of elements of the form $t_d$ where $t$ is a token and $d$ is a duration. Given a timed store $\sigma$, we denote by $\sigma^-$ the new store obtained by decreasing the duration associated with the tokens by one unit and by removing those associated in $\sigma$ with 1 unit of time. More precisely, if all the notations are understood to relate to multi-sets,

\[
\sigma^- = \{t_{d-1} : t_d \in \sigma, d > 1\}.
\]

Moreover, we define by $\sigma^+$ the multiset of the untimed versions of the tokens appearing with a strictly positive duration in $\sigma$. More precisely,

\[
\sigma^+ = \{t : t_d \in \sigma, d > 0\}.
\]

According to the two-phase functioning approach, a temporal step will be done when no communication primitives can be executed. However, given that the execution of tell primitives can always proceed, a temporal step only makes sense if the agent under consideration offers the hope of an execution step in the future, namely if it contains in an executable position an ask, nask or get or primitive or delay associated with a non null duration. This is formally expressed by the two following definitions. Firstly, the set $\mathcal{F}(A)$ of the primitives in an executable position in agent $A$ is formalized and then the presence in $\mathcal{F}(A)$ of a primitive ask, nask, get or delay with a strictly positive duration in $\mathcal{F}(A)$ is captured by the introduction of the predicate $A \gg$.

**Definition 7.** Let $\text{Scom}$ denote the set of communication primitives $\text{tell}_d(t), \text{ask}_d(t), \text{get}_d(t), \text{nask}_d(t)$ and $\text{delay}(d)$ for any $d \in \text{Stime}$ and $t \in \text{Stoken}$. Define $\mathcal{F} : \mathcal{L}^e \rightarrow \mathcal{P}(\text{Scom})$ as the following function: for any communication primitive $c$, procedure variable $X$ defined in $D$ by the declaration $(X,G)$, and agents $A$ and $B$,

\[
\mathcal{F}(E) = \emptyset
\]

\[
\mathcal{F}(c) = \{c\}
\]

\[
\mathcal{F}(X) = \mathcal{F}(G)
\]

\[
\mathcal{F}(A;B) = \mathcal{F}(A) \cup \mathcal{F}(B)
\]

\[
\mathcal{F}(A||B) = \mathcal{F}(A) \cup \mathcal{F}(B).
\]

**Definition 8.** For any agent $A$, the predicate $A \gg$ holds iff the set $\mathcal{F}(A)$ contains at least one primitive ask, nask, get or delay associated with a non null duration.

**Example 9.** Consider the agents $\text{Prod}$ and $\text{Cons}$ of Example 4 at the end of the first unit of time. $\text{Prod}$ has become $\text{delay}(5)$; $\text{Prod}$ and $\text{Cons}$ has become $\text{delay}(2)$; $\text{Cons}$. Let us name these agents $\text{Prod}^\prime$ and $\text{Cons}^\prime$, respectively. Intuitively, both agents are worth being continued. This is met by the above formalization. Indeed, one has $\mathcal{F}(\text{Prod}^\prime || \text{Cons}^\prime) = \{\text{delay}(5), \text{delay}(2)\}$ and, consequently, $(\text{Prod}^\prime || \text{Cons}^\prime) \gg$. In contrast, consider $\text{Cons}$ alone after one unit of time. As no item $i$ is produced, $\text{Cons}$ has become $\text{get}_{t_0}(i)$; $\text{delay}(2)$; $\text{Cons}$. Let us denote by $\text{Cons}''$ this agent. One has $\mathcal{F}(\text{Cons}'') = \{\text{get}_{t_0}(i)\}$ and, accordingly, $\text{Cons}''' \gg$. This translates the fact that it is not interesting to continue the computation of $\text{Cons}$ (which will remain blocked forever).

---

1 For details about the algebraic structure see example [10].

2 We use store as a synonym for shared date space.
The computations in \( \mathcal{L}^e \) may be modeled by a transition system written in Plotkin’s style. Following the explanation given above, the configurations to be considered consist of an agent together with a multi-set of timed tokens, denoting the tokens currently available for the computation together with their durations.

The transition rules are the ones given in Fig. 1. An operational semantics may be defined directly from them by reporting the traces of all the computation steps made during the executions both in terms of the contents of the shared space and the moments of execution. Formally, it is specified in Definition 11 where \( \delta^+ \) and \( \delta^- \) are respectively used as ending marks respectively denoting successful and failing computations. Moreover, given a set \( S \), we respectively denote by \( S^\omega \) and \( S^* \) the set of infinite sequences and finite sequences composed from elements of \( S \).

**Definition 10.**

1. Define the set of configurations \( \text{Stconf} \) as \( \mathcal{L}^e \times \text{Ststore} \times \text{Stime} \). Configurations are denoted as \( \langle A \mid \sigma \rangle_u \), where \( A \) is an agent, \( \sigma \) is a timed store of \( \text{Ststore} \) and \( u \) is a time.

2. Define the set \( \text{Sthist} \) as the set \( \text{Ststore}^\omega \cup \text{Ststore}^* \cdot \{\delta^+, \delta^-\} \).

**Definition 11.** Define the operational semantics \( \mathcal{O}_h : \mathcal{L} \rightarrow \mathcal{P}(\text{Sthist}) \) as the following function. For any agent \( A \),

\[
\mathcal{O}_h(A) = \{ \sigma_0^{[0]} \ldots \sigma_n^{[n]} \cdot \delta^+ : (A_0 | \sigma_0^{[0]} \cdot \cdots \cdot \sigma_n^{[n]} \cdot \delta^+ \rightarrow \cdots \rightarrow (A_n | \sigma_n^{[n]}),
A_0 = A, \sigma_0 = \emptyset, u_0 = 1, A_n = E, n \geq 0 \}
\]

\[
\cup \{ \sigma_0^{[0]} \ldots \sigma_n^{[n]} \cdot \delta^- : (A_0 | \sigma_0^{[0]} \cdot \cdots \cdot \sigma_n^{[n]} \cdot \delta^- \rightarrow \cdots \rightarrow (A_n | \sigma_n^{[n]} \not\rightarrow),
A_0 = A, \sigma_0 = \emptyset, u_0 = 1, A_n \neq E, n \geq 0 \}
\]

\[
\cup \{ \sigma_0^{[0]} \ldots \sigma_n^{[n]} \ldots : (A_0 | \sigma_0^{[0]} \cdot \cdots \cdot (A_n | \sigma_n^{[n]} \rightarrow \cdots \rightarrow, A_0 = A, \sigma_0 = \emptyset, u_0 = 1, \forall n \geq 0 : A_n \neq E \}
\]

where the \( \rightarrow \) arrow denotes either a \( \rightarrow \) transition or a \( \sim \) transition.

Note that the behavior of (untimed) Linda primitives is obtained as a particular case of the transitions where all the communication primitives are associated with a 1 time. Our language \( \mathcal{L} \) thus subsumes Linda and, consequently, the refinement theory we shall develop applies to Linda-like languages as well.
3. Reactive sequences instead of trace and failure sets

In the traditional lines, exemplified by [1,28], one may define a first notion of refinement by stating that an implementation I refines a specification S (both I and S being agents of the \( \mathcal{L} \) language) if any trace of execution made by I can also be made by S, namely if \( \Theta_b(I) \subseteq \Theta_b(S) \).

Unfortunately, trace refinement takes only into account the actual traces computed by the agents regardless of their possible interactions with an environment. As a result, trace refinement does not enjoy the required property of substitutability.

Consider for instance \( A = \text{get}_1(a) \) and \( B = \text{get}_1(b) \). One has \( \Theta_b(A) = \{ h_1, \delta^- \} = \Theta_b(B) \) and, consequently, \( A \) trace-refines \( B \). Consider now \( C = \text{tell}_1(b) \). The parallel composition \( B \parallel C \) has just one successful computation \( h_1, \{ b_1 \}, h_1, \delta^+ \) whereas the alternative parallel composition \( A \parallel C \) has just one failing computation \( h_1, \{ b_1 \}, h_2, \delta^- \). However, if trace refinement would enjoy the substitutibility property, given that \( B \parallel C \) is deadlock free, then \( A \parallel C \) should also be deadlock free.

This phenomenon is quite well known in concurrency theory and the solution is to use a failure semantics instead of a trace semantics. Intuitively, the idea is to complete the description of a trace ending in a deadlock by a description of the actions that the process can refuse. In other terms, if the environment offers the actions of the refusal set then the environment in parallel with the considered agent is still in a deadlock state. In the above example, this would allow us to distinguish the reason of the deadlock of \( A \) and \( B \): the former deadlocks because of the absence of the token \( a \) on the shared space whereas the latter deadlocks because of the absence of the token \( b \) on the shared space. As the deadlocks have different explanations, the failure semantics of \( A \) and \( B \) would be different and, consequently, one would not have that \( A \) refines \( B \).

Actually, refusal sets are the basis of fully abstract semantics for CCS and CSP like languages as well as for the development of tools like the FDR model checker (see [28]).

However, although intuitive, refusal sets, also called failures, have at least three drawbacks which prevent us from a direct transposition in our time coordination setting and, more generally, for Linda-like languages.

First, traditional algebras such as CCS and CSP are based on the synchronous communication of events. In this context, it is reasonable to build refusal sets from actions which cannot be made by the agent under consideration even if they are offered by the environment. However, in the \( \mathcal{L} \) context and more generally for Linda-like languages, what matters is not so much which actions are made but more importantly the current contents of the shared dataspace. Therefore, in contrast to actions of which many can be offered by the environment, only one contents of the shared space is of interest, namely the minimal one which allows computations to be continued. This has lead us in previous work to consider reactive sequences to build fully abstract semantics of Linda-like languages (see [3,4]) and, basically amounts to abandoning the idea of failures.

Second, the nature of the choice operator in the synchronous setting of CCS and CSP defines the failure sets of \( A + B \) as the union of the failure sets of \( A \) and \( B \). Rephrased in our setting, this would lead to consider that \( \text{ask}_1(a) \) failure-refines \( \text{ask}_1(a) + \text{tell}_1(b) \). However, \( \text{ask}_1(a) \) waits for one unit of time before deadlocking whereas \( \text{ask}_1(a) + \text{tell}_1(b) \) has no deadlock.

Third, as failures are traditionally defined in an untimed setting, there is no provision for time. One could think of using timed failure sets defined in [29] but, in view of the above two points, we prefer to define directly our notion of refinement. To that end, to give a complete picture with the traditional concurrency setting, we first define a fully abstract semantics, counterpart of the failure sets semantics for languages such as CCS and CSP. We then relate it with an event based semantics, which will provide the foundations for the refinement theory.

4. Fully abstract semantics

As mentioned above, we already studied fully abstract semantics for Linda-like languages in [3,4]. These languages incorporated untimed versions of the primitives exposed in Section 2. We show in this section how they can be extended to the timed version. However, to ease the presentation, we shall only concentrate on the finite sublanguage, thus discarding recursive calls. Recursion can nevertheless be treated by using suitable contractions on complete metric spaces as is illustrated in [3,4]. The interested reader can refer to [19] where all the details are exposed.

Our presentation is structured as follows. The semantic domain is first specified in Section 4.1. It describes which sequences of computation steps are to be considered. Next, the semantics of the primitives is provided in Section 4.2 and the semantics of the composition operators is presented in Section 4.3. The denotational semantics to be considered is then defined in Section 4.4 and proved correct with respect to the operational semantics in Section 4.5. Finally full abstraction is established in Section 4.6.

4.1. Semantic domain

Following [15], the main idea is to model transition steps in the form of pairs of input and output situations and take as semantic domain sets of sequences of such pairs. These sequences possibly contain gaps, accounting for actions of the environment. Moreover, they start in any situation and at any time, allowing previous steps to result in a possibly non-empty store and at a non-initial time.

Such sequences are called histories. As the language under consideration in this section only contains finite agents, the histories to be considered here are finite and are from the following set

\[
((St_{\text{store}} \times St_{\text{time}}) \times (St_{\text{store}} \times St_{\text{time}}))^* . ((St_{\text{store}} \times St_{\text{time}}) \times \{ \delta^+, \delta^- \}) .
\]
The set of histories associated with each agent and the way to compose sets of histories will be the topics of the next subsections. Let us however first examine here intuitively the meaning of the presence of a history $h$ in a set $S$ regarded as the denotational semantics of an agent $A$.

A history $h$ in $S$ can be seen as a sequence of steps for which the agent $A$ can be responsible if it computes in an environment whose evolution is able to fill the “gaps” in the history $h$. We consider an agent as responsible for a step if this step corresponds to a transition of the computation of the agent. More specifically, we need to distinguish the cases of time-constant steps and time-growing steps. Let us decompose the history respectively as $h = p.(\sigma_u, \tau_u).h'$ or as $h = p.(\sigma_u, \sigma_{u+1}).h'$ for some history prefix $p$ and history $h'$.

As regards time-constant steps, the presence in $S$ of a history $h$ having as first step $(\sigma_u, \tau_u)$ corresponds to the transition $\langle A \mid \sigma \rangle_u \rightarrow \langle A' \mid \tau \rangle_u$, performed by $A$ with $A' \in \mathcal{L}$. More generally, any step $(\sigma_u, \tau_u)$ occurring in $h$ can be interpreted similarly. The intuition is as follows. There is a particular environment in which agent $A$ will be responsible for the sequence of transitions described by the steps of $p$. Let $B$ be the agent resulting from the evolution of $A$ along these transitions. After the last step of $p$, if the environment leads the system into the situation $\sigma_u$, agent $B$ will be responsible for the step $(\sigma_u, \tau_u)$ corresponding to the firing of the transition $\langle B \mid \sigma \rangle_u \rightarrow \langle B' \mid \tau \rangle_u$ for some agent $B' \in \mathcal{L}$.

As regards time-growing steps, an agent $A$ will be considered as responsible for a step $(\sigma_u, \sigma_{-u+1})$ if two conditions are satisfied, which are the two conditions allowing to fire the transition $\langle A \mid \sigma \rangle_u \rightarrow \langle A^- \mid \sigma^- \rangle_{u+1}$: to be in a “blocked” situation (i.e. $A \sigma \delta \delta$), with an agent not already out-of-date (i.e. $A \gg$).

To conclude, a word on the last element of histories is in order. Let $h = h'$. $(\sigma_u, \delta)$ be such a history. The last element of the sequence has to be considered after the computation of all the transitions corresponding to the steps of $h'$. The situation $\sigma_u$ denotes a situation that can then be reached by the complete system, in particular, comprising agent $A$ at the time $u$ of the last step of $h'$. If this situation is reached by the system after the computation of all the steps in $h'$, the agent $A$ will be responsible for the computation of no additional step and its computation has to be considered as terminated. The symbol $\delta$ indicates this termination with, as before, $\delta^+$ denoting a successful end (i.e. $A$ has evolved to $E$) and $\delta^-$ denoting a failure (i.e. agent $A$ has not evolved to $E$ but is not able to continue its computation on the stable situation reached by the complete system).

The above description leads to the observation that the histories of a semantical set have some particularities. Firstly, two kinds of step can be identified. On the one hand, computational steps are time-constant, and, on the other hand, time-growing steps increase the time by one unit and modify the store by decreasing the duration of its tokens by one unit. Secondly, any update of time is expected to be expressed in an history by the presence of a temporal step in it. Consequently, from the time point of view, histories will be continuous. Stated in other words, one step has to start at the time on which the previous step terminates. This property is captured through the $t$-continuity notion, while a continuous history is an history in which every step starts both at the time and on the store on which the previous terminates. Thanks to these observations, the set of histories needs to be defined as follows.

**Definition 12.**

1. Define the set of steps $S_{\text{step}}$ as the following set

   \[ S_{\text{step}} = \{(\sigma_i, \tau_i) \in (S_{\text{store}} \times S_{\text{time}}) \times (S_{\text{store}} \times S_{\text{time}})\} \cup \{(\sigma_i, \sigma_{i+1}^-) \in (S_{\text{store}} \times S_{\text{time}}) \times (S_{\text{store}} \times S_{\text{time}})\}. \]

2. Define the set of denotational histories $S_{\text{hist}}$ as the set

   \[ S_{\text{hist}} = S_{\text{step}}^+ \times (S_{\text{store}} \times S_{\text{time}})^* \times (\delta^+, \delta^-). \]

3. Let $h = (\sigma_1^1, \tau_1^1), (\sigma_2^2, \tau_2^2), \ldots, (\sigma_{n-1}^{n-1}, \tau_{n-1}^{n-1}), (\sigma_n, \delta)$ be an history of $S_{\text{hist}}$. Define

   \[ \text{init}(h) = \sigma_1^1 \]

   \[ \text{step}(h) = (\sigma_i^i, \tau_i^i) \quad \text{for} \quad i = 1, \ldots, n-1 \]

   \[ \text{length}(h) = n. \]

Moreover, let $S$ be a subset of $S_{\text{hist}}$ and $p$ a sequence of $(S_{\text{store}} \times S_{\text{store}})^*$, then define

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

\[ S^+ = \{h : h = (\sigma_i, \delta^+) \in S\} \]

\[ S^- = \{h : h = (\sigma_i, \delta^-) \in S\} \]

\[ S^a = \{h : h = (\sigma_i.t).h' \in S\} \]

\[ S(\sigma_i) = \{h \in S : \text{init}(h) = \sigma_i\}. \]

4. The history $h = (\sigma_1^1, \tau_1^1), (\sigma_2^2, \tau_2^2), \ldots, (\sigma_{n-1}^{n-1}, \tau_{n-1}^{n-1}), (\sigma_n, \delta)$ is $t$-continuous iff for any $i = 1, \ldots, n-1$, it holds that $u_i = t_{i+1}$.

5. The history $h = (\sigma_1^1, \tau_1^1)(\sigma_2^2, \tau_2^2), \ldots, (\sigma_{n-1}^{n-1}, \tau_{n-1}^{n-1}), (\sigma_n, \delta)$ is continuous iff for any $i = 1, \ldots, n-1$, it holds that $t_i^i = \sigma_i^{i+1}$.
(6) Define $S^\text{hist}$ as the following set

$$S^\text{hist} = \{ h \in \text{Step}^* . ((\text{Ststore} \times \text{Stime}) \times \{ \delta^+, \delta^- \}) : h \text{ is } t\text{-continuous} \}.$$ 

As agents considered in this section are finite and since the histories associated with their denotational semantics only report their actions, it is expected that the histories to be considered have their length bounded. This is formally translated by the following definition of $\ell$-bounded sets of histories.

**Definition 13.** The set $S$ of histories is said to be $\ell$-bounded iff there exists a finite length $\ell$ such that for any history $h$ in $S$, it holds that $\text{length}(h) \leq \ell$.

The semantic domain for $\mathcal{L}$ can then be defined as follows.

**Definition 14.** Define the semantic domain of $\mathcal{L}$ as

$$\mathcal{H} = \{ S \in \mathcal{P}(S^\text{hist}) : S \text{ is } \ell\text{-bounded} \}.$$ 

Before going on with the definition of the semantics, let us introduce some technical but important properties required from denotational sets. The property of uniform non-emptiness at level $n$ expresses that, after the computation of any prefix of length $n$, all the possible behaviors of the environment are accepted. This is captured by the fact that any prefix of length $n$ admits an extension starting on any store. Moreover, we distinguish two particular categories of sets among those which are uniformly non-empty at every level $n > 1$. The distinction is made according to the sets of the starting situations of their histories. Firstly, we call uniformly non-empty on time $t$, those sets which contain only histories starting at the given time $t$ and which contain at least one history starting on any store at time $t$. Secondly, sets are said uniformly non-empty if they contain at least one history starting on any store at any time $t$.

**Definition 15.** Let $S$ be a subset of $S^\text{hist}$.

1. The set $S$ is uniformly non-empty at level $n > 0$ iff for any $p \in (\text{Step})^n$ such that $\text{step}_n(p) = (\sigma_u, \tau_u)$, if $S[p] \neq \emptyset$ then for any $\mu \in \text{Ststore}$ it holds that $S[p](\mu_u) \neq \emptyset$.
2. The set $S$ is uniformly non-empty on time $t$ iff it is uniformly non-empty at every level $n > 0$, involves only histories beginning at time $t$ and for any $\sigma$ in $\text{Ststore}$, it holds that $S(\sigma) \neq \emptyset$.
3. The set $S$ is uniformly non-empty if it is uniformly non-empty at every level $n > 0$, and for any $\sigma$ in $\text{Ststore}$ and any time $t$ in $\text{Stime}_0$, it holds that $S(\sigma) \neq \emptyset$.

Let us illustrate these definition by some examples.

**Example 16.** The set

$$S_1 = \{ h = (\phi_u, \{ t_1 \}_u). (\sigma_u, \delta^+) : h \in S^\text{hist} \}$$

is uniformly non-empty at level 1, and at any level $n > 0$ while

$$S_2 = \{ h = (\phi_u, \{ t_1 \}_u). (\{ t_1 \}_u, \delta^+) : h \in S^\text{hist} \}$$

is not uniformly non-empty.

Indeed,

$$S_2[(\emptyset, \{ t_1 \}_1)] \neq \emptyset \text{ and } S_2[(\emptyset, \{ t_1 \}_1), (\emptyset_1) = \emptyset.$$ 

It is worth noting that the set $S_1$ defined above is not uniformly non empty because it contains only histories starting on the empty store. The set

$$S_3 = \{ h = (\sigma_u, (\sigma \cup \{ t_1 \}_u). (\tau_u, \delta^+) : h \in S^\text{hist} \}$$

is uniformly non-empty, while

$$S_4 = \{ h = (\sigma_2, (\sigma \cup \{ t_1 \}_2). (\tau_2, \delta^+) : h \in S^\text{hist} \}$$

is uniformly non-empty at time 2.

Several properties are worth being observed.

**Lemma 17.**

1. Let $S$ be an uniformly non-empty $\ell$-bounded set of $\mathcal{P}(S^\text{hist})$. For any store $\sigma \in \text{Ststore}$ and any time $u$, there is a continuous history $h \in S$ such that $\text{init}(h) = \sigma_u$.
2. Let $S$ be an $\ell$-bounded set of $\mathcal{P}(S^\text{hist})$ uniformly non-empty on time $t$. For any store $\sigma \in \text{Ststore}$, there is a continuous history $h \in S$ such that $\text{init}(h) = \sigma_u$.
3. Let $S$ be an $\ell$-bounded subset of $\mathcal{P}(S^\text{hist})$, uniformly non empty at every level $n > 0$. Let $p = p'.(\sigma_u, \tau_u)$ be in $\text{Step}^n$ for $n \geq 1$ such that $S[p] \neq \emptyset$, and $\lambda$ be a store of $\text{Ststore}$. Then, there is a continuous history $h$ in $S[p]$ such that $\text{init}(h) = \lambda_v$ and $p.h$ is in $S$.

**Proof.** Simple verification. □
4.2. Semantics of primitives

Defining a compositional semantics consists, on the one hand, in specifying the meaning of elementary statements and, on the other hand, in providing an operator at the semantic level for each syntactic operator. We start with the first task in this subsection. The second one will be discussed in the next subsection.

There are five primitives: \emph{tell}, \emph{ask}, \emph{mask}, \emph{get} and \emph{delay}. The following subsections examine them each in turn.

4.2.1. Denotational semantics for \emph{tell}_d(t)

To start with, let us define the set of tell primitives to consider.

\textbf{Definition 18.} Define \( \mathcal{L}_{\text{tell}} \) as the following set

\[
\mathcal{L}_{\text{tell}} = \{ \text{tell}_d(t) : d \in \text{Stime}_0 \text{ and } t \in \text{Stoken} \}.
\]

Histories in the denotational semantics of a \emph{tell}_d(t) primitive for a strictly positive time duration \( d \) are composed of two steps. The first step corresponds to the transition resulting in adding the token to the store. As such a history corresponds to a successful computation, the second step is marked with \( \delta^+ \). However, there is one restriction on the situation of this second step: the time has to respect the \( t \)-continuous property.

This remark leads to the following definition of the denotational semantics for the \emph{tell}_d(t) primitive.

\textbf{Definition 19.} Define \( \mathcal{D}_t : \mathcal{L}_{\text{tell}} \to \mathcal{H} \) as follows: for any finite \( d \) in \( \text{Stime}_0 \) and any token \( t \),

\[
\mathcal{D}_t(\text{tell}_d(t)) = \{ (\sigma_u, \sigma \cup \{ t_d \}_u, (\tau_u, \delta^+) : \exists h \in \text{Sthist}, t > 0 \}
\]

Given this intuitively founded definition, let us now verify some of its formal property.

\textbf{Proposition 20.} For any agent \( A \) in \( \mathcal{L}_{\text{tell}} \), \( \mathcal{D}_t(A) \) is well defined and is uniformly non empty.

\textbf{Proof.} Direct from the definition. \( \square \)

4.2.2. Denotational semantics for \emph{ask}_d(t)

Let us now consider the set of the \emph{ask}_d(t) primitives.

\textbf{Definition 21.} Define \( \mathcal{L}_{\text{ask}} \) as follows:

\[
\mathcal{L}_{\text{ask}} = \{ \text{ask}_d(t) : d \in \text{Stime}_0 \text{ and } t \in \text{Stoken} \}.
\]

The primitive \emph{ask}_1(t) succeeds on an initial situation \( \sigma_u \) such that the token \( t \) is in \( \sigma^* \). These computations correspond to histories composed of two steps. The first one keeps the store and the time unchanged. The second one expresses the success on any situation satisfying the \( t \)-continuous property.

In the case of an initial situation on which \( \text{ask}_1(t) \) does not compute – i.e. a situation \( \sigma_u \) such that the token \( t \) is not in \( \sigma^* \) – one cannot deduce directly a failure. Indeed, the agent \( \text{ask}_1(t) \) satisfies the condition \( (\text{ask}_1(t))^- \neq \text{ask}_1(t) \). In this case, the first step of the history corresponds to the transition \( (\text{ask}_1(t) \mid \sigma_u) \leadsto (\text{ask}_0(t) \mid \sigma_u) \). After that, the second step expresses the failure on any situation satisfying the \( t \)-continuous property.

Those two cases are respectively captured by the two sets occurring in the following definition.

\textbf{Definition 22.} Define \( \mathcal{D}_d(t) : \{ \text{ask}_1(t) : t \in \text{Stoken} \} \to \mathcal{H} \) as follows: for any token \( t \),

\[
\mathcal{D}_d(t)(\text{ask}_1(t)) = \left\{ h = (\sigma \cup \{ \tau \}_u, \sigma \cup \{ \tau \}_u, (\tau_u, \delta^+) : h \in \text{Sthist}, t > 0 \right\}
\]

For any \( d > 1 \), the primitive \emph{ask}_d(t) may succeed after the first temporal transition. If not, a second temporal step is introduced and so on for \( d \) units of time. This leads to the inductive definition of the semantics of the \emph{ask}_d primitive.

\textbf{Definition 23.} Define the denotational semantics for the \emph{ask} primitives \( \mathcal{D}_d : \mathcal{L}_{\text{ask}} \to \mathcal{H} \) as follows: For any token \( t \) and any finite duration \( d > 1 \),

\[
\mathcal{D}_d(\text{ask}_1(t)) = \mathcal{D}_d(\text{ask}_1(t))
\]

\[
\mathcal{D}_d(\text{ask}_d(t)) = \left\{ h = (\sigma \cup \{ \tau \}_u, \sigma \cup \{ \tau \}_u, (\tau_u, \delta^+) : h \in \text{Sthist}, t > 0 \right\}
\]

Note that in the above definition the condition that \( h \in \text{Sthist} \) not only requires that stores are in \( \text{Ststore} \) and times in \( \text{Stime} \) but also ensures that histories in the resulting set are \( t \)-continuous.

\textbf{Proposition 24.} For any agent \( A \in \mathcal{L}_{\text{ask}} \), \( \mathcal{D}_d(A) \) is well defined and is uniformly non-empty.

\textbf{Proof.} Direct from the definition. \( \square \)
4.2.3. Denotational semantics for naskₜ(t)

The behavior of naskₜ(t) is similar to the behavior of the askₜ(t) primitive except for the fact that success is obtained in the case of the absence instead of the presence of the token t and failure is obtained in the case of presence of the token t instead of its absence.

As before, the denotational semantics is provided in an inductive way.

Definition 25. Define \( \mathcal{L}_{\text{nask}} \) as follows:

\[
\mathcal{L}_{\text{nask}} = \{ \text{nask}_t(t) : d \in \text{Stime}_0 \text{ and } t \in \text{Stoken} \}.
\]

Definition 26. Define the denotational semantics for the nask primitives \( \mathcal{D}_n : \mathcal{L}_{\text{nask}} \to \mathcal{H} \) as follows: For any token t and any finite duration \( d > 1 \),

\[
\mathcal{D}_n(\text{nask}_1(t)) = \{(\sigma_u, \sigma_u, \tau_u, \delta^+) \in \text{Sthist} : t \notin \sigma^*\} \\
\cup \\
\{(\sigma \cup \{t_u\}, \sigma^-, \tau_u, \delta^-) \in \text{Sthist} : v > 0\}
\]

\[
\mathcal{D}_n(\text{nask}_d(t)) = \{h = (\sigma_u, \sigma_u, \tau_u, \delta^+) \in \text{Sthist} : t \notin \sigma^*\} \\
\cup \\
\{h \in (\sigma \cup \{t_u\}, \sigma^- \cup \{t_u\}, \delta^-) \in \text{Sthist} : v > 0\}.
\]

Proposition 27. For any agent \( A \in \mathcal{L}_{\text{nask}} \), \( \mathcal{D}_n(A) \) is well defined and is uniformly non-empty.

Proof. Direct from the definition. □

4.2.4. Denotational semantics for getₜ(t)

Whatever the duration \( d \) is, the getₜ(t) primitive behaves similarly to the primitive askₜ(t). The only difference appears in the fact that, in case of success, the last but one step erases one occurrence of the token of the store.

Definition 28. Define \( \mathcal{L}_{\text{get}} \) as the following set

\[
\mathcal{L}_{\text{get}} = \{ \text{get}_t(t) : d \in \text{Stime}_0 \text{ and } t \in \text{Stoken} \}.
\]

Definition 29. Define the denotational semantics for the get primitives \( \mathcal{D}_g : \mathcal{L}_{\text{get}} \to \mathcal{H} \) as follows: For any token t and any finite duration \( d > 1 \),

\[
\mathcal{D}_g(\text{get}_1(t)) = \{(\sigma \cup \{t_u\}, \sigma_u, \tau_u, \delta^+) \in \text{Sthist} : v > 0\} \\
\cup \\
\{h = (\sigma_u, \sigma_u, \tau_u, \delta^+) \in \text{Sthist} : t \notin \sigma^*\}
\]

\[
\mathcal{D}_g(\text{get}_d(t)) = \{h \in (\sigma \cup \{t_u\}, \sigma^-, \tau_u, \delta^-) \in \text{Sthist} : v > 0\} \\
\cup \\
\{h \in (\sigma_u, \sigma^- \cup \{t_u\}, \delta^-) \in \text{Sthist} : t \notin \sigma^*\}.
\]

Proposition 30. For any agent \( A \in \mathcal{L}_{\text{get}} \), \( \mathcal{D}_g(A) \) is well defined and is uniformly non-empty.

Proof. Direct from the definition. □

4.2.5. Denotational semantics for delay(d)

After the four communication primitives, let us consider here the delay primitive.

Definition 31. Define \( \mathcal{L}_{\text{delay}} \) as the following set

\[
\mathcal{L}_{\text{delay}} = \{ \text{delay}(d) : d \in \text{Stime} \}.
\]

For any store and for any time at which its computation starts, the computation of the delay(d) primitive consists of d temporal steps followed by one step leaving the store and the time unmodified. The computation then succeeds immediately.

Definition 32. Define \( \mathcal{D}_d : \mathcal{L}_{\text{delay}} \to \mathcal{H} \) as follows: for any \( d > 0 \) in \( \text{Stime} \),

\[
\mathcal{D}(\text{delay}(0)) = \{(\sigma_0, \sigma_0, \sigma_0, \delta^+) \in \text{Sthist}\}
\]

\[
\mathcal{D}(\text{delay}(d)) = \{(\sigma_0, (\sigma_0)^{d+1}, \cdots, (\sigma_0)^{d+1} \cup \{d+1\}, \delta^+) \in \text{Sthist}\}.
\]

Proposition 33. For any \( A \in \mathcal{L}_{\text{delay}} \), \( \mathcal{D}_d(A) \) is well defined and is uniformly non-empty.

Proof. Direct from the definition. □
4.3. Semantical operators

Our intention is to provide a compositional semantics. This task requires to associate an operator on the semantical domain with each syntactic operator. This section is dedicated to the definition of these semantical operators. There are three syntactic operators to combine elementary agents: sequential composition, parallel composition and choice. We subsequently examine each of them in turn.

4.3.1. Sequential composition

The computation of the sequential composition of two agents begins with the computation of the first agent. In case of success, its computation is followed by the computation of the second agent. Similarly, the sequential composition of two histories consists of the concatenation of the steps of the first histories and, if the first one finishes with success, the steps of the second one. By extension, the sequential composition of two sets of histories consists of the set obtained by the sequential composition of any history of the first set with any history of the second one. However, this set has to preserve the two important properties we distinguished before: its histories have to be t-continuous and set itself has to be uniformly non-empty.

**Definition 34.** Define \( \tilde{\cdot} : \mathcal{H} \times \mathcal{H} \rightarrow \mathcal{H} \) as the following function: for any element \( S_1, S_2 \) of \( \mathcal{H} \),

\[
S_1 \tilde{\cdot} S_2 = \{ h = h_1.(s, \delta^-) \in S_1 \} \\
\cup \{ h = h_1.h_2 : h_1.(s, \delta^+) \in S_1 \land h_2 \in S_2, \text{ and } h \text{ is } t\text{-continuous} \}.
\]

**Lemma 35.**

(1) The function \( \tilde{\cdot} \) is well-defined.

(2) For any sets \( S_1, S_2 \) in \( \mathcal{H} \) uniformly non-empty, \( S_1 \tilde{\cdot} S_2 \) is uniformly non-empty.

**Proof.** Simple verification. □

4.3.2. Parallel composition

The definition of the semantic operator \( \parallel \) corresponding to the syntactical operator \( || \) is guided by the following intuition. The parallel composition of two sets \( S_1 \) and \( S_2 \) is the union of three types of sets.

(1) The first set of the union corresponds to histories beginning by a time constant step \((s, s')\) of a history of the first set and followed by a history of the parallel composition of the rest of the first set, namely \( S_1[(s, s')] \), and the second set \( S_2 \).

Another set of the union corresponds to the converse obtained by switching the roles of \( S_1 \) and \( S_2 \).

(2) The second type of sets of the union corresponds to histories beginning with a time-growing step. There are three sets of this type. In the first one, the two sets have to follow the temporal transition simultaneously.

The second set and the converse third set correspond to the composition of the terminated histories of one set with the histories starting by a temporal step of the other set. Those compositions correspond to the following intuition. On the one hand, a history is terminating on the store \( \sigma \) at time \( u \), which means “on this given store, at that given time, I’m not able to fire any transition nor a computational nor a temporal one”. On the other hand, an history is beginning with a temporal step on the store \( \sigma \) at time \( u \). According to the fact that temporal steps are only fired if no computational transition can be fired, this means “on this given store, at that given time, I’m blocked, but I’m interested in firing a temporal step”. When those two histories are put in parallel, the first one answer “well, you can go, for me the game is over”. Moreover, if it fails, it adds “do not forget that I fail, consequently our composition will be a failing one”.

(3) The third type of set is the composition of the terminating transitions. Two terminating histories can be combined only if they are terminating on the same store, at the same time. The combination of \( \delta \) symbols is easy to understand. The composition is successful if and only if the two composed elements are successful. If at least one of the two finite histories fails, the parallel composition also fails.

This informal presentation is captured in the following definition.

**Definition 36.** Define \( \widetilde{\cdot} : \mathcal{H} \times \mathcal{H} \rightarrow \mathcal{H} \) as follows for \( S_1, S_2 \in \mathcal{H} \),

\[
S_1 \widetilde{\cdot} S_2 = \psi^i(S_1, S_2) \cup \psi^i(S_2, S_1) \\
\cup \psi^t(S_1, S_2) \cup \tau(S_1, S_2) \cup \tau(S_2, S_1) \\
\cup (S_1 \# S_2)
\]
The well-definedness property is direct from the definition. Let us turn to the second property.

Definition

In that sub-case, the history $h'$ according to the set that involves results.

Intermediate result

For any sets $S_1$ and $S_2$ are $\ell$-bounded, denote by $M_1$ the maximal length of histories in $S_1$, and $M_2$ the maximal length of histories in $S_2$. According to the fact that $S_1$ and $S_2$ are uniformly non-empty, $M_1$ and $M_2$ are greater or equal to 1.

The proof is by induction on $M_1 + M_2$. The basic case, where $M_1 + M_2 = 2$, is a direct consequence of the uniform non-emptiness of $S_1$ and $S_2$ and the Definition 36 of $\parallel$.

Unfortunately the induction cannot be established directly. Indeed, if $S_1$ is uniformly non-empty $S_1[(\sigma_v, \tau_v)]$ involves only histories beginning at time $v$ and is not uniformly non-empty. However, it is uniformly non-empty at time $v$.

Consequently, the proof starts by establishing the following result.

Intermediate result 1.

For any sets $T_1$ and $T_2$ in $\mathcal{H}$ uniformly non-empty at time $u$, the composition $T_1 \parallel T_2$ is uniformly non-empty at time $u$.

This can easily be established by induction on the sum of the maximal length of histories in $T_1$ and $T_2$.

Using this first result and a similar inductive reasoning, we can also establish the following second intermediate result.

Intermediate result 2.

For any sets $V_1$ in $\mathcal{H}$ uniformly non-empty at time $u$, and $V_2$ in $\mathcal{H}$ uniformly non-empty, the composition $V_1 \parallel V_2$ is uniformly non-empty at time $u$.

The general case on $M_1 + M_2$ can be established using those results.

The non-emptiness of $(S_1 \parallel S_2)(\sigma_u)$ for any store $\sigma$ and any time $u$ is direct from the definition of $\parallel$ and the intermediate results.

Let us establish the uniform non-emptiness at level $n$. Let $p$ be a sequence of steps of length $n$ such that $(S_1 \parallel S_2)[p] \neq \emptyset$ and $v$ the final time of $p$. We have to establish that for any store $\sigma$,

$$(S_1 \parallel S_2)[p](\sigma_v) \neq \emptyset.$$ 

Given the non-emptiness of $(S_1 \parallel S_2)[p]$, one can select one history $h$ in $S_1 \parallel S_2$ which can be written $h = p.q.(\rho_1, \delta)$. According to Definition 36 of $\parallel$, we distinguish the following cases and sub-cases.

Case 1. The first step of $p$ is computational.

In this case $p$ can be written $p = (\lambda_v, \mu_v).p'$. We distinguish two sub-cases following the set of the definition of $S_1 \parallel S_2$ which involves $h$.

Sub-case i. $h \in (\lambda_v, \mu_v).S_1[(\lambda_v, \mu_v)] \parallel S_2$. In that sub-case, the history $h' = p'.q.(\rho_1, \delta)$ is in $(S_1[(\lambda_v, \mu_v)] \parallel S_2).$ The second intermediate result provides an history $g' = p'.q' \in (S_1[(\lambda_v, \mu_v)] \parallel S_2)$ such that $\text{init}(q') = \sigma_v$. Consequently, the history $g = p.q'$ is in $S_1 \parallel S_2$ which succeeds.

Sub-case ii. $h \in (\lambda_v, \mu_v).S_2[(\lambda_v, \mu_v)] \parallel S_1$. This sub-case is treated similarly to the previous one.

Case 2. The first step of $p$ is temporal.

Using the notation introduced in Definition 36, we distinguish three sub-cases according to the set that involves $h$.

Sub-case i. $h \in \Psi'(S_1, S_2)$. Stated in other terms, one has

$$h \in (\lambda_v, \lambda_{v-1}).S_1[(\lambda_v, \lambda_{v-1})] \parallel S_2[(\lambda_v, \lambda_{v-1})].$$

In that sub-case, the history $h' = p'.q.(\rho_1, \delta)$ is in $(S_1[(\lambda_v, \lambda_{v-1})] \parallel S_2[(\lambda_v, \lambda_{v-1})]).$ The first intermediate result provides a history $g' = p'.q' \in (S_1[(\lambda_v, \lambda_{v-1})] \parallel S_2[(\lambda_v, \lambda_{v-1})])$ such that $\text{init}(q') = \sigma_v$. Consequently, the history $g = p.q'$ is in $S_1 \parallel S_2$ which succeeds.

Sub-case ii. $h \in \tau(S_1, S_2)$. In that sub-case, one has $h = (\lambda_v, \lambda_{v-1}).h'.(\tau_1, \delta), h_1 = (\lambda_v, \lambda_{v-1}).h'.(\tau_1, \delta_1)$ in $S_1$ and $h_2 = (\lambda_v, \delta_2)$ in $S_2$ with $\delta_1 = \delta_2 = \delta^+$ if $\delta = \delta^+$ and at least one of them equals to $\delta^-$ if $\delta = \delta^-$. In that sub-case, the
uniform non-emptiness of $S_1$ provides a history $g'$ in $S_1$ that can be written $g' = \sigma'q'$ with $\text{init}(q') = \sigma''$. Consequently, the history $g = p'q'$ is in $S_1 \parallel S_2$ which suffices.

Sub-case iii. $h \in \tau(S_2, S_1)$. This sub-case is treated similarly to the previous one. □

A direct characterization of the histories occurring in such a composition can be proposed by means of the notion of $S_1\parallel S_2$-partitionable history. Before providing it, let us examine the main intuition of the operator $\parallel$. The key idea is that the presence of a history $h$ in the set $S_1 \parallel S_2$, is to be interpreted as the parallel computation of two agents and the sets $S_1$ and $S_2$, associated with these agents, as their denotational semantics. We call $S_1\parallel S_2$-partition of $h$, the respective histories $h_1$ in $S_1$ and $h_2$ in $S_2$ associated with the computation of the two agents whose composition provides the considered global computation associated with $h$.

The interleaving approach to the parallel composition of two agents we use suggests that the steps of a computation can be divided into two sets: those borrowed from one computation step of the first agent and those borrowed from the second one. But any step in the history does not correspond to a computational step. Some of them correspond to the firing of a temporal transition. These temporal steps have to be followed simultaneously by the two agents and then appear in the two histories. However, if one of the two agents terminates its computation, the remaining transitions only appear in the history associated with the computation of the second agent. In that case it is not required that the last step appears in the computation of the two agents but actually only in the non terminated one.

Let us provide some illustrations.

**Example 38.** Consider the successful history $h$ described here under, pairs of histories $h_1$, $h_2$ and $g_1$, $g_2$ are acceptable $S_1\parallel S_2$-partitions provided they are respectively in $S_1$ and $S_2$:

$$h = s_1, s_2, s_3, (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^+)$$

$$h_1 = s_1, s_2, s_3, (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^+)$$

$$h_2 = s_1, s_2, s_3, (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^+)$$

$$g_1 = s_1, s_2, s_3, (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^+)$$

$$g_2 = (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^+).$$

Failing histories admit a similar partitioning. The only distinction is that only one of the two combined histories has to be failing. The other one can either be failing or successful.

**Example 39.** Consider the failing history $h$ described here under, pairs of histories $h_1$, $h_2$ and $g_1$, $g_2$ are acceptable $S_1\parallel S_2$-partitions provided they are respectively in $S_1$ and $S_2$:

$$h = s_1, s_2, s_3, (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^-)$$

$$h_1 = s_1, s_2, s_3, (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^-)$$

$$h_2 = s_1, s_2, s_3, (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^-)$$

$$g_1 = s_1, s_2, s_3, (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^-)$$

$$g_2 = (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^-).$$

In the two previous examples, the two combined histories terminate on the same final state. Partition can also be made in a more asymmetric way of one history terminating before the other one. Let us illustrate this kind of partitions.

**Example 40.** Consider the successful history $h$ described here under, pairs of histories $h_1$, $h_2$ and $g_1$, $g_2$ are acceptable $S_1\parallel S_2$-partitions provided they are respectively in $S_1$ and $S_2$:

$$h = s_1, s_2, s_3, (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^+)$$

$$h_1 = s_1, s_2, s_3, (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^+)$$

$$h_2 = s_1, s_2, s_3, (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \delta^+).$$

$$g_1 = s_1, s_3, (\sigma_u, \delta^+).$$

$$g_2 = (\sigma_u, \sigma_{u+1}), s_4, s_5, (\tau_{u+1}, \tau_{u+2}), s_6, s_7, (\mu_{u+2}, \delta^+).$$

Notice that similar “asymmetric” partitions can be provided for failing histories. In that case, at least one of the two combined histories has to terminate with a $\delta^-$ symbol.

Technically speaking, these considerations are captured by the following definition.

**Definition 41.** Let $S_1, S_2$ be two sets of $\mathcal{H}$. 
(1) A finite history \( h = h'.(\sigma_1, \delta^+) \) of \( Sfthist \) is \( S_1 \rightarrow S_2 \)-partitionable iff there are
(a) two (possibly empty) finite sequences \((k_k)_{k=1,\ldots,\maxi} \) and \((j_k)_{k=1,\ldots,\maxj} \) such that:
\[
\{ i_k : 1 \leq k \leq \maxi \} \cup \{ j_k : 1 \leq k \leq \maxj \} = \{ 1, \ldots, \text{length}(h) - 1 \}
\]
\[
\{ i_k : 1 \leq k \leq \maxi \} \cap \{ j_k : 1 \leq k \leq \maxj \} = \{ l : m : \text{step}_l(h) \text{ is temporal} \}
\]
where \( m \) denotes the minimum of \( i_{\maxi} \) and \( j_{\maxj} \).
(b) histories \( h_1 \) of \( S_1 \) and \( h_2 \) of \( S_2 \) such that
\[
h_1 = \text{step}_{i_1}(h) . \cdots . \text{step}_{i_{\maxi}}(h).(\lambda_{\nu}, \delta^+)
\]
\[
h_2 = \text{step}_{j_1}(h) . \cdots . \text{step}_{j_{\maxj}}(h).(\mu_{\nu}, \delta^+)
\]
with either
1. \( \lambda = \mu = \sigma \) and \( t = v = w \).
2. \( \lambda = \sigma, t = v > w \) and \( \mu_{\nu} \) is the initial state of one temporal transition of \( h_1 \).
3. \( \mu = \sigma, t = w > v \) and \( \lambda_{\nu} \) is the initial state of one temporal transition of \( h_2 \).

(2) A finite history \( h = h'.(\sigma_1, \delta^-) \) of \( Sfthist \) is \( S_1 \rightarrow S_2 \)-partitionable iff there are
(a) two (possibly empty) finite sequences \((k_k)_{k=1,\ldots,\maxi} \) and \((j_k)_{k=1,\ldots,\maxj} \) such that:
\[
\{ i_k : 1 \leq k \leq \maxi \} \cup \{ j_k : 1 \leq k \leq \maxj \} = \{ 1, \ldots, \text{length}(h) - 1 \}
\]
\[
\{ i_k : 1 \leq k \leq \maxi \} \cap \{ j_k : 1 \leq k \leq \maxj \} = \{ l : m : \text{step}_l(h) \text{ is temporal} \}
\]
where \( m \) denotes the minimum of \( i_{\maxi} \) and \( j_{\maxj} \).
(b) histories \( h_1 \) of \( S_1 \) and \( h_2 \) of \( S_2 \) such that
\[
h_1 = \text{step}_{i_1}(h) . \cdots . \text{step}_{i_{\maxi}}(h).(\lambda_{\nu}, \delta^-)
\]
\[
h_2 = \text{step}_{j_1}(h) . \cdots . \text{step}_{j_{\maxj}}(h).(\mu_{\nu}, \delta^-)
\]
with \( \delta_1 = \delta^- \) or \( \delta_2 = \delta^- \) and either
1. \( \lambda = \mu = \sigma \) and \( t = v = w \).
2. \( \lambda = \sigma, t = v > w \) and \( \mu_{\nu} \) is the initial state of one temporal transition of \( h_1 \).
3. \( \mu = \sigma, t = w > v \) and \( \lambda_{\nu} \) is the initial state of one temporal transition of \( h_2 \).

**Definition 42.** Let \( S_1, S_2 \) be two sets of \( \mathcal{H} \) and let \( h \) be a history of \( Sfthist \). Histories \( h_1 \) and \( h_2 \) defined as in one of the two items of **Definition 41** are called an \( S_1 \rightarrow S_2 \)-partition of \( h \).

The following properties establish the relationship between the partitionability of histories and the parallel semantics composition of sets of histories.

**Proposition 43.** Let \( S_1 \) and \( S_2 \) be in \( \mathcal{H} \).
\[ S_1 \cong S_2 = \{ h \in Sfthist : h \text{ is } S_1 \rightarrow S_2 \text{-partitionable} \} \]

**Proof.** The two inclusions contained in the equality can be established by induction on the length of histories. \( \square \)

### 4.3.3. Choice

An operator \( \bigoplus \) is introduced in order to combine semantical sets associated with agents composed with the choice operator \( + \). Before providing the general definition of \( \bigoplus \) on the semantical domain \( \mathcal{H} \), let us recall the operational semantics of the composition of two agents with the choice operator. The main ideas are as follows.

(1) An agent always fires a computational transition if possible. If several computational transitions can be fired, the choice is nondeterministic.

(2) A temporal transition can be fired only if no computational one can occur. In this case, the choice between the two agents is postponed until one of the two agents becomes capable of firing a computational transition. If one of the two agents finishes its computation with failure after several temporal steps, the choice composition behaves as the unterminated one.

(3) Failure is declared if and only if no transition can be fired.

We examine now the choice composition of two sets of histories. The intuition behind choice agrees easily with the fact that histories starting with a computational step of both combined sets have to be in the choice composition. It appears also quite evident that directly failing histories are conserved in the composition only if they appear in the two combined sets. Indeed, if a directly failing history appears in only one of the two combined sets, the uniform non-emptiness at level 0 of the sets ensures the existence in the other set of a longer history starting on the same situation. In all the cases, this longer history will be preferred to a failure.

As regards histories starting with a finite sequence of temporal steps followed by a computational step, they will be included in the composition in the following two cases:

(1) if the other set includes a history starting with the same temporal prefix,

(2) if the other set includes a history that fails immediately after a sub-prefix of this temporal prefix.
Similarly, histories that fail immediately after a sequence of temporal steps and infinite sequences of temporal steps will be included in the composition either if they are in the two sets or if the other set involves a history that fails immediately after one of its temporal prefixes.

The following definition captures this intuition.

**Definition 44.** Define \( \tilde{\oplus} : \mathcal{H} \times \mathcal{H} \rightarrow \mathcal{H} \) as the following function: for any element \( S_1, S_2 \) of \( \mathcal{H} \),

\[
S_1 \tilde{\oplus} S_2 = S_1^1 \cup S_2^1 \cup (S_1 \cap S_2) \cup S_1 \tilde{\oplus} S_2 \cup S_2 \tilde{\oplus} S_1 \cup S_1 \tilde{\oplus} S_2 \cup S_2 \tilde{\oplus} S_1
\]

where

\[
S_1 \tilde{\oplus} S_2 = \{ h = t_1, \cdots, t_n, h_1 \in S_1 : t_i (1 \leq i \leq n) \text{ are temporal steps, } \text{step} \_1(h_1) \text{ is a computational step, } S_2[t_1, \cdots, t_n] \neq \emptyset \}
\]

\[
S_1 \tilde{\oplus} S_2 = \{ h = t_1, \cdots, t_n, h_1 \in S_1 : t_i (1 \leq i \leq n) \text{ are temporal steps, } \exists j < n : t_1, \cdots, t_j, (\text{init}(t_{j+1}), \delta) \in S_2 \text{ or } t_1, \cdots, t_n, (\text{init}(h_1), \delta) \in S_2 \}
\]

**Lemma 45.**

1. \( \tilde{\oplus} \) is well defined,
2. For any sets \( S_1 \) and \( S_2 \) in \( \mathcal{H} \) uniformly non-empty, \( S_1 \tilde{\oplus} S_2 \) is uniformly non-empty.

**Proof.**

1. Given \( S_1, S_2 \in \mathcal{H} \), the membership of \( S_1 \tilde{\oplus} S_2 \) in \( \mathcal{H} \) results directly from **Definition 44**.
2. Direct from the definition. \( \Box \)

### 4.4. Denotational semantics

We are now in a position to define the denotational semantics of any agent.

**Definition 46.** Define the denotational semantics as the following function \( \mathcal{D} : \mathcal{L} \rightarrow \mathcal{P}(\text{Sfhist}) \): for any token \( t \), any positive finite duration \( d \) and any agents \( A_1 \) and \( A_2 \)

\[
\mathcal{D}(\text{tell}_t(t)) = \mathcal{D}_t(\text{tell}_t(t))
\]

\[
\mathcal{D}(\text{ask}_t(t)) = \mathcal{D}_a(\text{ask}_t(t))
\]

\[
\mathcal{D}(\text{nask}_t(t)) = \mathcal{D}_n(\text{nask}_t(t))
\]

\[
\mathcal{D}(\text{get}_t(t)) = \mathcal{D}_g(\text{get}_t(t))
\]

\[
\mathcal{D}(\text{delay}(d)) = \mathcal{D}_d(\text{delay}(d))
\]

\[
\mathcal{D}(A_1 ; A_2) = \mathcal{D}(A_1) \tilde{\oplus} \mathcal{D}(A_2)
\]

\[
\mathcal{D}(A_1 || A_2) = \mathcal{D}(A_1) \parallel \mathcal{D}(A_2)
\]

\[
\mathcal{D}(A_1 + A_2) = \mathcal{D}(A_1) \tilde{\oplus} \mathcal{D}(A_2).
\]

**Proposition 47.** For any agent \( A \) in \( \mathcal{L} \),

1. \( \mathcal{D}(A) \) is uniformly non-empty,
2. \( \mathcal{D}(A)^+ \) is empty.

**Proof.** By structural induction. \( \Box \)

### 4.5. Correctness

The denotational semantics can be related to the operational one as follows.

**Proposition 48.** Let \( \alpha : \mathcal{P}(\text{Sfhist}) \rightarrow \mathcal{P}(\text{Shist}) \) be the function defined as follows. For any subset \( S \) of \( \text{Sfhist} \)

\[
\alpha(S) = \{ h : h \in S, h \text{ is continuous and } \text{init}(h) = \emptyset \}.
\]

Then

\[
\mathcal{O}_h = \alpha \circ \mathcal{D}.
\]
Proof. The proof follows from the fact that for any agent $A$,
\[ \mathcal{D}(A) = \{ (\sigma_u, \delta^+) : h \in S_f \text{this} \} \]
which can be established by induction on the structure.

Proposition 49. Extend the denotational semantics on the agent $E$ as follows:
\[ \mathcal{D}(E) = \{(\sigma_u, \delta^+) \in S_f \text{this} \} \]
Let $A$ be an agent and $s, t$ be situations such that $\mathcal{D}(A)((s, t)) \neq \emptyset$. Moreover, let $B_1, \ldots, B_n$ be all the agents such that $\langle A | s \rangle \rightarrow (B_i | t_i)$. Then,
\[ \mathcal{D}(A)((s, t)) \subseteq \mathcal{D}(B_1) \cup \cdots \cup \mathcal{D}(B_n). \]
Proof. By structural reasoning. \(\square\)

4.6. Full abstraction

The next property to ask is whether $\mathcal{D}$ contains the least information necessary to be compositional and correct. That corresponds to a full abstraction result. Let us recall the formal definition of that property.

Definition 50. Let $\square$ be a fresh symbol. Define the set of contexts $\text{Ccontext}$ by the following rule where $A$ represents an agent.
\[ C ::= \square | A | C ; A | A | C | C[A] || A | C | A + C. \]
The application of a context $C$ to an agent $B$ is defined as the new agent obtained by replacing the place holder $\square$ in $C$, if any, by $B$. This is subsequently denoted as $C[B]$.

Definition 51. The semantics $\mathcal{D}$ 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) $\mathcal{D}(A_1) = \mathcal{D}(A_2)$.

The proof of this result is conducted according to the lines of [3]. However, the introduction of time requires special care which leads to novel treatments.

4.6.1. Intuition

The compositional property of $\mathcal{D}$ together with Proposition 48 establish the implication (ii) $\Rightarrow$ (i). It thus remains to prove the converse (i) $\Rightarrow$ (ii). To that end, we shall proceed by contra-position. Given two agents $A_1, A_2$ such that $\mathcal{D}(A_1) \neq \mathcal{D}(A_2)$ 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 $C[A_2]$. With the relation between $\mathcal{O}_h$ and $\mathcal{D}$ stated in Proposition 48, this amounts to establishing the existence of a continuous denotational history, starting in $\emptyset_1$, which is in $\mathcal{D}(C[A_1])$ and not in $\mathcal{D}(C[A_2])$. To that end, following [15], we shall construct from $h$ a new history $k$ and agents $S$ and $T$ such that $k$ is in the denotation of $S$: $(A_1 || T)$ and $S$: $(A_2 || T)$ does not contain $k$.

The fact that $h$ is in $\mathcal{D}(A_1) \setminus \mathcal{D}(A_2)$ ensures that there is a finite prefix $p$ of $h$ such that $\mathcal{D}(A_1)[p] \neq \emptyset$ and $\mathcal{D}(A_2)[p] = \emptyset$. Our reasoning proceeds by induction on the minimal length of $p$.

In the base case of the induction, $p$ takes the form $(\sigma_u, \delta^-)$ or $(\sigma_u, \tau_r)$. In the case where $p$ is a final step, the tester is only one sequentially composed agent $S$ that essentially constructs a continuous sequence yielding $\sigma_u$ from the initial situation $\emptyset_1$ in a way that the sequential composition forces $A_1$ and $A_2$ to do the last step $(\sigma_u, \delta^-)$. By hypothesis, this is possible for $A_1$ and not for $A_2$.

In the case where $p$ is a computational or temporal step, it can be extended in a continuous history which is in $\mathcal{D}(A_1)$ but not in $\mathcal{D}(A_2)$. The tester is then built in a similar way to the previous case in order to force $A_1$ and $A_2$ to do the step $(\sigma_u, \tau_r)$ which is possible for $A_1$ and not for $A_2$.
In the non basic case, \( \rho = \text{step}_1(p), \rho' \) is composed of at least two steps and one has \( \mathcal{D}(A_1)[\text{step}_1(p)] \neq \emptyset \) while \( \mathcal{D}(A_2)[\text{step}_1(p)] \neq \emptyset \). The proof then uses induction. However, the induction should be applied for \( \rho' \) in \( \mathcal{D}(A_1)[\text{step}_1(p)] \) and not in \( \mathcal{D}(A_2)[\text{step}_1(p)] \). As stated by Proposition 49, these sets turned out to be basically but not exactly the denotations \( \mathcal{D}(A'_i) \) and \( \mathcal{D}(A'_{j}), \) of some agents \( A'_i \) and \( A'_{j} \). We shall consequently generalize slightly the induction to sets of denotational histories. This extension being discarded for the moment for the sake of simplicity, we thus apply the induction hypothesis for \( \rho', A'_i, \) and \( A'_{j} \). It points out a tester \( T' \) and a history \( h'' \) which is in \( \mathcal{D}(A'_i) \parallel \mathcal{D}(T') \) and not in \( \mathcal{D}(A'_{j}) \parallel \mathcal{D}(T') \). From there we should construct a tester \( T \) and a history \( k' \) in \( \mathcal{D}(A_1) \parallel \mathcal{D}(T) \) and not in \( \mathcal{D}(A_2) \parallel \mathcal{D}(T) \). Basically, the step \( \text{step}_1(p) = (\sigma, \tau) \) has to be done before \( h'' \) and since \( k' \) needs to be continuous, \( h'' \) has to start in a possibly non-empty store at any possible time. Hence, we have to generalize our reasoning and construct \( T \) in general from \( p \) a history \( h \) which starts with any initial store and at any time.

Given this generalization, the tester \( T \) basically first consists in making the necessary steps to produce \( \sigma_i \) from the given initial store, then in doing an auxiliary transition chosen so as to ensure that \( A_1 \) and \( A_2 \) have to do the step \( (\sigma, \tau) \) and finally of performing \( T' \).

Note that the above parallel composition is sequentially prefixed by an agent \( S \), yielding \( \sigma_i \) from the initial situation \( \emptyset \) as in the previous cases.

The conclusion is obtained from the fact that a history \( k \) is built along the construction to be in \( \mathcal{D}(S) \equiv (A_1 \parallel T) \) and not in \( \mathcal{D}(S) \equiv (A_2 \parallel T) \).

4.6.2. Auxiliary concepts

The above intuition points out one auxiliary task that consists in producing a given target store \( \tau \) from a given initial store \( \sigma \). These steps are subsequently achieved by means of the agent \( \text{Ag}^V_{\sigma_n \rightarrow \tau_n} \) where \( V \) denote a set of tokens that cannot appear in the definition of \( \text{Ag}^V_{\sigma_n \rightarrow \tau_n} \). This requirement is technically useful, when the agent will be put in parallel with another one, to recognize which agent is responsible for some given steps.

**Definition 52.** Let \( V \) be a finite set of tokens, \( \sigma \) and \( \tau \) be two stores. Let

\[
\sigma \setminus \tau = \{a_1^0, \ldots, a_m^n\}
\]

\[
\tau \setminus \sigma = \{t_1^0, \ldots, t_n^n\}
\]

with \( m, n \geq 0 \). Let \( a_1^0, \ldots, a_m^n \) be tokens not in \( V, \sigma, \) and \( \tau \). Abusing language by forgetting in the notation about these \( a \)'s, we denote by \( \text{Ag}^V_{\sigma_n \rightarrow \tau_n} \), the agent

\[
\begin{align*}
& \text{get}_{t_1^0}(a_1^0); \text{tell}_{t_1^0}(a_1^0); \\
& \vdots \\
& \text{get}_{t_n^n}(a_n^n); \text{tell}_{t_n^n}(a_n^n); \\
& \text{get}_{t_n^n}(a_1^1); \ldots; \text{get}_{t_n^n}(a_n^n).
\end{align*}
\]

Moreover, we note by \( \Sigma^V_{\sigma_n \rightarrow \tau_n} \) the associated sequence of states

\[
(\xi_0, \gamma_1, \xi_1), (\gamma_1, \xi_1), \\
(\xi_{m-1}, \gamma_{m}, \xi_{m}), (\gamma_{m}, \xi_{m}), \\
(\xi_{m-1}, \gamma_{m}, \xi_{m}), (\gamma_{m}, \xi_{m}), \\
(\rho_{m+n-1}, \tau_{n}, \xi_{m+n}), (\tau_{n}, \xi_{m+n}), \\
(\rho_{m+n}, \rho_{1}), \ldots; (\rho_{m+n}, \rho_{m+n})
\]

where

\[
\begin{align*}
\xi_0 &= \sigma \\
\rho_0 &= \xi_{m+n} \\
\rho_{m+n} &= \tau \\
\gamma_i &= \xi_{i-1} \setminus \{a^i\} \quad (1 \leq i \leq m) \\
\xi_i &= \gamma_i \cup \{a_i\} \quad (1 \leq i \leq m) \\
\tau_j &= \xi_{m+j-1} \cup \{t_j\} \quad (1 \leq j \leq n) \\
\xi_{m+j} &= \tau_j \cup \{a^{m+j}\} \quad (1 \leq j \leq n) \\
\rho_k &= \rho_{k-1} \setminus \{a_i\} \quad (1 \leq k \leq m + n).
\end{align*}
\]

Obviously, \( \text{Ag}^V_{\sigma_n \rightarrow \tau_n} \) can generate histories of the form \( \Sigma^V_{\sigma_n \rightarrow \tau_n}(\gamma, \delta^+) \) for any store \( \gamma \). If \( V \) is suitably chosen, it also has the property of being responsible for making the steps of \( \Sigma^V_{\sigma_n \rightarrow \tau_n} \) when placed in parallel with another agent.
Proposition 53. Let $\sigma$ and $\tau$ be two stores. Let $A$ be an agent and let $V$ contain the tokens present in the tell, get, ask and nask communication primitives of $A$.

1. Any history $h = \Sigma^V_{\sigma_u \rightarrow \tau_u} h'$ of $\mathcal{D}(Ag^V_{a \rightarrow a}) || A$ has a $\mathcal{D}(Ag^V_{a \rightarrow a})$-partition in $h_1 = \Sigma^V_{\sigma_u \rightarrow \tau_u} (\gamma_u, \delta^+) + h_2$ for some store $\gamma$ and some history $h_2 \in \mathcal{D}(A)$.

2. For any agent $B$, any history $h = \Sigma^V_{\sigma_u \rightarrow \tau_u} h'$ of $\mathcal{D}(Ag^V_{a \rightarrow a} ; B) || A$ has a $\mathcal{D}(Ag^V_{a \rightarrow a} ; B)$-partition in $h_1 = \Sigma^V_{\sigma_u \rightarrow \tau_u} h_0$ and $h_2$ for some histories $h_0 \in \mathcal{D}(A)$ and $h_2 \in \mathcal{D}(B)$.

**Proof.** The agent $Ag^V_{a \rightarrow a}$ is slightly different from that presented in [3]. Indeed its definition pays attention to the durations associated with the tokens added by $Ag^V_{a \rightarrow a}$ on the store. Nevertheless, the proof is a direct adaptation from that given in [3]. □

**Proposition 53** can be extended to more general sets of denotational histories.

**Notation 54.** Let $h$ be a history of $\mathcal{S}$-thist. Then $\text{diff}(h)$ denotes the set of tokens put or got on the store during this history by a computational step:

$$diff(h) = \{ t : \exists i, d > 0 : \text{step}(h) = (\sigma_u, \tau_u), t_d \in (\sigma \setminus \tau) \cup (\tau \setminus \sigma) \}$$

where $\cup$ and $\setminus$ denote, respectively, multi-set union and difference. Abusing notations, we shall lift $\text{diff}$ to sets of histories in the

natural way: for any set $S$ of histories of $\mathcal{S}$-thist,

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

**Proposition 55.** Let $\sigma$ and $\tau$ be two stores. Let $S$ be a set of $\mathcal{H}$ such that $\text{diff}(S)$ is finite, uniformly non-empty at every level $n > 0$, and let $V$ contain $\text{diff}(S)$.

1. Any history $h = \Sigma^V_{\sigma_u \rightarrow \tau_u} h'$ of $\mathcal{D}(Ag^V_{a \rightarrow a}) || S$ has a $\mathcal{D}(Ag^V_{a \rightarrow a})$-partition in $h_1 = \Sigma^V_{\sigma_u \rightarrow \tau_u} (\gamma_u, \delta^+) + h_2$ for some store $\gamma$ and some history $h_2 \in S$.

2. For any agent $B$, any history $h = \Sigma^V_{\sigma_u \rightarrow \tau_u} h'$ of $\mathcal{D}(Ag^V_{a \rightarrow a} ; B) || S$ has a $\mathcal{D}(Ag^V_{a \rightarrow a} ; B)$-partition in $h_1 = \Sigma^V_{\sigma_u \rightarrow \tau_u} h_0$ and $h_2$ for some histories $h_0 \in S$ and $h_2 \in \mathcal{D}(B)$.

**Proof.** Simple verification using Proposition 53. □

The hypotheses on the set $S$ required for application of Proposition 55 are satisfied by all the sets in $\mathcal{H}$ but the condition that $\text{diff}(S)$ is finite. The following lemma established that the denotational semantics $\mathcal{D}(A)$ of any agent $A$ verifies that additional condition.

**Lemma 56.** For any agent $A$ of $\mathcal{L}$, $\text{diff}(\mathcal{D}(A))$ is finite.

**Proof.** Note that a token occurs in $\text{diff}(h)$ of a history $h \in \mathcal{D}(A)$ if it is put or got on the store by the agent $A$. This can only occur if it is an argument of one of the primitives of $A$. As $A$ only uses a finite set of token, the finiteness property is met. □

4.6.3. Key proposition

We now establish the key result to prove the full abstraction property. It is motivated by the following observation.

The proof of full abstraction of $\mathcal{D}$ with respect to $\theta_h$ will be done *ab absurdo*, by assuming two agents $A_1$ and $A_2$ such that $\mathcal{D}(A_1) \neq \mathcal{D}(A_2)$ and by showing that in this case there is at least one context $C$ such that $\theta_h(C[A_1]) \neq \theta_h(C[A_2])$. This result is obtained by building the context $C$ simultaneously with a continuous history, starting in the situation $h_1$, that is in $C[A_1]$ but not in $C[A_2]$. This construction is based on a history $h$ being in $\mathcal{D}(A_1) \setminus \mathcal{D}(A_2)$ and obtained by induction on the length of the prefix of $h$ which is not a valid prefix for any history in $\mathcal{D}(A_2)$.

Unfortunately the projection $\mathcal{S}[p]$ of an uniformly non-empty set $S$ in $\mathcal{H}$ for a given prefix $p$ does not provide an uniformly non-empty set in $\mathcal{H}$. However the uniform non-emptiness at higher level is kept as the t-continuouness property of all the involved histories. Moreover, thanks to the t-continuouness of $S$, all the histories in $\mathcal{S}[p]$ are starting at equal time. These are the reasons leading to the following rather technical lemma.

**Lemma 57.** Let $u$ be a fixed time. Let $S_1$, $S_2$ be two sets of $\mathcal{H}$ t-continuous such that for $i = 1, 2$, $\text{diff}(S_i)$ is finite and $S_i$ is uniformly non-empty on time $u$.

Moreover, assume $p$ in $(\text{St} \times \text{St})^+ \cup ((\text{St} \times \text{St}) \times \{+, \delta^+\})$ such that $S_1[p] \neq \emptyset$ and $S_2[p] = \emptyset$ and init$(p) = \sigma_u$.

Then for any store $\omega$, there is an agent $T$ and a continuous history in $\mathcal{S}$-thist which starts in $\alpha_u$, and such that $h \in (S_1 \parallel \mathcal{D}(T)) \setminus (S_2 \parallel \mathcal{D}(T))$.

**Proof.** The proof is conducted by induction on the length of $p$.

Case I: $\text{length}(p) = 1$. Then $p$ is of the form $(\sigma_u, \delta^+)$, $(\sigma_u, \delta^-)$, or of the form $(\alpha_u, \sigma_u')$.

Subcase I: $p = (\sigma_u, \delta^+)$. Let us first examine the case where $p = (\sigma_u, \delta^+)$. By hypothesis, $(\sigma_u, \delta^+) \notin S_2$. Let $V$ be the set $\text{diff}(S_1 \cup S_2)$. Consider the agent $T = Ag^V_{a \rightarrow a}$. Obviously,

$$h = \Sigma^V_{\sigma_u \rightarrow \tau_u} (\gamma_u, \delta^+)$$

is a continuous history belonging to $S_1 \parallel \mathcal{D}(T)$.
To conclude in this case, let us prove that $h$ does not belong to $S_2$ $\parallel$ $D(T)$. Indeed, if so, by Proposition 55, $h$ should have $h_1 = \Sigma_{\alpha_u \rightarrow \sigma_u}(\gamma_u, \delta^\tau)$ and $h_2 \in S_2$ as a $D(T) - S_2$-partition, for some store $\gamma$, and some history $h_2 \in S_2$. Moreover, since $h$ ends after $\Sigma_{\alpha_u \rightarrow \sigma_u}$ by $(\sigma_u, \delta^\tau)$, one should have, by Definition 42, $\gamma = \sigma$, and the equality $h_2 = (\sigma_u, \delta^\tau)$. Therefore, $(\sigma_u, \delta^\tau)$ should belong to $S_2$, which contradicts the hypothesis.

Subcase ii: $p = (\sigma_u, \delta^-)$. The case where $p = (\sigma_u, \delta^-)$ can be treated similarly by considering the same agent $T$ and the history

$$h = \Sigma_{\alpha_u \rightarrow \sigma_u}(\sigma_u, \delta^-)$$

with the proof ending by noting that $h_2 = (\sigma_u, \delta^-)$ should belong to $S_2$, which contradicts the hypothesis.

Subcase iii: $p = (\sigma_u, \alpha')$. As $S_1$ is uniformly non-empty at every level $n > 0$, similar arguments to those used in the proof of Lemma 17 provide a continuous extension of $p$ in $S_1$. Let us denote it $p.q$. In this subcase again, the agent

$$T = \text{Ag}_{\alpha_u \rightarrow \sigma_u}^V$$

where $V = \text{diff}(S_1 \cup S_2)$ provides the solution. Indeed, we will establish that the history

$$h = \Sigma_{\alpha_u \rightarrow \sigma_u}^V p.q$$

admits a $D(T) - S_1$-partition but no $D(T) - S_2$-partition. The $D(T) - S_1$-partition is given by $\Sigma_{\alpha_u \rightarrow \sigma_u}(\tau_u, \delta^\tau)$ and $p.q$ where $\tau$ is fixed with respect to $q$. If $q$ terminates at time $u$ and has $(\mu_u, \delta)$ as last step, $\tau$ is fixed to $\mu$. Otherwise, it is fixed to the initial store of the first temporal step of $q$.

To conclude in this case, let us prove that $h$ does not have a $D(T) - S_2$-partition. Indeed, if so, by Proposition 55, the history $h$ should have the pair of histories $h_1 = \Sigma_{\alpha_u \rightarrow \sigma_u}(\tau_u, \delta^\tau)$ and $h_2 \in S_2$ as a $D(T) - S_2$-partition, for some store $\tau$ and some history $h_2 \in S_2$. Moreover, since no extension of $\Sigma_{\alpha_u \rightarrow \sigma_u}$ with $p$ is possible in $S_1$, according to Definition 42, one would then have $h_2 = p = h$, and thus $S_2[p] = \emptyset$ which contradicts the hypothesis.

Case II: $\text{length}(p) > 1$. Let us now consider the case where the length of $p$ is greater than 1. In that case, $p$ is of the form $p = (\sigma_u, \tau_u).p'$ for some store $\tau$ and some history $p'$. There are two cases to be considered: either $S_2[(\sigma_u, \tau_u)] = \emptyset$ or $S_2[(\sigma_u, \tau_u)] \neq \emptyset$ but $S_2[(\sigma_u, \tau_u)] = \emptyset$. Moreover in this second case, we will distinguish the two cases in which $u = v$ and $u + 1 = w$.

Subcase i: $S_2[(\sigma_u, \tau_u)] = \emptyset$. The subcase where $S_2[(\sigma_u, \tau_u)] = \emptyset$ is treated as subcase iii of case I.

Subcase ii: $S_2[(\sigma_u, \tau_u)] \neq \emptyset$ but $S_2[(\sigma_u, \tau_u)][p'] = \emptyset$. In that case, $S_1' = S_1[(\sigma_u, \tau_u)]$ and $S_2' = S_2[(\sigma_u, \tau_u)]$ satisfy the hypothesis with all their histories starting at time $u$. Moreover $S_1'[p'] \neq \emptyset$, $S_2'[p'] = S_2[p] = \emptyset$ and $\text{length}(p') < \text{length}(p)$.

We are thus in the position of applying the induction hypothesis. Let $\text{init}(p') = \mu_u$. Applying the induction hypothesis delivers, for an arbitrarily given store $\alpha'$ to be specified in a moment – a tester $T'$ and a continuous history $h_t$, starting in $\alpha'_u$ and which is in $(S_1' \parallel D(T')) \setminus (S_2' \parallel D(T'))$.

The proof then consists in prefixing $T'$ by some actions, yielding $T$, and $h_t$, by a suitable sequence, yielding $h$, such that $h$ starts in $\sigma_u$, is continuous, and is in $S_1 \parallel D(T)$ and not in $S_2 \parallel D(T)$. Applying the previous technique, $T$ should start by $\text{Ag}_{\alpha_u \rightarrow \sigma_u}^V$ to bring the situation $\alpha_u$ to $\sigma_u$, then leave $S_1$ and $S_2$ do the step $(\sigma_u, \tau_u)$, and finally resume by doing $T'$. In order to force the $S_1$'s to do so, we need a trick which basically consists in adding in $h$, after $(\sigma_u, \tau_u)$, a step that can only be made by $T$. Hence, let $t$ be a fresh token not appearing in $\text{diff}(S_1)$, $\text{diff}(S_2)$, and in the tokens used by $\text{Ag}^V_{\alpha_u \rightarrow \sigma_u}$ and let $\alpha' = \tau \cup \{t\}$. Note that such a $t$ exists thanks to the finiteness of $\text{diff}(S_1)$ and $\text{diff}(S_2)$.

Moreover, let us take

$$T = \text{Ag}_{\alpha_u \rightarrow \sigma_u}^V : \text{tell}_1(t) : T'$$

and

$$h = \Sigma_{\alpha_u \rightarrow \sigma_u}(\sigma_u, \tau_u)(\alpha'_u, \alpha'_{\alpha'_u}).h_t.$$ 

Finally, the history $h$ is in $(S_1 \parallel D(T))$. Indeed, as $h_t$ is in $S_1[(\sigma_u, \tau_u)] \parallel D(T')$, there are $h_1 \in S_1[(\sigma_u, \tau_u)]$ and $h_t \in D(T')$ which are a $S_1[(\sigma_u, \tau_u)] - D(T')$-partition of $h_t$. Consequently, $(\sigma_u, \tau_u).h_1 \in S_1$ and $\Sigma_{\alpha_u \rightarrow \sigma_u}(\alpha'_u, \alpha'_{\alpha'_u}).h_t \in D(T)$ are a $S_1 - D(T)$-partition of $h$. Therefore $h$ is in $S_1 \parallel D(T)$.

To conclude, it remains to be established that $h \not\in S_2 \parallel D(T)$.

We shall proceed by contradiction as before. If $h \in S_2 \parallel D(T)$, then, in view of Proposition 55, the history $h$ should have a $D(T) - S_2$-partition provided by the histories $\Sigma_{\alpha_u \rightarrow \sigma_u}^V h_u$, and $h_u$ for some histories $h_u \in D(\text{tell}_1(t) \parallel T')$ and $h_u \in S_2$. Moreover, $T$ cannot be responsible for the step $(\sigma_u, \tau_u)$, restated in formal terms, $h_t$ cannot be of the form $h_t = (\sigma_u, \tau_u).h'_t$. Indeed, if this was the case, then $\tau = \sigma \cup \{t\}$, whereas by definition $t \not\in \tau^*$. Hence, $h_t = (\sigma_u, \tau_u).h'_t$ for some history $h'_t$. Note that, since $h_t \in S_2$, one has $h'_t \in S_2[(\sigma_u, \tau_u)]$. Moving one step further in $h$, again, thanks to the choice of $t$, $S_2$ cannot perform the step $(\tau_u, \alpha'_{\alpha'_u})$. I.e. $h'_t$ cannot rewrite as $h'_t = (\tau_u, \alpha'_{\alpha'_u})h''_t$.

To sum up, $h'_t$ and $h''_t$ should be a $D(T') - S_2[(\sigma_u, \tau_u)]$-partition of $h$, and consequently, $h_t \not\in S_2[(\sigma_u, \tau_u)] \parallel D(T')$, which contradicts the fact that by construction the history $h_t$ is in $(S_1[(\sigma_u, \tau_u)] \parallel D(T')) \setminus (S_2[(\sigma_u, \tau_u)] \parallel D(T'))$. 

Subcase iii: $S_2[(\sigma_u, \tau_v)] \neq \emptyset$ but $S_2[(\sigma_u, \tau_v)][p'] = \emptyset$ with $v = u + 1$. As $(\sigma_u, \tau_v)$ is a valid first step for some histories in $t$-correct sets, one has $\tau_v = \sigma_{u+1}$. The required agent and history are provided by an adaptation of the previous case by taking

$$T = Ag_{\sigma_u \rightarrow \sigma_2}^V \cdot \text{delay}(1) \cdot \text{tell}_1(t) \cdot T'$$

and

$$h = \Sigma_{\sigma_u \rightarrow \sigma_2}^V \cdot (\sigma_u, \tau_v). \cdot (\tau_v, \tau_v). \cdot (\tau_v, \alpha'). \cdot h_1,$$

The temporal step $(\sigma_u, \tau_v)$ occurs in the two histories of the partition and is followed by $(\tau_v, \tau_v)$ in the history of $D(T)$. \[ \square \]

4.6.4. Proof of the full abstraction property

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

**Proposition 58.** The semantics $D$ is fully abstract with respect to the semantics $\Theta_h$.

**Proof.** Following Definition 51, the two following properties should be established equivalent:

(i) for any context $C$, $\Theta_h(C[A_1]) = \Theta_h(C[A_2])$;

(ii) $D(A_1) = D(A_2)$.

The implication (ii) $\Rightarrow$ (i) follows directly from Proposition 48.

The other implication (i) $\Rightarrow$ (ii) is proved by contra-position. Assume $D(A_1) \neq D(A_2)$. Then, since both $D(A_1)$ and $D(A_2)$ are sets, there is a history $h$ which is in one set and not in the other one. Without loss of generality, we may assume that $h \in D(A_1)$ and $h \notin D(A_2)$.

The proof consists in building a context $C$ and a continuous history $k$ starting on $\emptyset_1$ such that $k \in D(C[A_1])$ and $k \notin D(C[A_2])$.

Let $n$ denote the minimal length such that $D(A_2)[h[n]] = \emptyset$. We distinguish two cases according as the value of $n$ is 1 or bigger.

Case I: $n = 1$. As $S_1$ is in $H$ and satisfies $S_1^* = \emptyset$, there are thus two types of first step of $h$ to consider. It can be an immediately failing history $(\sigma_u, \delta^-)$ or a transition $(\sigma_u, \tau_v)$.

Subcase i: $h = (\sigma_u, \delta^-)$. In the case where the history $h = (\sigma_u, \delta^-)$ is included in $D(A_1)$ but not in $D(A_2)$, a context satisfying the requested properties is obtained by combining sequentially three agents in order to reach by a continuous history the situation $\sigma_u$. Consider the agents $T_1 = \text{delay}(u - 1)$, and $T_2 = Ag_{\sigma_u \rightarrow \sigma_2}$ and the two histories

$$(\emptyset_1, \emptyset_2), \ldots, (\emptyset_{u-1}, \emptyset_u), (\emptyset_u, \emptyset_u), (\emptyset_u, \delta^+) \in D(\text{delay}(u - 1)),$$

and

$$\Sigma_{\sigma_u \rightarrow \sigma_2}^u (\sigma_u, \delta^+) \in D(Ag_{\sigma_u \rightarrow \sigma_2}).$$

The history

$$k = (\emptyset_1, \emptyset_2), \ldots, (\emptyset_{u-1}, \emptyset_u), (\emptyset_u, \emptyset_u), \Sigma_{\sigma_u \rightarrow \sigma_2}^u (\sigma_u, \delta^-)$$

is continuous, starts in $\emptyset_1$, belongs to $D(T_1) \sim D(T_2) \sim D(A_1)$ but not to $D(T_1) \sim D(T_2) \sim D(A_2)$. Indeed, if so, by Definition 34, $k$ should be in $D(T_1 \cup T_2)$ or there should be $k_1, k_2 \in D(T_1 \cup T_2)$ and $k_2 \in D(A_2)$ such that $k_1, k_2 = k$. As

$$(\emptyset_1, \emptyset_2), \ldots, (\emptyset_{u-1}, \emptyset_u), (\emptyset_u, \emptyset_u), \Sigma_{\sigma_u \rightarrow \sigma_2}^u (\sigma_u, \delta^+)$$

is the only continuous history starting on $\emptyset_1$ in $D(T_1 \cup T_2)$ and is successful, one has, on the one hand, the fact that the first possibility may not occur and, on the other hand, the second should imply that $(\sigma_u, \delta^-)$ is in $D(A_2)$ which is not the case.

Let us now define the context $C$ as $C = T_1 \cup T_2$. \[ \square \] By Proposition 48, $\emptyset_1, \overline{K}$ is an operational history of $\Theta_h(C[A_1])$.

Consider now $C[A_2]$. The history $\emptyset_1, \overline{K}$ is not in $\Theta_h(T_1 \cup T_2 \cup A_2)$ which concludes the proof in this case.

Subcase ii: $h[1] = (\sigma_u, \tau_v)$. Following Lemma 17, $h[1]$ can be continuously extended in a history $g = h[1].h'$ of $S_1$. This history $g$ is not in $S_2$ since $S_2[g[1]] = S_2[h[1]]$ is empty. The context built in the previous subcase can be considered again associated with the history

$$k = (\emptyset_1, \emptyset_2), \ldots, (\emptyset_{u-1}, \emptyset_u), (\emptyset_u, \emptyset_u), \Sigma_{\sigma_u \rightarrow \sigma_2}^u$$

which is continuous, starts in $\emptyset_1$, belongs to $D(T_1) \sim D(T_2) \sim D(A_1)$ but not to $D(T_1) \sim D(T_2) \sim D(A_2)$.

The conclusion is obtained, as is subcase i, by considering the context $C = T_1 \cup T_2$ and the history $\emptyset_1, \overline{K}$.

Case II: $n > 1$. In the case where $n$ is bigger than one, $\text{step}_1(h) = (\sigma_u, \tau_v)$. One has that $S_1 = D(A_1)$$[(\sigma_u, \tau_v)]$ and $S_2 = D(A_2)[(\sigma_u, \tau_v)]$ satisfy all the requested hypotheses of the key Proposition 57 since, following the $t$-correctness of $D(A_1)$ and $D(A_2)$, all their histories are starting at time $v$. Moreover $p = \text{step}_2[h]. \ldots . \text{step}_n(h)$ satisfies $S_1[p] \neq \emptyset$ and $S_2[p] = \emptyset$. Denote $\text{step}_2(p) = (\mu_v, \mu'\nu)$. 

\[ \square \]
We apply here a reasoning similar to the one of the proof of the key Proposition 57 in Case II, subcase ii and iii. However, instead of a recursive call, we apply here the key Proposition 57 to the sets $S_1$ and $S_2$. These developments provide us agents $T$ and $T'$ and histories $k'$ and $h$, verifying the following equalities:

$$
T = Ag_{\sigma_u \rightarrow \sigma_u}; \ t_{ll}(t); \ T'
$$

$$
k' = \Sigma'_{\sigma_u \rightarrow \sigma_u}.(\sigma_u; \ t_u). (\tau_u; \ \alpha'_u). h_f
$$

or

$$
T = Ag_{\sigma_u \rightarrow \sigma_u}; \ delay(1); \ t_{ll}(t); \ T'
$$

$$
k' = \Sigma'_{\sigma_u \rightarrow \sigma_u}.(\sigma_u; \ t_u). (\tau_u; \ \tau_u). (\tau_u; \ \alpha'_u). h_f
$$

according as the value of $v$ is $u$ or $u + 1$. Moreover, in both cases, $k'$ is in $\mathcal{D}(A_1) \parallel \mathcal{D}(T)$ but not in $\mathcal{D}(A_2) \parallel \mathcal{D}(T)$.

The construction process is finished by a sequential composition of this agent with the two agents $T_1$ and $T_2$ introduced in the first case of this proof. The history

$$
k = (\emptyset_1, \emptyset_2, \ldots, (\emptyset_{u-1}, \emptyset_u), (\emptyset_u, \emptyset_u), \Sigma_0 \rightarrow \sigma_u \cdot k'
$$

which is continuous, starts in $\emptyset_1$, belongs to $\mathcal{D}(T_1) \sim \mathcal{D}(T_2) \sim \mathcal{D}(A_l \parallel T)$ but not to $\mathcal{D}(T_1) \overset{\sim}{\rightarrow} \mathcal{D}(T_2) \sim \mathcal{D}(A_l \parallel T)$.

Let us now define the context $C$ as $C = T_1; \ T_2; (\circ \parallel T)$. On the one hand, by Proposition 48, $\emptyset_1, \ K$ is an operational history of $\mathcal{O}_h(C[A_1])$. On the other hand, $\emptyset_1, \ K$ is not in $\mathcal{O}_h(C[A_1])$ which concludes the proof. $\Box$

5. Event based semantics

As just proved, the denotational semantics allows us to identify two agents who behave similarly in any context. Although powerful, this property can however be too demanding in some cases, for instance, to compare an implementation with a specification (which in general offers more behaviors). We thus turn in this section to another semantics, based on events.

A first set of events to be considered corresponds to the consultations and modifications of the store. Accordingly, the addition of a token $t$ with duration $d$ to the store is denoted by the event $t_d^+$, a check of the presence of the token $t$ is denoted by the event $t^\circ$, and check of its absence, by $t^\circ$. Moreover, the removal of an occurrence of $t$ out of the store corresponds to the event $t^\circ$.

The second kind of event corresponds to an internal step of the agent, without interaction with the environment. We will denote such an event $r$. Finally, the last event corresponds to the tick of the clock. It is denoted by $v$.

**Definition 59.** Define $S_{\text{event}}$ as the set $\{t_d^+, t^\circ, t^\circ : t \in \text{Stoken}, \ d \in \text{Stime}\} \cup \{\tau, v\}$. Moreover, let $T \subseteq \text{Stoken}$ be a set of tokens. Define $\text{events}(T)$ as the set $\{t_d^+, t^\circ, t^\circ : t \in T, \ d \in \text{Stime}\}$.

We associate with an agent the set of the events it can be responsible for.

**Definition 60.** Given an agent $A$, we define

$$
A^+ = \{t : \ t_{ll}(t) \in \mathcal{F}(A), \ d > 0\} \quad A^- = \{t : \ t_{ll}(t) \in \mathcal{F}(A), \ d > 0\}
$$

$$
A^\circ = \{t : \ t_{ll}(t) \in \mathcal{F}(A), \ d > 0\} \quad A^\circ = \{t : \ t_{ll}(t) \in \mathcal{F}(A), \ d > 0\}
$$

These sets are useful to characterize which agent may fail on some stores. An agent fails if it is not able to do any internal step, nor to compute a primitive, nor to do a temporal step. This is the case of an agent $A$ for which $\mathcal{F}(A)$ contains only obsolete primitives (with 0 as duration) or ask, get, nask primitives on a store that does not allow any of them to fire. Note that if an ask and a nask primitive in $\mathcal{F}(A)$ have the same token as argument (i.e. $A^\circ \cap A^\circ$ is not empty) the agent $A$ is computable whatever the store is. Similarly, if $A^- \cap A^\circ$ is not empty, the agent is always able to compute one primitive whatever the store is.

**Proposition 61.** Let $A$ be an agent. There exists a store on which it cannot fire any computational transition iff the following three conditions hold: (i) $A^+ = \emptyset$, (ii) $\text{delay}(0) \notin \mathcal{F}(A)$ and (iii) $(A^\circ \cup A^-) \cap A^\circ = \emptyset$.

**Proof.** An induction on the syntactic structure of $A$ establishes that for any agent $A$, store $\sigma$ and time $u$, there exist an agent $B$ and a store $\rho$ satisfying $(A | \sigma)_u \rightarrow (B | \rho)_u$ if and only if one of the five following cases occurs.

1. there exist $t \in \text{Stoken}, \ d > 0$ such that $t_{ll}(t) \in \mathcal{F}(A)$
2. there exist $t \in \text{Stoken}, \ d > 0$ such that $t_{ll}(t) \in \mathcal{F}(A)$ and $t \in \sigma^*$
3. there exist $t \in \text{Stoken}, \ d > 0$ such that $get(t) \in \mathcal{F}(A)$ and $t \in \sigma^*$
4. there exist $t \in \text{Stoken}, \ d > 0$ such that $nask(t) \in \mathcal{F}(A)$ and $t \notin \sigma^*$
5. $\text{delay}(0) \in \mathcal{F}(A)$
where $\sigma^*$ denotes the multisets of the tokens occurring in $\sigma$ without their subscript duration. Conversely, $A$ is unable to fire a computational step on the store $\sigma$ at time $u$ if and only if all these conditions are falsified. Therefore, for a given agent $A$, there exist a store $\sigma$ and a time $u$ satisfying $\langle A \mid \sigma \rangle_u \not\rightarrow$ if and only if it is possible to provide a store $\sigma$ and a time $u$ satisfying the five following conditions: (1) $A^+ = \emptyset$, (2) $A^+ \cap \sigma^+ = \emptyset$, (3) $A^- \cap \sigma^* = \emptyset$, (4) $A^* \subseteq \sigma^*$, (5) $\text{delay}(0) \not\in F(A)$. Conditions (1) and (5) are directly satisfied by an agent $A$ failing on a store. As $F(A)$ is finite, conditions (2)–(4) express a finite set of conditions which is always satisfiable but in case a token has both to be and not to be in the store, i.e. if $(A^+ \cup A^-) \cap \sigma^* \neq \emptyset$. □

Definition 62. Let $A$ be an agent. We denote by $A \ll$ the existence of a store on which $A$ fails.

Proposition 63. Let $A$ be an agent. It cannot fire any transition on any store iff $A \ll$ and $A \gg$.

Proof. On the one hand, the condition $A \ll$ occurs if and only if $F(A)$ only contains ask, get and nask primitives. On the other hand, the condition $A \gg$ occurs if and only if none of the ask, get, nask or delay primitives in $F(A)$ have a positive duration. The conjunction of the two conditions occurs then if and only if $F(A)$ involves only ask0, get0 and nask0 primitives, and therefore, if and only if it is unable to fire any transition on any store. □

The transition system of Fig. 1 can be rephrased as a transition system where computation steps are reformulated as their corresponding events. For instance, the label $t^+ \nu$ is used to indicate the addition of the token $t$ with duration $d$ by the computation of a $tell_d(t)$ primitive. The resulting tagged transition system is described in Fig. 2.

A first relation between the two transition systems is easy to establish.

Proposition 64. Let $A$ be an agent of $\mathcal{L}$, $A \xrightarrow{\nu} \gamma$ if and only if there are a store $\sigma$ and a time $u$ such that $\langle A \mid \sigma \rangle_u \sim\nu$.

The computation of an agent may be defined as a sequence of events. As for the semantics $\Theta_h$, a computation may be infinite or finite and, in that latter case, terminated by the symbol $\delta^-$ to denote a successful termination or by the symbol $\delta^+$ to denote a deadlock computation.

The ability to actually perform a sequence of events depends on the contents of the store. For instance, a $t^d$ event may only occur on stores containing the token $t$. Conversely a $t^*$ event may only occur on stores that contain no occurrence of $t$. Similarly, event $\nu$ can only occur on stores on which the agent is blocked. However, as described until now, the temporal event $\nu$ does not contain enough information to decide whether it can be fired on a given store. From now on, we associate with a temporal step $\nu$ two sets $F$ and $G$ in order to indicate respectively which tokens have to be present and absent from the store in order to allow a transition to take place. The careful reader will directly notice the extension made with respect to the failure set semantics discussed in the previous section.
Definition 65.

(1) Define the set of the tagged temporal events \( \text{S} \) as follows
\[
\text{S} = \{ t^+_d, t^0, t^\ast, t^- : t \in \text{Stoken} \} \cup \{ \tau \} \cup \{ \nu^F_G : F, G \subseteq \text{Stoken} \}.
\]

(2) The set \( \mathcal{T} \) of traces is defined as follows
\[
\mathcal{T} = \text{S}^{\omega} \cup \text{S}^* \cdot \{ \delta^+ \} \cup \text{S}^* \cdot \{ \delta^- \}.
\]

The event semantics associates traces in \( \mathcal{T} \) to any agent.

Definition 66. Define the event semantics \( \mathcal{E} : \mathcal{L} \rightarrow \mathcal{P}(\mathcal{T}) \) as the following function. For any agent \( A \),
\[
\mathcal{E}(A) = \{ e_0, \cdots, e_n : A_0 \xrightarrow{e_0} \cdots \xrightarrow{e_{n-1}} A_n, A_0 = A, A_n = E, n \geq 0, \text{ with for any } i \leq n - 1 : e_i = v^A_{A_i} \text{ if } e_i = v \text{ and } e_i = e_i' \text{ otherwise} \}
\cup
\{ e_0, \cdots, e_n : A_0 \xrightarrow{e_0} \cdots \xrightarrow{e_{n-1}} A_n, A_0 = A, A_n = E, n \geq 0, \text{ with for any } i \leq n - 1 : e_i = v^A_{A_i} \text{ if } e_i = v \text{ and } e_i = e_i' \text{ otherwise} \}
\cup
\{ e_0, \cdots, e_n, \cdots : A_0 \xrightarrow{e_0} \cdots \xrightarrow{e_{n-1}} A_n, A_0 = A, \forall i \geq 0, A_i \neq E, e_i = v^A_{A_i} \text{ if } e_i = v \text{ and } e_i = e_i' \text{ otherwise} \}.
\]

Note that this semantics is somewhat richer than the denotational semantics presented in Definition 46. Indeed, in the case of the consultation of the store, the corresponding event in \( \mathcal{E} \) preserves the information specifying which token was searched.

Example 67. Let \( A, B \) and \( C \) denote respectively the three following agents:
\[
A = \text{ask}_1(t) + \text{nas}_1(t) \\
B = \text{ask}_1(v) + \text{nas}_1(v) \\
C = \text{delay}(0).
\]

They have the following semantics
\[
\mathcal{E}(A) = \{ t^0, t^+, t^\ast, t^- \} \\
\mathcal{E}(B) = \{ v^0, v^+, v^\ast, v^- \} \\
\mathcal{E}(C) = \{ \tau, \delta^+ \}.
\]

However they have the same denotational set
\[
\mathcal{D}(A) = \mathcal{D}(B) = \mathcal{D}(C) = \{ (\sigma_u, \sigma_v) : (\rho_u, \delta^+) : \sigma, \rho \in \text{Ststore}, u \in \text{Stime} \}.
\]

This semantics shows that the semantics \( \mathcal{E} \) and \( \mathcal{D} \) are not equivalent. However, we will show that \( \mathcal{E} \) is correct with respect to \( \mathcal{D} \). In order to focus our presentation on the relation between the intuitions underlying the two semantics, we consider in the rest of this section the finite sublanguage considered in Section 4. The proofs for the complete language use contractions on complete metric spaces and can be found by the interested reader in [19].

The restriction to agents in this sublanguage allows one to manipulate only event-semantics sets characterized by a maximal length of their sequences. Consequently, developments can be based on an induction on this length. This is established by the following lemma.

Lemma 68.

(1) For any agent \( A \) in (the finite restriction of) \( \mathcal{L} \), there is an integer \( n \) such that \( \text{length}(s) \leq n \), for any sequence \( s \) in \( \mathcal{E}(A) \).

(2) Let \( A \) and \( B \) be two agents in \( \mathcal{L} \) and \( e \) be an event in \( \text{S} \) such that \( A \xrightarrow{e} B \). It holds that
\[
\sup \{ \text{length}(s) : s \in \mathcal{E}(A) \} < \sup \{ \text{length}(s) : s \in \mathcal{E}(B) \}.
\]

Proof. The proof is easily conducted by induction on the structure of \( A \) by observing that for a primitive \( p \) with associated delay \( d \),
\[
\sup \{ \text{length}(s) : s \in \mathcal{E}(p) \} = d + 1. \quad \square
\]
The correctness of the event semantics \( \mathcal{E} \) with respect to the denotational semantics \( \mathcal{D} \) is established by using an intermediate semantics \( \mathcal{S} \) defined by a functional interpretation of the event.

The rest of this section is structured as follows. We firstly introduce the functional interpretation of events. Then the intermediate semantics is defined and established to be equivalent to the denotational semantics. Finally, we define a function \( \beta \) which establishes that the intermediate semantics can be obtained from the event semantics and, in this way, proves the expected correctness. A functional interpretation of the events allows to relate them with respect to the corresponding effects on stores.

**Definition 69.** For any event \( e \) of \( \text{Seven} \) we define a partial function \( f_e : \text{Ststore} \times \text{Stime} \rightarrow \text{Ststore} \times \text{Stime} \) as follows: for any store \( \sigma \), any time \( u \)

\[
\begin{align*}
f_{e^+}(\sigma_u) &= (\sigma \cup \{t_d\})_u \\
f_{e^+}((\sigma \cup \{t_d\})_u) &= (\sigma \cup \{t_d\})_u \\
f_{e^-}(\sigma) &= \sigma \\
f_{e^-}(\sigma) &= \sigma_u \\
f_{e^\top}(\sigma_u) &= \sigma_{u+1} \\
\text{if there is no } d \text{ such that } t_d \in \sigma
\end{align*}
\]

where \( \sigma^* \) denotes the set of the tokens occurring in \( \sigma \) without their subscript duration.

The following lemma already establishes a link between the two transition systems.

**Lemma 70.** For any agents \( A \) and \( B \), any stores \( \sigma \) and \( \rho \) and any time \( u \), it holds that

\[
\begin{align*}
(1) \langle A \mid \sigma \rangle_u &\rightarrow \langle B \mid \rho \rangle_u \text{ iff there is } e \in \text{Seven} \setminus \{v\} \text{ such that } A \xrightarrow{e} B \text{ and } \rho_u = f_e(\sigma_u) \\
(2) \langle A \mid \sigma \rangle_u &\rightarrow \text{iff for all } e \in \text{Seven} \setminus \{v\} \text{ the fact that } A \xrightarrow{e} B \text{ implies that } f_i(\sigma_u) \text{ is undefined} \\
(3) \langle A \mid \sigma \rangle_u &\rightarrow \text{iff for all } e \in \text{Seven} \setminus \{v\} \text{ the fact that } A \xrightarrow{e} B \text{ implies that } f_e(\sigma_u) \text{ is undefined and that } A \xrightarrow{\rho} \\
(4) \langle A \mid \sigma \rangle_u &\nrightarrow \text{ iff there is no } e \in \text{Seven} \text{ such that } A \xrightarrow{e} .
\end{align*}
\]

**Proof.** The proof of the first property is obtained by induction on the syntax of agents, first for guarded ones and then on arbitrary ones. The second and third properties result directly from the first property. Finally, the fourth property occurs if and only if \( \mathcal{F}(A) \) contains only ask_0, get_0 and nask_0 primitives, i.e. iff \( A \neq E, A \downarrow \) and \( A \nrightarrow \). "

The intermediate semantics is defined by induction of the maximal length of sequences in the event semantics of agents as follows. The well-foundedness of this induction results from Lemma 68.

**Definition 71.** For any agent \( A \) of \( \mathcal{L} \),

\[
\begin{align*}
\mathcal{S}(E) &= \{(\sigma_u, \delta^+) \in \text{Sthist}\} \\
\mathcal{S}(A) &= \begin{cases} 
\{(\sigma_u, \delta^-) \in \text{Sthist}\} & \text{if } A \downarrow \text{ and } A \gg, \\
\{h \in (\sigma_u, \rho_u).\mathcal{S}(B) : B \in \mathcal{E}, h \in \text{Sthist}, \exists e \in \text{Seven}, e \neq v, A \xrightarrow{e} B, \rho_u = f_e(\sigma_u)\} \\
\cup \{h \in (\sigma_u, \sigma_{u+1}).\mathcal{S}(B) : B \in \mathcal{E}, h \in \text{Sthist}, A \xrightarrow{\rho} \forall e \in \text{Seven} \setminus \{v\} : A \xrightarrow{e} \Rightarrow f_e(\sigma_u) = \bot\} & \text{otherwise}.
\end{cases}
\end{align*}
\]

**Proposition 72.** For any agent \( A \) of \( \mathcal{L} \), it holds that \( \mathcal{S}(A) = \mathcal{D}(A) \).

**Proof.** Let us first observe that for any agent \( A \) of \( \mathcal{L} \),

\[
\begin{align*}
\mathcal{D}(A) &= \{h \in (\sigma_u, \tau_v).\mathcal{D}(B) : h \in \text{Sthist}, \langle A \mid \sigma \rangle_u \mapsto \langle B \mid \tau \rangle_v\} \\
&\cup \{(\sigma_u, \delta^-) \in \text{Sthist} : \langle A \mid \sigma \rangle_u \nmapsto\}
\end{align*}
\]

which can be established by induction on the structure.

The equivalence of the two semantics then follows directly from Lemma 70. "

In order to relate the event semantics and the denotational semantics, we now introduce a function \( \beta \) which provides a set in the denotational domain \( \mathcal{H} \) associated with a set in the event semantics domain of the agents of \( \mathcal{L} \), namely the subsets of \( \mathcal{T} \) whose sequences admit a maximal length.
Lemma The event semantics

Fig. 3. Tagged transition system for $\Delta_T A$.

\[ \begin{align*}
(\Delta_1) & \quad A \xrightarrow{\alpha} A', a \in \text{events}(T) \cup \{\nu\} \\
& \quad \Delta_T A \xrightarrow{\alpha} \Delta_T A' \\
(\Delta_2) & \quad A \xrightarrow{\alpha} A', a \notin \text{events}(T) \\
& \quad \Delta_T A \xrightarrow{\tau} \Delta_T A'
\end{align*} \]

Definition 73.

(1) Let $\mathcal{P}^b(\mathcal{T})$ denote the following set

\[ \mathcal{P}^b(\mathcal{T}) = \{ S \in \mathcal{P}(\mathcal{T}) : \exists \ell : \forall s \in \text{Slength}(s) \leq \ell \} \]

(2) The function $\beta : \mathcal{P}^b(\mathcal{T}) \rightarrow \mathcal{H}$, is defined recursively as follows: for any $S$ in $\mathcal{P}^b(\mathcal{T})$:

\[ \beta(S) = \begin{cases} 
\{ (\sigma_u, \delta^+) \in \text{S}fthist \} & \text{if } S = \{ \delta^+ \} \\
\{ (\sigma_u, \delta^-) \in \text{S}fthist \} & \text{if } S = \{ \delta^- \} \\
\{ h \in (\sigma_u, f_e(\sigma_u)), \beta(S[e]) : h \in \text{S}fthist, S[e] \neq \emptyset, f_e(\sigma_u) \neq \bot \} & \text{otherwise.} 
\end{cases} \]

Proposition 74. For any agent $A$ in $\mathcal{L}$, $\beta \circ \mathcal{E}(A) = \delta(A)$.

Proof. Let us first observe that, for finite agents, Definition 66 is equivalent to the following recursive one.

\[ \mathcal{E}(E) = \{ \delta^+ \} \]

\[ \mathcal{E}(A) = \begin{cases} 
\{ \delta^- \} & \text{if } A \downarrow \text{ and } A \gg \\
\{ s \in e.\mathcal{E}(B) : A \xrightarrow{e} B, e \in \text{Sevent} \setminus \{ \nu \} \} \\
\cup \{ s \in v_{A^{\infty,\bot}}^A.\mathcal{E}(B) : A \xrightarrow{\nu} B \} & \text{otherwise.} 
\end{cases} \]

Consequently,

\[ \beta \circ \mathcal{E}(E) = \{ (\sigma_u, \delta^+) \in \text{S}fthist \} \]

\[ \beta \circ \mathcal{E}(A) = \begin{cases} 
\{ (\sigma_u, \delta^-) \in \text{S}fthist \} & \text{if } A \downarrow \text{ and } A \gg \\
\{ s \in (\sigma_u, f_e(\sigma_u)), \beta \circ \mathcal{E}(B) : A \xrightarrow{e} B, e \in \text{Sevent} \setminus \{ \nu \}, f_e(\sigma_u) \neq \bot \} \\
\cup \{ s \in (\sigma_u, \sigma_{u+1}^-).\beta \circ \mathcal{E}(B) : A \xrightarrow{\nu} B, f_{A^{\infty,\bot}}^A(\sigma_u) = \sigma_{u+1}^- \} & \text{otherwise.} 
\end{cases} \]

which, according to Lemma 70 is equivalent to $\delta$. \qed

Proposition 75. The event semantics $\mathcal{E}$ is correct with respect to the denotational semantics $\mathcal{D}$.

Proof. Thanks to Proposition 72, it suffices to prove that two agents admitting the same event semantics $\mathcal{E}(A) = \mathcal{E}(B)$ also have the same intermediate semantics $\delta(A) = \delta(B)$. This is indeed the case since, by Proposition 74, $\delta(A) = (\beta \circ \mathcal{E})(A)$. \qed

6. Refinement

6.1. Auxiliary notions

We are now in a position to define our notion of refinement. To make our theory more general, we introduce a hiding operator $\Delta_T$ whose purpose is to focus the observation on the tokens in $T$ and to hide the other tokens considered as local details, simply observed as $\tau$ steps. The rules to be added to the tagged transition system of Fig. 2 are given in Fig. 3. Some auxiliary notations will also be needed.

Definition 76. For any agent $A$ and $A'$ and any event $\alpha$ we denote

(1) $A \Rightarrow A'$ whenever there are agents $A_i$ for $i = 1, \ldots, n$ such that $A \xrightarrow{\tau} A_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} A_n \xrightarrow{\tau} A'$

(2) $A \Rightarrow^\alpha A'$ whenever there are agents $A_i$ for $i = 1, \ldots, n$ such that $A \xrightarrow{\alpha} A_1 \xrightarrow{\alpha} \cdots \xrightarrow{\alpha} A_n \xrightarrow{\alpha} A'$
3. A ⇒^ν A' whenever there are agents A_i for i = 1, ..., n such that A 1→ A_1 1→ ... 1→ A_n 1→ A'
4. A ⇒^ω whenever there is an infinite sequence of agents A_i for i ∈ N such that A 1→ A_1 1→ ... 1→ A_i 1→ ...

Similarly, for any agents A and A', any set of token T and any event a we denote

1. Δ_TA ⇒ Δ_TA'
2. Δ_TA ⇒^0 Δ_TA'
3. Δ_TA ⇒^ν Δ_TA'
4. Δ_TA ⇒^ω whenever there is an infinite sequence of agents A_i for i ∈ N such that Δ_TA 1→ Δ_TA_1 1→ ... 1→ Δ_TA_i 1→ ...

6.2. Refinement

An refinement relation is defined as follows.

**Definition 77.** A relation R on \(L^S \times \mathcal{P}(\text{Token}) \times L^S\) is a refinement relation if any triple (I, T, S) in R satisfies the following properties

1. \(\text{tokens}(S) \subseteq T\)
2. for any event a and agent I' such that ΔTI ⇒^0 ΔTI' there is some S' in \(L^S\) such that S ⇒^S S' with \((I', T, S') \in R\).
3. for any agent I' such that ΔTI ⇒^ν ΔTI' there is some S' such that S ⇒^S S' with \((I', T, S') \in R\).
4. ΔTI ⇒^ω implies S ⇒^ω.
5. for any agent I' such that ΔTI ⇒ ΔTI' \not ⇒ there is some agent S' such that S ⇒^S S' \not⇒.
6. ΔTI ⇒ ΔTE implies S ⇒ E

where \(\text{tokens}(S)\) denotes the set of tokens appearing in S and the transition rules for ΔTA are provided by Fig. 3.

In this definition, it is worth observing that an interaction of the implementation I with the store has to be a possible interaction of the specification S with the store. Similarly, temporal events of the implementation have to be temporal events made by the specification. But these are not the only required properties. The ability of actions by the specification has to be preserved. This is expressed in conditions 6, 5 and 4 associated with condition 2, by the fact that termination, failure or zeno behavior of the implementation are acceptable only if they respectively correspond to termination, failure and zeno behavior of the specification.

**Example 78.** To illustrate the above definition, let us reconsider a weather service which has to publish a piece of information captured by the token w and to update this token every two units of time. A specification of this service can be provided by the agent W defined as follows

\[
W = \text{tell}_2(w) ; P \\
P = \text{delay}(2) ; \text{tell}_2(w) ; P.
\]

Observing the computation from the point of view of the tagged transition system of Fig. 2 provides the following sequence:

\[
W \overset{w^+}{\rightarrow} W_1 \overset{\nu}{\rightarrow} W_2 \overset{\nu}{\rightarrow} W_3 \overset{\nu}{\rightarrow} W_4 \overset{w^+}{\rightarrow} W_1 \rightarrow \ldots
\]

for some agents W_1 to W_4. This sequence can be read as follows. Firstly, the token w is put on the store with duration 2. Then, two temporal steps are fired. At each temporal step, the argument of the delay decreases by one unit. After two of them it turns to zero and the computation of delay(0) provides an internal step. The token w is updated on the store and the computation goes on similarly.

Using the notation introduced in **Definition 76**, the behavior of W can be expressed by the following sequence

\[
W \Rightarrow^\nu W_1 \Rightarrow^\nu W_2 \Rightarrow^\nu W_3 \Rightarrow^\nu W_4 \Rightarrow^\nu W_1 \ldots.
\]

Assume now that an implementation can be obtained by using an existing service, say X, manipulating another piece of information coded by token r and updated every unit of time. Consider the implementation C built out of two parallel procedures X and Y defined as follows

\[
C = X \parallel Y \\
X = \text{tell}_1(r) ; \text{delay}(1) ; X \\
Y = \text{ask}_1(r) ; \text{compute} ; \text{tell}_2(w) ; \text{delay}(2) ; Y.
\]

The intuition underlying this implementation is as follows. The procedure X denotes the service that updates the information r every unit of time. The procedure Y after consulting the information r, makes some internal computation and
then publishes the information \( w \). Every two units of time, the procedure is restarted. Let us observe the possible behavior according to the tagged transition system

\[
\Delta_{[w]}C \quad \left\{ \begin{array}{c}
\tau \xrightarrow{\text{tell}_1} \\
\tau \xrightarrow{\text{ask}_1} \\
\tau \xrightarrow{\text{compute}} \\
\tau \xrightarrow{\text{tell}_2}
\end{array} \right\} \Delta_{[w]}C_4
\]

\[
\Rightarrow\Delta_{[w]}C_1 \quad \rightarrow \quad \Delta_{[w]}C_6 \quad \rightarrow \quad \Delta_{[w]}C_7
\]

\[
\Delta_{[w]}C_8 \quad \left\{ \begin{array}{c}
\tau \xrightarrow{\text{delay}} \\
\tau \xrightarrow{\text{ask}_1} \\
\tau \xrightarrow{\text{compute}} \\
\tau \xrightarrow{\text{tell}_2}
\end{array} \right\} \Delta_{[w]}C_4 \rightarrow \ldots
\]

for some agents \( C_4 \) to \( C_8 \), where the indication under the arrow are provided to follow to which primitive the computation corresponds and the \{ \} and \} indicate sequences which can be combined according to any interleaving.

Stated in terms of the \( \Rightarrow \) relation, they are several possible sequences according to the choices followed in the order of interleaving. However, they are all of the following type:

\[
\Delta_{[w]}C \Rightarrow w_2 \Delta_{[w]}D \Rightarrow w_1 \Delta_{[w]}E \Rightarrow w_1 \Delta_{[w]}F \Rightarrow w_2 \Delta_{[w]}G \ldots
\]

for some agent \( D, E, F, G, \ldots \). The reader can easily convince himself that \( C \trianglelefteq [w] W \).

As a counter-example, we can observe that the agent \( A = Y \), where \( Y \) is defined as above, does not refine \( W \). Indeed, it admits sequences of the following type:

\[
\Delta_{[w]}A \Rightarrow w_2 \Delta_{[w]}A_1 \Rightarrow w_1 \Delta_{[w]}A_2 \Rightarrow w_1 \Delta_{[w]}A_3 \Rightarrow w_2 \Delta_{[w]}A_4 \ldots
\]

for some agents \( A_1, A_2, A_3, A_4, \ldots \), but also, for example,

\[
\Delta_{[w]}A \Rightarrow w_1 \Delta_{[w]}B_1
\]

and

\[
\Delta_{[w]}A \Rightarrow w_2 \Delta_{[w]}A_1 \Rightarrow w_1 \Delta_{[w]}A_2 \Rightarrow w_1 \Delta_{[w]}A_3 \Rightarrow w_1 \Delta_{[w]}A_4 \ldots
\]

for some agents \( A_1 \) and \( B_1 \). The reader can easily convince himself that those last sequences are not allowed by the specification \( W \). Consequently, agent \( A \) does not satisfy point 5 of Definition 77.

As in traditional concurrency theory, many refinement relations may be defined so that, as usual, we shall focus now on the maximal one defined as the union of all the refinement relations.

**Definition 79.** The refinement relation \( \trianglelefteq \) is the union of all the refinement relations on \( \mathcal{L}^e \times \mathcal{P} (\text{stoken}) \times \mathcal{L}^e \). Moreover, for the ease of reading, we write \( A \trianglelefteq_T S \) whenever the triple \((A, T, S)\) is in \( \trianglelefteq \).

### 6.3. Properties

**Proposition 80.** For any agents \( A, B, C \) of \( \mathcal{L}^e \) and any \( T \) set of tokens,

1. if tokens(\( A \)) \( \subseteq T \), then \( A \trianglelefteq_T A \)
2. if tokens(\( A \)) \( \subseteq T \), then \( A \trianglelefteq_T A + A \)
3. if tokens(\( A \)) \( \subseteq T \), then \( A + A \trianglelefteq_T A \)
4. if tokens(\( B \)) \( \cup \) tokens(\( C \)) \( \subseteq T \), if \( A \trianglelefteq_T B \) and if \( B \trianglelefteq_T C \) then \( A \trianglelefteq_T C \)
5. if \( A \Rightarrow B \) and if \( A \trianglelefteq_T S \) then \( B \trianglelefteq_T S \).

**Proof.** The first property is obtained by observing that the relation \( R \) on \( \mathcal{L}^e \times \mathcal{P} (\text{stoken}) \times \mathcal{L}^e \) defined as follows \( R = \{(A, T, A) : \text{tokens}(A) \subseteq T\} \) is a refinement relation. It is then included in \( \trianglelefteq \).

Properties 2 to 5 are direct consequences of property 1, by observing that inclusion of the set of tokens and the five required properties are satisfied. \( \square \)

**Definition 81.** An agent \( A \) of \( \mathcal{L} \) is said to be deadlock free iff its operational semantics \( \varnothing_h \) does not contain any failing computations.

An important property is that an agent refining a deadlock-free specification is also deadlock-free.

**Theorem 82 (Deadlock-Freeness Preservation).** For any agents \( A \) and \( S \) of \( \mathcal{L} \) such that \( A \trianglelefteq_T S \), if \( S \) is deadlock-free, then \( A \) is also deadlock-free.
Proof. The proof is established by contradiction. Let $s$ be a deadlocking history of $\theta_h(A)$. This history can be written $s = \sigma_{u_1} \ldots \sigma_{u_n} \delta^-$ for some stores $\sigma(i = 1, \ldots, n)$ and times $u_i(i = 1, \ldots, n)$ with $\sigma^1_1 = \emptyset_1$. One then has, for a sequence of agents $A_i(i = 1, \ldots, n)$ of $\mathcal{L}_r$ with $A_1 = A$,

$$(A_1 \mid \sigma_1)_{u_1} \rightarrow \cdots \rightarrow (A_n \mid \sigma_n)_{u_n} \not\rightarrow.$$ 

Proposition 64 and Lemma 70 provide then a sequence of events $e_i(i = 1, \ldots, n - 1)$ of $\text{Sevent}$ such that

$$A_i \not\rightarrow A_{i+1} \quad \text{and} \quad \sigma_{i+1}^j = f_j(\sigma_i^j) \quad \text{for} \ i = 1, \ldots, n - 1.$$ 

Moreover, thanks to Lemma 70, as $\langle A_n \mid \sigma_n \rangle u_n \not\rightarrow$, the agent $A_n$ is not able to fire any tagged transition. According to the transition system of Fig. 3, one has then for any $i = 1, \ldots, n - 1$, $\Delta_T A_i \not\rightarrow \Delta_T A_{i+1}$ or $\Delta_T A_i \not\rightarrow \Delta_T A_{i+1}$.

Let us now define the subsequence of the events preserved by the filter, i.e. those events in $T \cup \{v\}$. Formally, it is provided by the subsequence $e_i(k = 1, \ldots, m)$ such that $j_0 = 1$ and $\Delta_T A_k \not\rightarrow A^{(k)}_k \Delta_1 A_{k+1}$.

The definition of the refinement provides a sequence of agents $S_k(k = 1, \ldots, m)$ with $S_1 = S$, such that $S_k \not\rightarrow S_{k+1}$. Moreover, as $\Delta_T A_n \not\rightarrow \Delta_T A_n$, there is some $S_{m+1}$ such that $S_m \not\rightarrow S_{m+1} \not\rightarrow 1$. As $f_r(\rho) \not\rightarrow \rho$ for any store $\rho$, one has, thanks to Proposition 64 and Lemma 70, for some agents $S^1, \ldots, S^m(k)$,

$$(S_k \mid \rho_k)_{u_k} \rightarrow (S_k^1 \mid \rho_k)_{u_k} \rightarrow \cdots \rightarrow (S_k^{m} \mid \rho_k)_{u_k} \not\rightarrow$$

which concludes the proof. \(\square\)

The following lemma will help to establish the substitutability property.

Lemma 83. For any agents $A$, $B$ and $S$ of $\mathcal{L}$ and for any set of tokens $T$ such that $A \trianglelefteq T$ and tokens($A$) \cap tokens($B$) \subseteq T, it holds that

1. $A \parallel B \equiv_{T \cup \text{tokens}(B)} S \equiv B$ and $A \equiv_{T \cup \text{tokens}(B)} B$ and $S$

2. $A \parallel B \equiv_{T \cup \text{tokens}(B)} S \equiv B$ and $A \equiv_{T \cup \text{tokens}(B)} B$ and $S$

3. $A \parallel B \equiv_{T \cup \text{tokens}(B)} S \equiv B$ and $A \equiv_{T \cup \text{tokens}(B)} B$ and $S$

Proof. The proofs being similar we only establish here that for $A ; B$. Consider the relation $R_{\text{aux}}$ defined as

$$\{(A ; B), T', (S ; B): T' = T \cup \text{tokens}(B), \text{tokens}(A) \cap \text{tokens}(B) \subseteq T, A \equiv S\}$$

and let us establish that $R = \equiv \cup R_{\text{aux}}$ is a refinement relation, in which case, as it includes $\equiv$, the relation $R$ is the $\equiv$ relation itself.

Let $(A ; B, T', S ; B)$ be an element of $R_{\text{aux}}$ and $T$ be a set of token such that $T \cup \text{tokens}(B) = T'$, tokens$(A) \cap \text{tokens}(B) \subseteq T$ and $A \equiv S$. Let us examine each of the six properties required in Definition 77.

1. The inclusion of the sets of tokens is direct from the definition of $R_{\text{aux}}$.

2. Assume $\Delta_T(A ; B) \not\rightarrow \Delta_T(C)$. According to the transition system of Fig. 3, three cases are possible.

   Case 1. $C = A'$; $B$ with $\Delta_T(A) \not\rightarrow \Delta_T(A')$ and $A' \not\rightarrow E$. In this case, as $\text{tokens}(A) \cap T' \subseteq T$, one also has $\Delta_T(A) \not\rightarrow \Delta_T(A')$. As $A \not\rightarrow T$, there is then some $S'$ such that $S \not\rightarrow S'$ and $A' \not\rightarrow S'$. Therefore, according to the definition of $R$ the triple $(A' ; B', T', S')$ is also in $R$. The conclusion then follows from the fact that $S \not\rightarrow S'$ and $B \not\rightarrow B$.

   Case 2. $C = B$ with $\Delta_T(A) \not\rightarrow \Delta_T(E)$. In this case, one similarly has $\Delta_T(A) \not\rightarrow \Delta_T(E)$. As $A \not\rightarrow T$, one has then $S \not\rightarrow S'$ and $S' \not\rightarrow E$. As $B \not\rightarrow B$, it is then direct that $B \not\rightarrow S', B$ and $(B, T', S')$ is in $R$.

   Case 3. $C = B'$ with $\Delta_T(A) \not\rightarrow \Delta_T(E)$ and $\Delta_T(B) \not\rightarrow \Delta_T(B')$. In this case, one also has $\Delta_T(A) \not\rightarrow \Delta_T(E)$. As $A \not\rightarrow T$, one has $S \not\rightarrow E$. Moreover, as tokens $(A) \subseteq T'$ one has $B \not\rightarrow B'$. Therefore one has $S \not\rightarrow B'$ with $(B', T', B')$ in $R$ which suffices.

3. The case $\Delta_T(A ; B) \not\rightarrow \Delta_T(C)$ is treated exactly as $\Delta_T(A ; B) \not\rightarrow \Delta_T(C)$.

4. Assume $\Delta_T(A ; B) \not\rightarrow$. Two cases are to be distinguished.

   Case 1. $\Delta_T(A) \not\rightarrow$. In this case, one also has $\Delta_T(A) \not\rightarrow$ and then $S \not\rightarrow$. Therefore one has $S \not\rightarrow$ which suffices.

   Case 2. $\Delta_T(A) \not\rightarrow \Delta_T(E)$ and $\Delta_T(B) \not\rightarrow$. In this case, one also has $\Delta_T(A) \not\rightarrow \Delta_T(E)$ and then $S \not\rightarrow E$. Moreover, as tokens$(B) \subseteq T'$, one has $B \not\rightarrow$ which suffices.

5. Assume $\Delta_T(A ; B) \not\rightarrow \Delta_T(C) \not\rightarrow$. Two cases are to be distinguished.

   Case 1. $C = A'$; $B$ and $\Delta_T(A) \not\rightarrow \Delta_T(A') \not\rightarrow$. In this case, one also has $\Delta_T(A) \not\rightarrow \Delta_T(A') \not\rightarrow$ and then $S \not\rightarrow$. Therefore one has $S \not\rightarrow S'$ and $B \not\rightarrow$ which suffices.

   Case 2. $\Delta_T(A) \not\rightarrow \Delta_T(E)$ and $\Delta_T(B) \not\rightarrow \Delta_T(C) \not\rightarrow$. In this case, one also has $\Delta_T(A) \not\rightarrow \Delta_T(E)$ and then $S \not\rightarrow E$. Moreover, as tokens$(B) \subseteq T'$, one has $B \not\rightarrow C \not\rightarrow$ which suffices.

6. Finally, assume $\Delta_T(A ; B) \not\rightarrow \Delta_T(E)$. This situation occurs only if $\Delta_T(A) \not\rightarrow \Delta_T(E)$ and $\Delta_T(B) \not\rightarrow \Delta_T(E)$. In this case, on the one hand, as tokens$(A) \cap T \subseteq T$, one also has $\Delta_T(A) \not\rightarrow \Delta_T(E)$ and $S \not\rightarrow E$. On the other hand, as tokens$(B) \subseteq T'$, one has $B \not\rightarrow E$. Therefore $S \not\rightarrow E$ which suffices. \(\square\)
Proposition 84. Let $S$ be a specification and $A$ an agent refining $S$ with respect to $T$. For any context $C[.]$ such that $\text{tokens}(C) \cap \text{tokens}(A) \subseteq T$, then $C[A]$ refines $C[S]$ with respect to $T \cup \text{tokens}(C)$.

Proof. The proposition is established by using Lemma 83 and a simple induction on the structure of the context. □

Theorem 85. Let $S$ be a specification and $A$ an agent refining $S$ with respect to $T$. For any context $C[.]$ such that $\text{tokens}(C) \cap \text{tokens}(A) \subseteq T$, if $C[S]$ is deadlock free then $C[A]$ is also deadlock free.

Proof. For any context $C[.]$ such that $\text{tokens}(C) \cap \text{tokens}(A) \subseteq T$, Proposition 84 ensures that $C[A] \subseteq T \cup \text{tokens}(C) \subseteq C[S]$. Proposition 82 then ensures that if $C[S]$ is deadlock free, then $C[A]$ is also deadlock free. □

7. Semantics and refinement relations

Before concluding this paper, a last natural question to explore concerns the relation between the semantics and the refinement notion that we have studied. The relation between the denotational semantics $\mathcal{D}$ and the event semantics $\mathcal{E}$ is fixed by Proposition 75 which shows that $\mathcal{D} = \beta \circ \mathcal{E}$ and Example 67 which provides agents $A$ and $B$ such that $\mathcal{D}(A) = \mathcal{D}(B)$ and $\mathcal{E}(A) \neq \mathcal{E}(B)$.

In the following subsections, we focus firstly on the relation between the event semantics and the refinement and secondly on the relation between the denotational semantics and the refinement.

As previously, for succinctness of the developments, we limit our presentation to finite behaviors.

7.1. Event semantics and refinement

As the definition of refinement is based on a filtered set of events, the existence of a refinement relation between two agents $A$ and $B$ cannot easily be achieved as an information on their event semantics $\mathcal{E}(A)$ and $\mathcal{E}(B)$ but most directly provides information on filtered versions of those sets. Let us first define the filtering on the event semantics. Then we will describe the relation following from the refinement.

The filtering operator is defined on traces and then extended on sets of traces.

Definition 86.

(1) Let $T$ be a set of tokens. Define $\nabla^E_T : \mathcal{T} \to \mathcal{T}$ recursively as follows: for any event $e$ and any trace $t$ in $\mathcal{T}$:

$$
\nabla^E_T(\delta^+) = \delta^+ \\
\nabla^E_T(\delta^-) = \delta^- \\
\nabla^E_T(e,t) = \begin{cases} e, & \text{if } e \in \text{events}(T) \cup \{v\} \\
\nabla^E_T(t) & \text{otherwise.}
\end{cases}
$$

(2) Let $T$ be a set of tokens. Define $\nabla^E_T : \mathcal{P}(\mathcal{T}) \to \mathcal{P}(\mathcal{T})$ as the direct extension: for any set $S$ in $\mathcal{P}(\mathcal{T})$,

$$
\nabla^E_T(S) = \{\nabla^E_T(t) : t \in S\}.
$$

For the ease of reading, we will denote by $\mathcal{E}_T$ the application of the operator $\nabla^E_T$ to the event semantics. This can be understood as a filtered event semantics.

Notation 87. Let us denote by $\mathcal{E}_T$ the composition $\nabla^E_T \circ \mathcal{E}$.

This filtered event semantics can be related to the refinement relation through the following proposition.

Proposition 88. For any agent $A$ and $B$, and any set of tokens $T$ such that $A \sqsubseteq_T B$, it holds that $\mathcal{E}_T(A) \subseteq \mathcal{E}_T(B)$.

Proof. This result is established by induction on the maximal length of traces in $\mathcal{E}_T(A)$.

In the base case, traces in $A$ have length 1 and thus can only be $\delta^+$ or $\delta^-$. We establish here that if $\delta^+$ is in $\mathcal{E}_T(A)$, then it is also in $\mathcal{E}_T(B)$. The case of $\delta^-$ is similar. According to the definition of $\mathcal{E}_T$, $\delta^+$ is in $\mathcal{E}_T(A)$ iff there are some events $e_1, \ldots, e_n$ which are not in $\text{events}(T)$, are not temporal events and are such that $e_1, \ldots, e_n, \delta^+$ is in $\mathcal{E}(A)$. In this case, it holds that $\Delta_T A \Rightarrow \Delta_T E$. The existing refinement relation ensures then that $B \Rightarrow E$. This suffices to conclude that $\delta^+$ is in $\mathcal{E}(B)$ and in $\mathcal{E}_T(B)$.

In the recursive case, we assume that the property is verified for any agent admitting traces of maximal length $n$ and we establish that it is also the case for any agent $A$ admitting traces of length $n+1$. Let $t$ be a trace in $\mathcal{E}_T(A)$. If the length of $t$ is 1, its presence in $\mathcal{E}_T(B)$ is established similarly to the basic case. Otherwise, $t$ can be written $e.t'$ for some event $e$ in $\text{events}(T) \cup \{v\}$ and some trace $t'$. The presence of $e.t'$ in $\mathcal{E}_T(A)$ is the consequence of the presence of some trace $t'' = e_1, \ldots, e_n, e.t''$ in $\mathcal{E}(A)$ such that $e_1, \ldots, e_n$ are not in $\text{events}(T)$, are not temporal events and are such that $\nabla^E_T(t'') = t'$. In this case, there exists an agent $A'$ such that $\Delta_T A \Rightarrow^e \Delta_T A'$ and $t'' \in \mathcal{E}(A')$. According to the refinement relation, there exists an agent $B'$ such that $B \Rightarrow^e B'$ and $A' \sqsubseteq_T B'$. The recursion hypothesis then ensures that $t'$ is in $\mathcal{E}(B')$ and consequently, it holds that $t = e.t' \in \mathcal{E}_T(B)$. □
We have seen that the refinement relation implies the inclusion of the filtered event semantics. No converse property can be established. Indeed, as shown by the following example, the inclusion, the equality of event semantics, or the filtered event semantics do not imply the refinement relation.

**Example 89.** Let $A$ and $B$ denote respectively the following agents:

$A = \text{tell}_1(a); (\text{tell}_1(b) + \text{tell}_1(c))$

$B = (\text{tell}_1(a); \text{tell}_1(b)) + (\text{tell}_1(a); \text{tell}_1(c))$.

They both accept the same event semantics:

$\mathcal{E}(A) = \mathcal{E}(B) = \{a_1^+, b_1^+, \delta^+, a_1^-, c_1^+, \delta^+\}$.

If we consider the set of tokens $T = \{a, b, c\}$, it holds also that $\mathcal{E}_T(A) = \mathcal{E}_T(B)$. However, $A \not\mathcal{E}_T B$. Indeed $\text{tell}_1(b) + \text{tell}_1(c)$ does not refine $\text{tell}_1(b)$ nor $\text{tell}_1(c)$.

### 7.2. Denotational semantics and refinement

In the previous subsection, we have defined an operator $\nabla_T$ whose application to traces selects the events concerning tokens in $T$ and temporal events. Similarly, we define here an operator $\nabla_D^T$ whose application to sequences of steps selects those steps adding or removing tokens in $T$ as well as the temporal steps.

**Definition 90.**

(1) Let $T$ be a set of tokens. Define $\nabla_D^T : \text{Sfhist} \to \text{Sfhist}$ recursively as follows: for any stores $\sigma$, $\tau$, any time $u$, $v$ and any history $h$ in $\text{Sfhist}$:

$\nabla_D^T((\sigma_u, \delta^+)) = (\sigma_u, \delta^+)$

$\nabla_D^T((\sigma_u, \delta^-)) = (\sigma_u, \delta^-)$

$\nabla_D^T((\sigma_u, \tau_v), h) = \begin{cases} (\sigma_u, v).\nabla_D^T(h) & \text{if } u \neq v \text{ or } T \cap ((\sigma^* \setminus \tau^*) \cup (\tau^* \setminus \sigma^*)) \neq \emptyset \\ \nabla_D^T(h) & \text{otherwise.} \end{cases}$

(2) Let $T$ be a set of tokens. Define $\nabla_D^T : \mathcal{H} \to \mathcal{H}$ as the direct extension: for any set $S$ in $\mathcal{H}$,

$\nabla_D^T(S) = \{\nabla_D^T(h) : h \in S\}$.

Let us introduce a short notation corresponding to the application of $\nabla_D^T$ to the denotational semantics of an agent.

**Notation 91.** Let us denote by $\mathcal{D}_T$ the composition $\nabla_D^T \circ \mathcal{D}$.

Let us now relate this filtered denotational semantics to the filtered event semantics. The relation between the (unfiltered) denotational and even semantics is established in **Proposition 75** thanks to an operator $\beta$ whose application, roughly speaking, turns events into steps. Unfortunately this operator can not be directly reused to express the relation between $\mathcal{E}_T$ and $\mathcal{D}_T$. The reason of this lack may be understood as follows. On the one hand, we have seen that $\mathcal{E}_T$ is a filtered version of $\mathcal{E}$ selecting, with the temporal events, events related to tokens in $T$. Among those events some are checks of presence or checks of absence which do not modify the content of the store. On the other hand, $\mathcal{D}_T$ is a filtered version of $\mathcal{D}$ selecting, with temporal steps, the steps which modify the store by adding or removing tokens in $T$. It is worth noting that as regards the denotational semantics, checks of absence or presence of a token in $T$ appear as a step which do not modify the content of the store. Consequently, it cannot be distinguished from a check of absence or presence of an other token, or from an internal step of the agent and it is not possible to define a filter that preserves information about checks of absence or presence. Due to this different way of considering events which do not modify the store, the relation between $\mathcal{E}_T$ and $\mathcal{D}_T$ cannot be expressed through a direct functional reading of $\mathcal{E}_T$ but has to additionally cover events which do not modify the content of the store. This is achieved by combining $\beta$ with $\nabla_D^T$.

**Definition 92.** Let $T$ be a set of tokens. Define $\beta_T : \mathcal{P}(T) \to \mathcal{H}$, by $\beta_T \equiv \nabla_D^T \circ \beta$.

Using $\beta_T$ the relation between $\mathcal{E}_T$ and $\mathcal{D}_T$ is given by the following proposition.

**Proposition 93.** Let $T$ be a set of tokens. For any agent $A$, it holds that

$\mathcal{D}_T(A) = \beta_T \circ \mathcal{E}_T(A)$.

**Proof.** According to the definitions of $\mathcal{D}_T$, $\mathcal{E}_T$ and $\beta_T$ and **Proposition 75**, the proof of this proposition is obtained by establishing that for any trace $t$ in $\mathcal{E}(A)$, it holds that

$(\nabla_D^T \circ \beta)(t) = (\nabla_D^T \circ \beta) \circ \mathcal{D}_T(t)$. 
This is quite direct according to the intuitive reading of both sides of the equation. On the left-hand side, the trace is turned into a step reading by $\beta$ and then, in resulting histories, operator $\nabla^D_T$ selects steps that add or remove tokens in $T$ and temporal steps. On the right-hand side, the operator $\nabla^E_T$ firstly selects events related to tokens in $T$ and temporal steps. Then the functional reading and selection of steps that add or remove tokens in $T$ and temporal steps is applied in the same way as what is done in the left-hand side. As the non-temporal events and steps under consideration manipulate at most one token, the equivalence of the two sides is direct.

Thanks to this relation between $E_T$ and $D_T$, we can now easily express the relation between the refinement and the denotational semantics.

**Proposition 94.** For any agents $A$ and $B$ and any set of tokens $T$ such that $A \trianglelefteq_T B$, it holds that $D_T(A) \subseteq D_T(B)$.

**Proof.** This is a direct consequence of Propositions 93 and 88.

Fig. 4 summarizes, the relations between the denotational semantics, the event semantics and the refinement studied in this section. The filtered semantics $D_T$ and $E_T$ are respectively obtained from $D$ and $E$ through filtering with $\nabla^D_T$ and $\nabla^E_T$. The denotational semantics $D$ can be deduced from the event semantics $E$ by applying $\beta$. The filtered denotational semantics $D_T$ can be deduced from the filtered event semantics $E_T$ by applying $\beta_T$. The refinement relation implies the inclusion of filtered event semantics as well as the inclusion of filtered denotational semantics.

8. Conclusion

Building upon previous work on timed coordination languages, this paper has presented two tools to answer the question of safely replacing an agent by another one in any interacting context. On the one hand, a fully abstract semantics allows one to identify two processes who behave similarly in any context. On the other hand, a refinement theory allows one to compare processes that appear to be different in view of the fully abstract semantics but which satisfy the substitutability property: if the implementation $I$ refines the specification $S$ and if $C[S]$ is deadlock free, for some context $C$, then $C[I]$ is also deadlock free.

To our best knowledge, the article [30] is the only piece of work which has developed a refinement theory in the context of coordination languages. However, this work takes the complementary perspective of using a first order temporal logic to write specifications and of employing an axiomatic semantics to derive properties. Moreover, it uses a Prolog-like rule format for manipulating tuples. As appreciated by the reader, our work is based on an algebraic perspective. Accordingly, specifications and implementations are agents of the same language, and are related thanks to an abstraction operator and a refinement relation. Moreover, another family of coordination models, featuring Linda-like primitives, traditional concurrent operators and time, are tackled.

Refinements have been studied for classical concurrent languages. We have shown that trace refinement, underlying among others the B method [1], is not suited for our purposes. Moreover, refinement based on failure sets, classically used for process algebras such as CCS [24] and CSP [14], is also not adapted to our coordination context. We have thus refined the notion of refusal sets by replacing actions by tokens to be present or absent from the shared dataspace and have imposed restrictions on temporal transitions. The resulting refinement relation has then been shown to be adequate to obtain the substitutability property.

Because this property has a compositional flavor, it is expected that it will help to scale model checking. Our future research will aim to contribute to this area by building a tool similar to FDR [28] dedicated to our timed coordination languages.

Fully abstract semantics have been proposed for many different languages. As mentioned in Section 3, the traditional failure semantics cannot be reused in our asynchronous coordination context. The article [15] is one of the first to have...
defined a fully abstract denotational semantics for a asynchronous concurrent languages based on assignments of variables and if-then-else constructs as basic operations. Similar constructs have been presented in [9] for a general framework embodying a variety of concurrent languages all based on asynchronous communication. Our previous work [3,4] has built along these lines to prove that the two ideas of [15], on the one hand, of employing gaps in state sequences to represent possible interactions of agents with the state and, on the other hand, of the testing technique allow us to obtain a fully abstract compositional denotational semantics for Linda-like languages. In this paper, we have proved that these techniques can be enhanced to the context of temporal coordination languages. As the careful reader will have noticed, this has required several extensions at the level of the semantic domain, the semantics of the primitives and of the operators. As regards the semantic domain, two kinds of steps have been introduced in the denotational histories in order to reflect the fact that the transition system includes two kinds of transitions: on the one hand, computational ones which are allowed to update the store but in constant time and, on the other hand, temporal steps, which increase time by one unit and which results in the aging of one unit of the tokens on the store. Denotational histories have also been required to be time-continuous.

As regards the semantics of primitives, they have been required to capture the duration associated with the primitives. For the tell primitive, this is achieved by tagging the token added on the store with the duration of validity d. For primitives consulting the store (askd, naskd and getd) the associated duration of validity d indicates how many temporal steps can occur in the history before the primitive becomes an irretrievably failing one.

Concerning the semantic operators, they have been defined in order to respect the fact that temporal transitions can be followed only if no computational one is available. In particular for the parallel composition, the simple interleaving is re-defined in a more complex way to capture this particularity of temporal steps. It follows that the study of properties of the parallel composition is technically more complex.

Finally, the techniques of [3] have been reused but with some adaptations aiming at filling in the gaps of denotational histories. Thanks to the fact that denotational histories are time continuous, these “gaps to fill in” are actually time constant and some care has to be taken because of the fact that temporal steps have to occur in the two histories computed in parallel. The delay primitive is then introduced in the distinguishing context to allow the examined agent to follow a temporal transition.

This paper is an extended version of [17] where the refinement calculus was first introduced. It has been completed in this paper with a fully abstract semantics and a relation with the event semantics on which our refinement is based.

Acknowledgments

We would like to thank the University of Namur for its support as well as the anonymous reviewers for their helpful comments.

References