Logic for Mobility: a Denotational Approach

Joël-Alexis Bialkiewicz
UPMC Paris Universitas - LIP6

Frédéric Peschanski
UPMC Paris Universitas - LIP6

Keywords: logic for mobility, denotational semantics, proof normalization

Abstract. We describe a logical framework based on a denotational characterization of mobile process semantics. The logic builds on a trace model that associates to processes the sequences of actions one can observe from their behaviors. The basic predicates of the logic are properties about such sets of sequences. In order to support the reasoning on branching structures, a notion of structured location is used to decorate the observations made on process behaviors. The same notion of location is also employed to address some critical issues of mobile-calculi such as freshness and scope extrusion. We also develop a proof normalization procedure to increase the abstraction level of the logic.

Introduction

There are various forms of mobility involved in modern computing environments: mobile users, mobile devices (PDAs, smartphones, etc.) and (less frequently) mobile agents. There is a common denominator in these different point of views: the dynamic nature of the environment - the space - in which things move. A mobile user, for instance, sees its hardware/software environment changing while moving. Mobile devices also evolve in a dynamic environment, they must for example deal with constant disconnections and needs for reconnections. The various resource bindings of a mobile agent will also need to be updated upon migration. The mobile calculi, in particular the π-calculus [9], allow to model and reason about such dynamic perturbations in computer environments.

Various logics for mobility were proposed to formalize the reasoning about mobile systems: μ-calculi for the π-calculus [5], spatial logics [3], etc. In general, these logics correspond to variants of temporal logics, adopting a purely operational point of view. Reasoning in such logics is close to the
very fine grained abstraction level of the operational rules. In this paper, we adopt a complementary point of view with the description of a logical framework based on a denotational characterization of process semantics. In our opinion, this approach allows to increase the level of abstraction of the logic. A good example is the fixed-point characterization of recursive behaviors [8].

The approach we follow is similar to the logical characterization of the CSP trace model [8]. Similarly, the logic builds on a trace model that associates to processes the sequences of actions one can observe from their behaviors. The basic predicates of the logic are properties about such sets of sequences. The main difficulty, though, is to design a trace model expressive enough to encompass the features of mobile calculi such as the π-calculus. First, standard trace models do not take into account the branching structure of process behaviors. Bisimulation techniques are proposed to overcome this limitation, but they also require to adopt the operational point of view. In [11] we show that the branching structure of process behaviors can be encoded together with the observations using a notion of location. A more structured notion of location can be employed to also address some critical issues of mobile calculi such as freshness and scope extrusion. A location identifies a single point in the space-time geometry of process semantics (two distinguished observations can not be made at the same location), it is thus a good source of uniqueness. Consequently, fresh names are easy to generate in a denotational way. The language we use to describe mobile process behaviors can be seen as a variant of the π-calculus with some subtleties (most notably the generalization of match and mismatch to first-order guards). Alternatively, given its denotational treatment, the language can be interpreted as a mobile variant of CSP.

Two main contributions are emphasized in this paper. To start with, we provide the first logical characterization of mobile calculi in a purely denotational way, at least from what we know. Some works propose alternative denotational semantics for mobile-calculi [2, 6] but they remain very close to the operational point of view, and we are not aware of any logical characterization as of today. An important part of the logic is the location model that allows to solve, at once, various critical aspects of process semantics: encoding of the branching structure, denotational interpretation of scope extrusion, characterization of recursion as fixed points, etc. The second contribution emphasized in the paper is the definition of a proof normalization procedure for the logic. This is in our opinion a good illustration of the abstraction potential offered by the denotational point of view.

The outline of the paper is as follows. In section 1 we present the model underlying the logical framework. This concerns the syntax of process expressions and the informal presentation of the denotation in which these processes are interpreted. The logical framework is then described in section 2. Each non-trivial rule is discussed in detail, and examples of proof trees are provided. The proof normalization procedure is discussed in section 3. A companion technical report with most of the proof details is also available [12].

1. The model

1.1. The language

We first briefly overview the syntax of our study language. As shown in Table 1, the focus of the study is a variant of the π-calculus [9].

Two parallel processes, e.g. \( P \parallel Q \), can communicate with each other using the prefixes of output \( c!v \) (emitting \( v \) along \( c \)) and input \( c?v \) (receiving along \( c \) of a name bound to \( x \) in the continuation). For example, the processes \( cd \parallel c?!x.x!e \) can synchronize and continue as \( d!e \). Such a possibility of using a communicated name (\( d \) in the example) as a channel, a phenomenon designed as mobility, gives the π-calculus much of its expressive power (and also the complexity of its semantics). The
expression \((\nu n) P\) denotes a scope restriction for name \(n\) in process \(P\). If the name \(n\) is emitted at some point in \(P\), it is said to escape its scope, which is then extruded so that it also encloses potential receivers. The sum \(P + Q\) is the non-deterministic choice between two processes. As in CSP an explicit recursion operator is introduced. The notation is standard, e.g. \(\mu X. (\nu c) g! c.X\) denotes an infinite generator of fresh names on channel \(g\). A process behavior \(P\) can be conditioned by a guard \(\phi\) expressed in a subset of first-order logic, which we note \([\phi] P\). The basic propositions of the guards are based on match \([a = b]\) and mismatch \([a \neq b]\) expressions, used to compare names for respectively equality and inequality\(^1\).

### 1.2. The trace semantics

Our objective is to obtain a trace-based semantic characterization of mobile process behaviors. There are a few issues to address while doing so. First, trace models such as [8] do not capture the branching structure of behaviors. In logical terms, only safety properties can be expressed and reasoned about. As explained in [11] it is possible to encode the branching structure within traces by marking the non-deterministic branches of the behaviors using locations.

Consider the examples of Figure 1. The two labeled transition systems (LTS) at the top of the figure represent two distinct process behaviors. In the left process there is a non-deterministic choice between two possible continuations starting with an \(\alpha\) action. In the right process the action \(\alpha\) is first performed and then an external choice is performed for either the action \(\beta\) on the left or \(\gamma\) on the right. In standard trace semantics these two behaviors are not distinguished, and it is one of the main argument to rely on bisimulations to compare process behaviors. As shown below these two examples, our trace characterization provide an encoding of the branching structure using locations.

\(^1\)Historically, the mismatch operator was not part of the \(\pi\)-calculus; see [2] for an argument in favor of its introduction.
This encoding can be reinterpreted in terms of LTS, as seen on the bottom part of the picture. In this model, it is clear that the two behaviors are distinguished, as expected.

But the characterization of the branching structure is not the only problem we have to face. The semantics of mobile calculi such as the $\pi$-calculus and its variants introduce history dependence [10] in the semantics. A first example is when some data is received from the environment, e.g. in a prefix $c?x$ where $x$ must be bound to “something” we do not really know about. For example if this input is followed by a match $[x = y]$ then we must “remember” $x$ was bound and that we must now assume it is equal to $y$. Another example is when a private name $n$ is emitted to the environment, e.g. in a prefix $c!n$ under a restriction $\nu(n)$. Now the name $n$ is not private anymore because it can be received by external processes, but it is neither public because it can only be known by those external processes which actually receive the name. Once again this introduces an history-dependence in the behavior since we have to remember than $n$ escaped the process, and also when it escaped. Moreover, this name must be guaranteed fresh, i.e. unique up-to any context in which the behavior can be observed. This freshness guarantee is difficult to model except in a symbolic way using scope extrusion laws [9]. The other issue we have to deal with is the interpretation of match and mismatch (or any combination of these), which is quite easy in symbolic terms [7] but much less so when considering a denotational interpretation.

Interestingly, we use the very same idea of location to solve most of these issues at once. Of course, we need a slightly more structured notion of location than e.g. [11].

**Definition 1.1.** An observation $\alpha ::: l$ is either an observable (input or output) action $\alpha$ or the termination observation $\checkmark$, decorated with a relative location $l$.

**Definition 1.2.** A relative location $l$ is defined by the following grammar where $\epsilon$ is the origin, $s$ a strong locator, $w$ a weak locator and $\varphi$ a logical guard on names:

$$
\begin{align*}
    l & ::= \epsilon \mid \tilde{\lambda} \\
    \tilde{\lambda} & ::= \lambda \mid (\varphi, w)\tilde{\lambda} \\
    \lambda & ::= (\varphi, s)
\end{align*}
$$

For the sake of brevity, weak and strong locators may appear without their guard if it is true.

**Definition 1.3.** Let $i, j$ be integers such that $1 \leq i \leq j$. A locator is either a strong locator $\diamondsuit^i_j$, a weak locator $\tilde{\diamondsuit}^i_j$ or the origin locator $\epsilon$. $\triangleright$ and $\triangleright$ stand for $\diamondsuit^1_1$ and $\tilde{\diamondsuit}^1_1$ respectively.

First, as illustrated in Figure 1, this location model is expressive enough to encode the branching structure of process behaviors. Each branch is decorated by a distinct split location $\diamondsuit^n$ where $n$ is the total number of branches and $i$ the index of the branch. Moreover, each name received from the environment through a binder $x$ is named $\rho_T$. The absolute location $\tilde{\ell}$ can be reconstructed from the (relative) location $l$ of an observation. This absolute location – we explain its construction below – is guaranteed unique in the whole process behavior. This implies the freshness of the name $\rho_T$ received from the external environment. All free occurrences of $x$ are thus replaced by $\rho_T$ and thus the binder “vanishes” as well as the history-dependence issue. For bound output (output of a private name) the treatment is similar. Any private name $n$ that is emitted to the environment is replaced by an escaped name $\nu_T$ also guaranteed fresh by construction.

The remaining part of the trace model is quite standard, indeed very close to the CSP trace model [8]. A trace set $T$ is composed of a potentially infinite set of observation sequences. The empty sequence is noted $\langle \rangle$ and $\langle \alpha_1::l_1, \alpha_2::l_2, \ldots, \alpha_n::l_n \rangle$ denotes a non-empty sequence of finite size $n$ where the $\alpha_i$’s are the recorded actions at their respective location $l_i$. 
Definition 1.4. The absolute location of observation $\alpha_n$ within sequence $\langle \alpha_1::l_1, \alpha_2::l_2, \ldots, \alpha_n::l_n, \ldots \rangle$ is the concatenation $l_1.l_2\ldots l_{n-1}.l_n$.

Not all sets of sequence are valid traces, it is thus important to characterize precisely the structure of the set $T$ of all possible traces. This structure relies on the following axiomatic constraints:

- **[fin]** \( \forall T \in T, S \text{ is finite} \)
- **[pref]** \( \forall S \in T, \forall S' \leq S, S' \in T \)
- **[loc]** \( \forall \alpha::l_1, \beta::l_2 \in T, T_1 = T_2 \implies \alpha = \beta \)
- **[move]** \( T\{(\varphi, o^n) \mapsto (\psi, o'^n)\}_i = T \iff \varphi \iff \psi \)

The [fin] axiom says that all recorded sequence are finite, which is sufficient to capture finitely branching (and potentially infinite) behaviors, a common assumption of process algebras\(^2\).

The prefix-closure property [pref] is fundamental for the extraction of least fixed points from trace behaviors. The combination with [fin] means that a sequence $S$ only has a finite number of prefixes, and thus the prefix order in traces is well founded.

The axiom [loc] ensures that two observations made at the same absolute location are indeed the same.

The last axiom, named [move], explains that the different branches of a given choice point in the branching structure can be safely commuted (i.e. the local ordering of branches is not relevant). The axiom relies on an operator for the commutation of branches, defined as follows (where $\bullet$ is a dummy location distinct from $l_1$ and $l_2$).

**Definition 1.5.** \( T\{l_1 \leftrightarrow l_2\}_i \overset{\text{def}}{=} T\{l_1 \mapsto \bullet\}_i\{l_2 \mapsto l_1\}_i\{\bullet \mapsto l_2\}_i \)

The notation $T\{l_1 \leftrightarrow l_2\}_L$ describes the local relocation of a trace. For all sequences, the absolute location $L$ is followed from the initials of $T$ until we reach a given observation $\alpha::l$. In this case, the component $l_1$ of the location $l$ is locally changed to $l_2$ in $T$ and the procedure ends. Note that if the strong locator at the end of an observation’s location has been removed, the observation is changed to a termination. If no such observation is reachable in a given sequence, it is left unchanged; this extends to trace $T$ itself. The received and escaping names, whose freshness is guaranteed by absolute (and thus unique) locations, must be renamed when branches are relocated. This explains the relative technical complexity of the definition, which can be found in the associated technical report [12].

Another useful notation is the post-trace $T \upharpoonright l$ of a trace $T$ at a location $l$. Likewise, the absolute location $L$ is followed from the initials (i.e. the heading observations) in all the sequences of $T$ until we reach a given observation $\alpha::l_0$ such that $\overline{l_0} = l^\upharpoonright l_0$. The post-trace at $l$ is the subtrace with that observation as initial. It corresponds to selecting a subtree in the process branching structure, keeping only the behavior which can be observed after first following a given execution path. The post-trace notation is especially useful to define trace predicates, as explained in the next section.

An important lemma can be demonstrated thanks to the axioms given above.

**Lemma 1.1.** In a trace $T$ any absolute location $l$ is unique

---

\(^2\)Models of infinite traces allow to capture behaviors with unbounded non-determinism [13]. But this heavily impacts the semantics and the proof techniques we develop.
In any sequence $S$ of $T$ where a location $l$ appears, $l$ is of course unique because the concatenation operator generates strings of increasing length. The $[fin]$ axiom ensures $\bar{l}$ is of a finite length. Now, consider two sequences $S_1$ and $S_2$ where respectively $\bar{l}_1$ and $\bar{l}_2$ can be constructed. If $\bar{l}_1 = \bar{l}_2$ then the two actions observed at $\bar{l}_1$ in $S_1$ and at $\bar{l}_2$ in $S_2$ are identical according to the $[loc]$ axiom. By $[pref]$ we can infer $S_1$ and $S_2$ are identical or related in the prefix ordering, up to the exchange of split location indices (according to $[move]$).

Note that the $[move]$ axiom is not required to ensure the uniqueness of absolute locations. Indeed, an interesting variant of the model can be obtained by removing the ability to commute branches in the processes behaviors. In this variant the notion of prioritized choice becomes natural, the ordering of split locations can be used to discriminate the branches in an implicit way. The more thorough study of this variant is left as a future work.

2. Logical framework

A formula $\varphi$, $\psi$ is either

- a trace predicate $p(T) \in \{true, false\}$ where $T$ is a trace set
- a negation of a formula $\neg \varphi$
- a conjunction of formulas $\varphi \land \psi$
- a disjunction of formulas $\varphi \lor \psi$

To avoid the introduction of dedicated rules in the logic, the negation is defined as a derived operator:

- $\neg p(T) \overset{\text{def}}{=} \overline{p(T)}$ (set complement)
- $\neg \neg \varphi \overset{\text{def}}{=} \varphi$
- $\neg (\varphi \land \psi) \overset{\text{def}}{=} (\neg \varphi) \lor (\neg \psi)$
- $\neg (\varphi \lor \psi) \overset{\text{def}}{=} (\neg \varphi) \land (\neg \psi)$

We write $P \models \varphi(T)$ iff the traces $T$ of the process expression $P$ satisfy the formula $\varphi(T)$. We can consider arbitrary trace predicates, i.e. arbitrary functions from trace sets to $\{true, false\}$. An example of such a predicate is $true$ such that $\forall T$, $true(T) = true$.

2.1. Basic rules

The first schema axiom of the logic gives a meaning to the $true$ predicate, explaining that any process $P$ satisfies $true(T)$ for any trace set $T$.

$$P \models true(T) \ \text{truth}$$

An important rule allows to weaken a specification:

$$P \models \varphi(T) \ \forall T, \ varphi(T) =\Rightarrow \psi(T) \ \text{weaken}$$

The rule for conjunction is the usual one:
There are also two rules for disjunction:
\[
\begin{align*}
P \models \varphi(T) & \quad \frac{}{P \models \varphi(T) \lor \psi(T)} \text{disj}_L \\
P \models \psi(T) & \quad \frac{}{P \models \varphi(T) \lor \psi(T)} \text{disj}_R
\end{align*}
\]

2.2. Process rules

We now describe the rules of the proposed logic that characterize the semantics of process expressions.

The basic deadlock process is characterized by a void execution:
\[
\bot \models T = \{\emptyset\} \text{ dead}
\]

For the terminating process we have the following axiom:
\[
0 \models T = \{\langle \checkmark :: \epsilon \rangle, \emptyset\} \text{ term}
\]

These axioms play an important role in the logic since they are required to finalize any proof about deadlock or correct termination. For instance, any process \( P \) terminates correctly iff there is a location \( l \) at which \( P \models T \downarrow l = \{\langle \checkmark :: \epsilon \rangle, \emptyset\} \). Similarly, a process deadlocks at a location \( l \) iff \( P \models T \downarrow l = \{\emptyset\} \).

The silent step \( \tau \) is characterized as follows:
\[
P \models \varphi(T) \quad \frac{\tau.P \models \exists T', T = T'\{\mathrm{true, } \triangleright\} \land \varphi(T')}{\text{silent}}
\]

This means that in order to prove that \( \tau.P \) satisfies a formula \( \varphi \) at a location \( l \), we have to prove that the process \( P \) satisfies \( \varphi \) at a location next to \( l \). The notation \( \triangleright \) means that the next location corresponds to a weak branching in the trace. This characterizes the fact that an internal step can play a role in the behavior of a process, but only in the branching structure captured by locations.

For output and input the rules are as follows:
\[
P \models \varphi(T) \quad \frac{c!a.P \models \exists T', T = \{c\}![a]:(\mathrm{true, } \triangleright).T' \land \varphi(T')}{{output}}
\]
\[
P \models \varphi(T) \quad \frac{c?x.P \models \exists T', T = \{c\}?:(l = (\mathrm{true, } \triangleright)).T'(\rho_l/x) \land \varphi(T')}{{input}}
\]

For an output the initials of a non-empty trace (i.e. the set of the heads of its non-empty sequences) must be the output itself. The continuation must also satisfy the property \( \phi \). Input is more involved because it introduces a binder that must be interpreted in the denotation. The idea is to introduce a name \( \rho_l \) that is bound to the absolute location ending in \( l \). This name is guaranteed fresh by
construction, because (absolute) locations are unique in traces (by Lemma 1.1). This name must replace all the occurrences of the bound name \( x \) in the continuation of the behavior.

The language allows to prepend any process expression by a guard represented as a first-order logical formula where the two basic conditions are match and mismatch conditions.

\[
P \models \varphi(T) \quad \sigma \overset{def}{=} \{ a \cup b/a, a \cup b/b \mid g \rightarrow a = b \}
\]

This rule expresses that guards are considered equal if they are equivalent in the (classical) logical sense. The non-trivial substitutions involved in the rule are required because the match conditions in guards are interpreted as constructions of equivalence classes of the equated names. For example a guard \( [a = b] \) will be interpreted as the construction of the equivalence class \( \{ a, b \} \).

The restriction operator has a rather complex denotations. Similarly to the input prefix, it serves as a binder. But it is also the creation of a private name only known to the local process which created it. Moreover, it is possible that in the continuation of this process the name escapes its scope and becomes shared with other processes that receive it later on. Providing a general characterization of all these features in a denotational way at once is quite challenging. This is reflected by the rule for restriction written as follows:

\[
P \models \varphi(T) \quad (\nu n)P \models \exists T', T = \{ \alpha::(g \wedge \psi, \lambda)L.S \mid \alpha::(\psi, \lambda)L.S \in T' \sigma \} \wedge \varphi(T') \overset{res}{=} \hline
\]

The general principle underlying the record of the observational effects of restriction - where much of the inherent complexity of mobility is concentrated - is to substitute all occurrences of fresh names that may be output on non-private channels by names that we say are escaping their scope. This is performed by the escaping function \( \mathcal{E}_n \) where \( n \) is the name that escapes its current scope. The formal definition of the escaping function is provided in Table 2. The principle is to cut short the sequences from the point where they may not interact with their environment any longer (that is, from any action whose subject is an occurrence of the restricted name \( n \) that is not escaped), and also finding the actions where the restricted name indeed escapes. Each escape is effected by replacing all the free occurrences of \( n \) by a fresh binder \( \nu_T \) from this point on. Since absolute locations are unique by construction it is ensured that those names will be absolutely fresh. Two auxiliary functions are used to apply the logical consequences of restriction to the process behavior: a restricted name cannot be equal to any name except if it has been received after it escaped, and an observation guarded by \( false \) never takes place.

The rule for the choice (sum) and parallel operators are as follows:

\[
P \models \varphi(T) \quad Q \models \psi(T) \quad P + Q \models \exists T_1, T_2, T = T_1 \oplus T_2 \wedge \varphi(T_1) \wedge \psi(T_2) \overset{sum}{=} \hline
\]

\[
P \parallel Q \models \exists T_1, T_2, T = T_1 \oplus T_2 \wedge \varphi(T_1) \wedge \psi(T_2) \overset{par}{=} \hline
\]

These rules rely on operators that are similar to disjoint unions (operator \( \oplus \) for sum) and interleaving (operator \( : \) for parallel). The usual set-theoretic operators must be adapted to the particular structure of the trace sets we consider. First, we must for instance preserve the basic axioms as well as the location model. Moreover, as in CSP we must also interpret the potential communications involved when putting processes in parallel. The detailed definitions of the operators are given in Table
The rule can be trivially extended to vectors of fixed points, allowing to characterize mutual recursion. The issue of compositionality is discussed more thoroughly in the technical report [12].

The final rule is for recursion, it is a standard characterization (see e.g. [8]):

\[
\forall Y, \ Y \models \varphi(T) \implies P(\tau.Y/X) \models \varphi(T) \quad \varphi(\langle \rangle) \quad \text{rec}
\]

### 2.3. Examples

We give two examples of simple deductions using the proposed logic. The proof trees are almost complete, only a few trivial steps are omitted. Note that in order to save some space, the formula to prove is always noted \( \phi(T) \) in the proof trees.

Let the counting operator be defined as follows:

\[
\langle \rangle[\alpha] \overset{\text{def}}{=} 0, \quad S[\alpha] \overset{\text{def}}{=} \text{tl}(S)[\alpha] + \begin{cases} 1 & \text{if } \text{hd}(S) = \alpha \\ 0 & \text{otherwise} \end{cases}
\]

The classical example of a predicate is \( \forall S \in T, \ S[alb] \leq S[alc] \), which means that in every sequences of the trace there must be less instances of \( alb \) than of \( alc \).

Now let us try to prove that \( P = alc.alb.0 \) satisfies this predicate. The proof tree is as follows:

<table>
<thead>
<tr>
<th>Step</th>
<th>Formula</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ( \models ) ( T = (\langle \rangle, \langle \rangle) )</td>
<td>( \text{true} )</td>
<td>( \text{term} )</td>
</tr>
<tr>
<td>0 ( \models ) ( T = (\langle \rangle, \langle \rangle) )</td>
<td>( \models ) ( T = (\langle \rangle, \langle \rangle) )</td>
<td>( \text{true} )</td>
</tr>
<tr>
<td>0 ( \models ) ( T = (\langle \rangle, \langle \rangle) )</td>
<td>( \models ) ( \forall S \in T, \ S[alb] \leq S[alc] )</td>
<td>( \text{weaken} )</td>
</tr>
<tr>
<td>( acl.alb.0 \models \exists T', T = acl.alb.0.T' \land \forall S \in T', S[alb] \leq S[alc] )</td>
<td>( a lb.0 \models \forall S \in T, \ S[alb] \leq S[alc] + 1 )</td>
<td>( \text{weaken} )</td>
</tr>
<tr>
<td>( acl.alb.0 \models \exists T', T = acl.alb.0.T' \land \forall S \in T', S[alb] \leq S[alc] + 1 )</td>
<td>( acl.alb.0 \models \forall S \in T, \ S[alb] \leq S[alc] + 1 )</td>
<td>( \text{weaken} )</td>
</tr>
</tbody>
</table>

An interesting aspect of the logic is that we may express properties that can not be verified on individual sequences only. For instance, we can check that at any point a process may terminate. A simple way to express this fact is to say that for every sequence that does not terminate in the trace, the latter also contains another sequence with the same prefix but ending with a correct termination. Now formally, this is \( \forall S \in T \setminus \{\langle \rangle\}, \ \exists S' \in T, \ \exists S'', \ S' = S'' \land \langle \langle \cdot \cdot \cdot \rangle \rangle \in S'' \).

The proof tree showing that \( \mu P.((\mu n)\text{gen!n.P + stop?}.0) \) satisfies this property is given in Figure 2.

### 2.4. Trace transformations

The logic with the rules as defined above cannot be used to assert some important basic properties. For example, one cannot prove \( \models G[\alpha].P \models \phi \), for an arbitrary formula \( \phi \), that \( [G[\alpha].G]P \models \phi \).

\( \text{The rule can be trivially extended to vectors of fixed points, allowing to characterize mutual recursion.} \)
\[ T \oplus \{ \langle \rangle \} = \{ \langle \rangle \} \oplus T = \{ \langle \rangle \} \otimes \{ \langle \rangle \} = \{ \langle \rangle \} \oplus T = T \]

\[ \text{[t-sum]} \quad T_1 \oplus T_2 \overset{\text{def}}{=} T_1 \{ (\varphi, s^n_i) \mapsto (\varphi, s^n_{i+1}) \} \cup T_2 \{ (\psi, s^m_j) \mapsto (\psi, s^m_{j+1}) \} \]

\[ \text{[t-par]} \quad T_1 \otimes T_2 \overset{\text{def}}{=} (T_1 \otimes T_2) \oplus (T_2 \otimes T_1) \oplus (T_1 \otimes T_2) \oplus (T_2 \otimes T_1) \]

\[ \text{[neutral]} \quad T \oplus \{ \langle \rangle \} = \{ \langle \rangle \} \oplus T = (T \otimes \{ \langle \rangle \}) = \{ \langle \rangle \} \otimes T = T \]

\[ \text{[t-prod]} \quad \bigcup_{i=1}^{n} \alpha_i \cdot l_i \cdot T_i \bigcup_{j=1}^{m} \alpha'_j \cdot l'_j \cdot T'_j \overset{\text{def}}{=} \bigcup_{a=1}^{n} \bigcup_{b=1}^{m} \alpha_a \cdot l_a \cdot (T_a \oplus \alpha_b \cdot l'_b \cdot T'_b) \]

\[ \text{[t-comm]} \quad \bigcup_{i=1}^{n} \alpha_i \cdot l_i \cdot w_i(\varphi_i, \varphi'_i) \cdot T_i \bigcup_{j=1}^{m} \alpha'_j \cdot l'_j \cdot w'_j(\psi_j, \psi'_j) \cdot T'_j \overset{\text{def}}{=} \bigcup_{a=1}^{n} \bigcup_{b=1}^{m} \alpha_a \cdot l_a \cdot w_a l'_b \overset{\text{def}}{=} \bigcup_{k=1}^{n} \alpha_k \cdot l_k \cdot T_k \]

\[ \text{[escaping]} \quad \mathcal{E}_n(\langle \rangle) \overset{\text{def}}{=} \emptyset \]

\[ \mathcal{E}_n(\alpha \cdot l \cdot S) \overset{\text{def}}{=} \left\{ \begin{array}{ll}
\alpha \cdot l \{ \text{false} / (n = x) \} . S & \text{if } \text{grd}(l) \implies n = x \\
\alpha \cdot (\text{false} \land \varphi, \lambda) \cdot L & \text{if subj} (\alpha) = n \text{ and } l = (\varphi, \lambda) \cdot L \\
\mathcal{F}_n^L(\alpha \cdot l \cdot S) \{ \nu_l / n \} & \text{if } \alpha = c \cdot n \text{ and } c \neq n \\
\alpha \cdot l \cdot \mathcal{E}_n(S) & \text{otherwise} \end{array} \right. \]

\[ \mathcal{F}_n(\langle \rangle) \overset{\text{def}}{=} \emptyset \]

\[ \mathcal{F}_n(\alpha \cdot l \cdot S) \overset{\text{def}}{=} \left\{ \begin{array}{ll}
\alpha \cdot l \{ \text{false} / (n = x) \} . S & \text{if } \text{grd}(l) \implies n = x \text{ and not } x = \rho_m, L \leq m \\
\text{or else } \alpha \cdot l' \cdot \mathcal{F}_n(S) \text{ with} \quad l' = \left\{ \begin{array}{ll}
l \{ \text{true} / (n \neq x) \} & \text{if } \text{not } x = \rho_m, L \leq m \\
l & \text{otherwise} \end{array} \right. \end{array} \right. \]

\[ \text{[location functions]} \quad f(\emptyset) \overset{\text{def}}{=} \emptyset \]

\[ f((\varphi, l) \cdot L) \overset{\text{def}}{=} \left\{ \begin{array}{ll}
\emptyset & \text{if } \varphi \iff \text{false} \\
l \cdot f(L) & \text{otherwise} \end{array} \right. \]

Table 2. Operators on trace sets
Figures 2. Proof tree for property $\psi(T) = \forall Y \in T, \exists S' \in T, \exists S'', S' = S'' \land (\forall : e) \in S''$

Guards also introduce similar discrimination if the trace sets are interpreted too strictly in the logic. To overcome these limitations, the general idea is to allow a certain number of transformations that preserve the branch structure and observational content of process behaviors.

**[merge]**
\[
\forall s, s' \in \{g, \overline{g}\}, T\{((\varphi, s^n_o) \rightarrow (\varphi \lor \psi, s^n_b))\}l_i \{[(\xi_i, s^n_i)l_i \leftarrow \xi_i, s^{n-1}_i]l_i \forall i \geq a\}L
\]

**[perco]**
\[
T\{(\varphi, \psi, l_1) \rightarrow (\psi, l_1)\}L \text{ if } \text{grd}(T) = \varphi
\]

**[weak false]**
\[
T\{l_f \rightarrow (\psi, s^n_{l_f})\}L \text{ where } \text{grd}(T_f) = \varnothing, \text{hd}(l_f) = (\varphi, s^n_k)
\]

**[strong false]**
\[
\forall i > k, T\{((\varphi, s^n_i) \leftarrow (\varphi, s^{n-1}_i) \forall i < k\}L_i\{(\varphi, s^n_i) \leftarrow (\varphi, s^{n-1}_i) \forall i > k\}L_i,
\]

\[
\text{grd}(T_f) = \varnothing \text{ and } \text{fr}(T_f) = \emptyset
\]

Table 3. The transformations allowed on traces

These transformations, described in Table 3, heavily rely on trace relocation. The function \(\text{fr}\) (cf. Table 2) is also employed to remove from a location the locators that are conditioned by \(\varnothing\).

Despite their somewhat technical definition, the transformation are conceptually simple:

**Merge** The transformation merges two distinct branches (identified by two different \(s^n_o\) and \(s^n_b\) strong or weak split locators) at a given location \(l\) whenever they exhibit the same behavior. Put in other terms, one of them will be deleted, and all the other branches at location \(l\) will be renumbered so that split locator numbering remains consistent. In terms of processes, *merge* transforms \(P + P\) into \(P\) compositionally.

**Perco** Guards are logical conditions that control whether an action is allowed. Since actions are treated sequentially, a guard is implicitly in conjunction with all the previous guards of its prefix sequence. For that reason, any guard which is already implied by the ones of the absolute location \(\text{fr}\) where it appears will be removed. In terms of processes, *perco* transforms \([G] \alpha. [G]P\) into \([G] \alpha. P\).

**Weak false** If the full (absolute) guard of an observation’s strong locator is false, then this observation will never be reached, and it does not belong to the process behavior. However, if at least
one of the internal actions included in the observation (as weak locators) is reachable then a deadlock condition exists, which must be recorded properly. In terms of processes, \textit{weak false} transforms \( \tau.\{\text{false}\}P + Q \) into \( \tau.0 + Q \).

\textbf{Strong false} This complements the previous one when every single locator of an observation is unreachable, in which case the whole observation must be removed. This consists in removing all the branches starting at the disabled location, up-to the renumbering of split locators for consistency. In terms of processes, \textit{strong false} transforms \( \{\text{false}\}P + Q \) into \( Q \). Note the use of function \( f \) to calculate the reachable part of a location, may be found at the bottom of Table 2.

In order to offer the possibility of transforming traces within proof trees, we introduce the following inference rules:

\[
\begin{align*}
\frac{[G \land H]P \models \varphi(T)}{[G]P + [H]P \models \varphi(T)} \quad \text{\textit{merge}}
\quad & \frac{[G]\alpha.P \models \varphi(T)}{[G]\alpha.\gamma P \models \varphi(T)} \quad \text{\textit{perco}}
\quad & \frac{\bot \models \varphi(T)}{[\text{false}]P \models \varphi(T)} \quad \text{\textit{false}}
\end{align*}
\]

\subsection{Weak Satisfaction}

Our logic as it was presented does not abstract the internal actions \( \tau \) as captured by weak locations. Unlike CCS it is already the case that no additional observation is added to the trace when an internal action is encountered, but the weak locators that record them are concatenated. It is possible to abstract a little bit more from the weak locators by avoiding to count those that do not contribute to the branching structure. The following rules can optionally be added to the logic:

\[
\begin{align*}
\frac{P \models \varphi(T)}{P \models \varphi(T)} \quad \text{\textit{wsat}}
\quad & \frac{\tau.P \models \varphi(T)}{\tau.\tau.P \models \varphi(T)} \quad \text{\textit{compress}}
\end{align*}
\]

A process \( P \) is said to weakly satisfy property \( \varphi(T) \) iff \( P \models \varphi(T) \). This allows to only take into account the internal actions that have an impact on the process branching structure, without sacrificing compositionality and the possibility of reasoning about divergence. For a more in depth argument on this issue, see [12].

As an example consider a property saying that every observable action is preceded by one internal action at most. One way to define it would be \( \forall S \in T, \alpha:l \in S \implies \text{length}(l) \leq 2 \) since the relative location of an observation contains a strong locator (or \( \epsilon \) in the case of \( \checkmark \)) and as many weak locators as internal actions precede its observable action.

It is clear that \( \tau.\alpha.l.0 \not\models \forall S \in T, \alpha:l \in S \implies \text{length}(l) \leq 2 \) since the action \( \alpha.l.0 \) will be located at \( \lll \). However, since there is no choice involved at the internal actions place in the branching structure it is possible to prove that \( \tau.\alpha.l.0 \models \forall S \in T, \alpha:l \in S \implies \text{length}(l) \leq 2 \). In this case, the property does not read that every observable action is preceded by at most one internal action, but that it is preceded by at most one internal action that impacts the branching structure. The quite simple proof tree is as follows:
\[ 0 \models T = \{ (\checkmark : \epsilon), \emptyset \} \text{ term} \]

\[ 0 \models \forall S \in T, \alpha :: l \in S \Rightarrow \text{length}(l) \leq 2 \text{ weaken} \]

\[ a!b.0 \models \exists T', T = \{ a \}!(b) :: S \land \forall S \in T', \alpha :: l \in S \Rightarrow \text{length}(l) \leq 2 \text{ output} \]

\[ a!b.0 \models \forall S \in T', \alpha :: l \in S \Rightarrow \text{length}(l) \leq 2 \text{ weaken} \]

\[ \tau.\alpha!b.0 \models \exists T', T = T' \{ (\checkmark : (\text{true}, \checkmark)) \} \land \forall S \in T', \alpha :: l \in S \Rightarrow \text{length}(l) \leq 2 \text{ silent} \]

\[ \tau.\alpha!b.0 \models \forall S \in T', \alpha :: l \in S \Rightarrow \text{length}(l) \leq 2 \text{ weaken} \]

\[ \tau.\tau.\alpha!b.0 \models \forall S \in T', \alpha :: l \in S \Rightarrow \text{length}(l) \leq 2 \text{ compress} \]

The obvious side conditions of [weaken] are omitted: since the maximum number of weak locations is 1 before [wsat] the property is always satisfied.

3. Proof normalization

3.1. Trace normalization

A directed interpretation of the transformations of Table 3 allows to define a rewrite system [1].

**Definition 3.1.** A **trace rewrite** is a triple \((T, \mathcal{U}:l, T')\) with \(T'\) the result of applying the transformation \(\mathcal{U}\) (excluding identity) on the subtrace of \(T\) at the absolute location \(l\). We also use the notation \(T \xrightarrow{\mathcal{U}:l} T'\). If the considered rewrite is not possible at the given location, we note \(T \xrightarrow{\mathcal{U}:l} \).

An arbitrary rewrite (at an arbitrary location) is noted \(T \xrightarrow{} T'\). If a trace \(T\) is such that \(T \xrightarrow{} \) then \(T\) is said in normal form, which is noted \(\hat{T}\).

As explained above, the normal form of a trace \(T\), denoted \(\hat{T}\), is obtained when no further rewrite is possible. Within a proof tree, this means it is not possible to apply the rewrite rules anymore. The problem with the individual application of the rewrite rules is that their may be an arbitrary number of places where they can be applied (because a trace can contain an infinite number of sequences). This can lead to infinite proof trees in the logic, and thus artificially undecidable properties.

It is possible to take advantage of the finitely branching structure of behaviors as well as the well-foundedness of the prefix ordering on locations to uncover a weak termination property of the normalization process. For this we must introduce a higher-level notion of parallel rewrite, which consists in applying simultaneously all the independent rewrites that can be applied on a given trace. Two rewrites are strongly independent if they apply at unrelated locations (with respect to the prefix-ordering on locations), and weakly independent if their location is comparable but they can be applied in an arbitrary order.

**Definition 3.2.** A single parallel rewrite of a trace \(T\) is a triple \((T, \Upsilon, T')\) with \(\Upsilon\) the set of all the independent rewrites applicable on \(T\). The triple is noted \(T \xrightarrow{\Upsilon} T'\).

In the companion technical report [12] we motivate the introduction of this parallel rewriting relation by showing that the order of application of independent rewrites is not significant wrt. trace set
identity. Put in other terms the trace rewrites are locally confluent and enjoy a \textit{diamond property}. This gives the parallel rewrite systems a decisive weak termination property.

\textbf{Lemma 3.1.} The parallel rewrite of a trace $T$ is terminating, i.e the descending chain $T \xrightarrow{\Upsilon_1} T' \xrightarrow{\Upsilon_2} T'' \ldots \xrightarrow{\Upsilon_n} \hat{T}$ is terminated.

This can be demonstrated by an induction on the structure of locations in trace $T$. The important step is the fact that a parallel rewrite $\Upsilon_{k+1}$ can only be performed at locations that are prefixes of the locations of $\Upsilon_k$, and there is no infinite descending chains of location prefixes.

With this result, it is possible to introduce a rule that normalizes proof trees:

$$
\begin{align*}
P \models \varphi(T) & \quad P \models T = T_0 \quad Q \models T = T_1 \quad \hat{T}_0 = \hat{T}_1 \quad \text{norm} \\
Q \models \varphi(T) &
\end{align*}
$$

Of course, this does not means that all proof trees are finite (i.e. all properties are decidable!), but any proof tree with an arbitrary (even infinite) number of application of the rewrite rules can be replaced by \textit{[merge]}, \textit{[perco]} and \textit{[false]}-free proof trees with individual uses of the \textit{[norm]} rule. We think the situation is quite comparable to cut-elimination results in classical logics.

Most proof trees that use several instances of rewrite rules can be simplified using \textit{[norm]}. For instance, let us try to prove that for any property $\varphi(T)$, $P + P + [\text{false}]Q \models \varphi(T)$ can be deduced from $P \models \varphi(T)$ and $\bot \models \varphi(T)$. The second hypothesis is necessary because not every property is satisfied by $\bot$, termination being the obvious example.

$$
\begin{align*}
P \models \varphi(T) & \quad P + P \models \varphi(T) & \quad \bot \models \varphi(T) & \quad [\text{false}]Q \models \varphi(T) & \quad \sum \quad \text{weaken} \\
P + P + [\text{false}]Q \models \exists T_1, T_2, T = T_1 \oplus T_2 \land \varphi(T_1) \land \varphi(T_2) & \quad P + P + [\text{false}]Q \models \varphi(T)
\end{align*}
$$

The previous proof can also be done with one single \textit{[norm]}:

$$
\begin{align*}
P \models \varphi(T) & \quad P \models T = T_0 \quad P + P + [\text{false}]Q \models T = T_1 \quad T_1 \quad \text{weak} & \quad \text{strong} & \quad \text{merge} & \quad T_0 \\
T_0 = T_1 & \quad \text{norm}
\end{align*}
$$

4. Related work

There exist various logical frameworks developed to address the issue of reasoning about mobility. Dam proposed in [5] a non-trivial variant of the $\mu$-calculus that encompasses the specific features of the $\pi$-calculus. In the tradition of temporal logics, the approach adopts a purely operational point of view. A good aspect is that model checking algorithms can be developed - though the procedure proposed in [5] does not seem to have been experimented in practice. But working on the labeled transition systems directly prevents formal reasoning at a higher level of abstraction. In contrast,
the trace predicates that can be expressed in denotational logics are much more flexible and more general. They can deal with various aspects such as the counting of resources, geometrical properties about the branching structure, etc. The development of spatial logics [3] is also worth mentioning. The reasoning about hierarchical dynamic structures is not yet expressible in the logic we propose. We do think, though, that this could be reflected in a denotational way by enriching the location model - we could allow locations to nest within locations.

The study of the π-calculus semantics from a denotational point of view has been investigated in [2, 6] with the objective of characterizing full abstraction lemmas wrt. testing preorders. Most interpretations consider set-theoretic trace models built from the operational semantics. The idea is to relate trace-based denotations, so-called acceptance traces, with tests based on the operational semantics in order to provide full abstraction lemmas. In this paper, we adopt a complementary point of view of building trace models directly, with the goal of providing proof principles and techniques directly applicable at the denotational level, as in CSP.

The model we propose is closer to the CSP semantics [8, 13]. Our work can be seen as an extension of trace-equivalence in the context of the π-calculus with its specificities. We largely depart from CSP to analyze the branching structure of behaviors. Stable failures do not seem to be compatible with the notion of mobility. With name-passing, nothing much can be said about the recording of refusal sets. Making assertions about what a process receiving and sending names may refuse from the environment seems hardly realizable. The encoding of the branching structure within traces also makes the model better integrated, without having to consider multiple semantic domains (traces, stable failures, etc.). The idea of marking observations in traces with locations to capture the branching structure of process behaviors was initially inspired by the labeling concept of [4]. The authors used labels to identify concurrent subprocesses – two distinct branches in the same process share the same label – whereas we identify all non-deterministic branches in the behaviors. A global uniqueness property for labels is assumed, which appears to us as contradicting compositionality. In our model, the locations are unique by construction. The closest location model we can think of is that of term positions in rewrite systems [1]. While positions are absolute in term structures, we adopt a relative encoding that is more economic, but requires absolute locations to be reconstructed from the origin of “time” ϵ. Note that our model also permits the encoding of infinite tree structures.

5. Conclusion and future work

We presented in this paper a logical framework that adopts a denotational point of view to characterize the behavior of mobile systems. Despite the relative notational complexity of some of the operators employed (e.g. the escaping function), we do think the logic we propose to be comparable to the basic CSP logic (without the stable failures part) in terms of complexity. Our experiments with the logic shows that the proof normalization procedure plays an important role. It allows to develop fine-grained transformations that are very close to what can be obtained at the level of the operational semantics. The normalization procedure allows to abstract away from single rewrites, it thus fits perfectly the denotational nature of the framework.

Of course, the logic is still in its early stage of development and further investigation is required at the meta-theoretic level. In [12] we develop a hierarchy of equivalence relations for process behaviors. The most abstract one is the trace equivalence that indeed abstracts too much from the process semantics (in particular their branching structure). At the other end of the spectrum is the localized trace equivalence, which is the process equivalence implied by the trace set equality discussed in this paper. In between lies the split trace equivalence that allows the fine-grained rewrites explained in Table 3. We currently see whether different variants of the logic (different set of inference rules) can
be used to characterize these semantical equivalences in logical terms. For this, we must find precise restrictions on the trace predicates admissible in each one of these variants. The weak variant of the logic proposed in the paper is a simple way to abstract from internal actions (i.e. weak locators) in a compositional way. It is probably possible to go a little bit further and characterize an equivalent of the rooted weak bisimilarity in a denotational way by defining proper rewrite rules.

References