Communicating Process Architectures 2009
Peter Welch, Herman Roebbers and Tobe Announced (Eds.)
IOS Press, 2009

© 2009 The authors and IOS Press. All rights reserved.

A Denotational Study of Mobility

Joël-Alexis BIALKIEWICZ and Frédéric PESCHANSKI

UPMC Paris Universitas – LIP6,
104, avenue du Président Kennedy, 75016 Paris, France.

{Joel-Alexis.Bialkiewicz , Frederic.Peschanski} @lip6.fr

Abstract. This paper introduces a denotational model and refinement theory for a process algebra with mobile channels. Similarly to CSP, process behaviours are recorded as trace sets. To account for branching-time semantics, the traces are decorated by structured locations that are also used to encode the dynamics of channel mobility in a denotational way. We present an original notion of split-equivalence based on elementary trace transformations. It is first characterised coinductively using the notion of split-relation. Building on the principle of trace normalisation, a more denotational characterisation is also proposed. We then exhibit a preorder underlying this equivalence and motivate its use as a proper refinement operator. At the language level, we show refinement to be tightly related to a construct of delayed sums, a generalisation of non-deterministic choices.

Keywords. mobility, denotational semantics, refinement

Introduction

Mobile calculi such as the π-calculus [1] provide a suitable abstraction to model and reason about the dynamics of concurrent systems. In the spirit of CCS, they adopt in general a purely *operational* point of view; a syntax is elaborated, and then some operational semantics rules are figured out. Proof principles, generally based on bisimulation, are proposed above these. In our opinion there is little room in such an approach for high-level reasoning principles such as the ones available in the world of CSP: fixed-point characterisations, refinement, etc. On the other hand, the *denotational* point of view makes the constructs of the language simple syntactic sugars for *natural* operators found at the semantic level. The syntax is a derivative of the semantics and not the converse. Our objective is thus to elaborate solid foundations for mobile calculi from a denotational point of view. Unsurprisingly, we adopt the same basic construction as CSP: a model of trace semantics.

There are various difficulties in designing a trace model that encompasses the features and expressivity of mobile calculi such as the π-calculus. First, standard trace models do not take the branching structure of process behaviours into account. Instead of relying on stable failures, we use an alternative approach - introduced in [2] - of enriching trace sets with structured locations that record at the same time *when* and *where* actions are performed. The interest of this approach is that beyond the adequate and well-integrated characterisation of non-determinism, the location model also provides a solution for the mobile features of the π-calculus, in particular *name passing* and the important issue of *freshness*. The idea is to relate the events concerning names (e.g. the creation of fresh names or the extrusion of their scope) to the locations where these events are taking place.

The contributions of the paper are as follows. First, the proposed trace model underlies a family of equivalences, most notably a notion of *split-equivalence* that is satisfying in that it is both observational and compositional. Regarding the proof techniques, we introduce

original principles of *trace transformation* and *normalisation* that allow to equate process behaviours in an easily mechanisable way. The second contribution of the paper relates to *refinement* [3,4]. At the language level, we show that the refinement ordering is tightly related to a construct of *delayed sums*, a strict generalisation of the standard choice operators. As an illustration, we show that the refinement order underlies a complete lattice structure of least fixed points used as foundations for the characterisation of recursive behaviours.

The outline of the paper is as follows. In Section 1, we describe the main characteristics of the proposed denotational model. The construction of the trace semantics of process behaviours is discussed in Section 2. In Section 3 we present a language with its syntactic constructs built above the trace semantics. Then, in Section 4, we present and discuss the family of behavioural equivalences we use to distinguish trace sets in complementary ways. We insist on the original notion of split-equivalence that accounts for branching-time behaviours. The refinement order and its complete lattice structure are discussed in Section 5. This is followed by a panorama of related work, the conclusion and bibliographical references. In the paper we omit a few auxiliary definitions and proof details, as well as the complete axiomatisation of split-equivalence. These can be found in a companion technical report [5].

1. The Trace Model

1.1. Observations and Locations

The goal of a denotational model for a process algebra is to characterise precisely the external, or *observational*, part of process behaviours, abstracting from the details of their internal computations. Our characterisation is based on trace models, largely inspired by the CSP semantics. A trace is a sequence of *observations* – or observable actions – recorded from a process behaviour.

Definition 1. *An **action** is either an output* $c!d$ *of subject (channel)* c *and object* d*, an input* $c?$ *of subject* c *or a termination* ✓*. The subject of an action* α *is denoted* $subj(\alpha)$ *and its object* $obj(\alpha)$*.*

Note that the input action does not involve any bound variable. The binder is in fact implicitly defined by the location when and where the input is observed. Another important remark is that unlike CSP, the characterisation of mobility requires to consider channels as first-class citizens[1]. The data passed along channels are *names*, which on occasion may identify other channels. We distinguish the plain names e.g. $a, b, \ldots$ (which are known in the global scope), received names $\rho_{\bar{l}}$ (received at location l) or *escaping name* $\nu_{\bar{l}}$ (escaped at location l). Note that in order to give proper semantics to name equality and extract adequate laws (cf. Section 4.3), the occurrences of names within traces are in fact equivalence classes of names. For convenience, the singleton set $\{n\}$ with n a name is simply denoted n.

Figure 1 depicts two processes that are only distinguishable by their branching-time behaviour. In the left process there is a non-deterministic choice between two possible continuations starting with an α action. In the right process the action α is first performed and then an external choice is performed for either the action β on the left or γ on the right. In standard trace semantics these two behaviours are not distinguished, and it is one of the main argument to rely on bisimulations to compare process behaviours. Below these two examples, we give the characterisation of these behaviours in the proposed framework. Ignoring for now the details of this encoding, the bottom part of the picture provides an operational

[1]The integration of CSP-like *events* in the model is not very difficult. The idea is to consider events as observations and modify the means of synchronisations between events. But events cannot be used for channel mobility so we omit them in the paper.

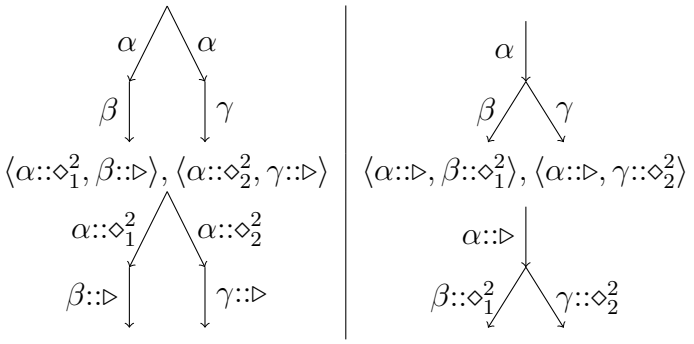


Figure 1. Examples of branching behaviours

interpretation. We can see that the two behaviours are distinguished in this interpretation. The important aspect is that the notion of observation is tightly related to the notion of location in the proposed model.

Definition 2. *An **observation** is the adjunct $\alpha::l$ of an action α and a location l.*

The characterisation of the branching structure is not the only problem we have to face. The semantics of mobile calculi such as the π-calculus and its variants introduce *history dependence* [6] — the semantics of a name depends on what happened before its considered occurrence. 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 $[x = y]$ (comparison between names x and y) then we must "remember" x was bound and also assume now 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 behaviour 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 behaviour can be observed. This *freshness* guarantee is difficult to model except in a symbolic way using *scope extrusion laws* [1]. 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. [2].

Definition 3. *Let i, j be integers such that $1 \leq i \leq j$. A **locator** is either a strong locator $\diamond_i^j$, a weak locator $\widetilde{\diamond}_i^j$ or the origin locator ϵ. A (relative) **location** l is defined by the following grammar where ϵ is the origin, s a strong locator, w a weak locator and φ a logical guard on channel names:*

$l ::= \epsilon \mid \tilde{\lambda}$
$\tilde{\lambda} ::= \lambda \mid (\varphi, w).\tilde{\lambda}$
$\lambda ::= (\varphi, s)$

As a convenience, we denote λ the location $(true, \lambda)$. Locations share many features with *term positions* in term algebras [8]. Since the location of a given observation is relative to its predecessors, the origin locator is necessary to be able to reconstruct absolute locations. We denote $\bar{l}$ the absolute location leading to the relative location l. For each atomic location (φ, λ) the formula φ corresponds to the guard "protecting" locator λ and thus the observation made there (i.e. the observation *really* occurs only when φ is $true$). Since guards protect

locators, it is necessary to be able to extract the combined guard of a location. This is the role of the $\mathfrak{grd}$ function.

Definition 4. *Let φ_n range over guards and l_n over locators:*
$\mathfrak{grd}((\phi_1, l_1) \dots (\phi_n, l_n)) = \phi_1 \wedge \dots \wedge \phi_n$
$\mathfrak{grd}(\epsilon) = \mathit{true}$

A split location (either strong or weak) expresses a branching, or a non-deterministic choice, in the behaviour. A branching $\diamond_i^j$, or $\widetilde{\diamond}_i^j$ (with $1 \leq i \leq j$) corresponds to the i-th branch within a choice among j distinct branches. The weak variant is used to describe non-deterministic choices due to internal actions[2]. The locator $\diamond_1^1$ (resp. $\widetilde{\diamond}_1^1$) describes the absence of a choice, which we call a strong (resp. weak) *next* locator, denoted $\triangleright$ (resp. $\widetilde{\triangleright}$) for the sake of readability.

1.2. Sequences and Trace sets

Definition 5. *A **sequence** is an ordered collection of properly decorated observations. The empty sequence is denoted $\langle\rangle$ and a non-empty sequence $\langle \alpha_1{::}l_1, \alpha_2{::}l_2, \dots \rangle$. The **absolute location** of observation α_n within sequence $\langle \alpha_1{::}l_1, \alpha_2{::}l_2, \dots, \alpha_n{::}l_n, \dots \rangle$ is the concatenation $l_1.l_2 \dots l_{n-1}.l_n$. It is equivalently denoted $\overline{l_n}$.*

For instance, in the sequence $\langle \alpha{::}\triangleright, \beta{::}\widetilde{\diamond}_3^8\triangleright, \gamma{::}\diamond_2^2 \rangle$, the absolute location of β is $\triangleright\widetilde{\diamond}_3^8\triangleright$

Notation 1. *We use a few standard notations for sequences. The* prefixing *of a sequence S by a decorated observation $\alpha{::}l$ is denoted $\alpha{::}l.S$ The* concatenation *of sequences S_1 and S_2 is denoted $S_1{}^\frown S_2$. Sequence S_1 is a* prefix *of sequence S_2, which is denoted $S_1 \leq S_2$, if and only if $\exists S',\ S_2 = S_1{}^\frown S'$.*

A few operators on sequence are introduced to deal with locations.

Definition 6. *The **pre-sequence** (resp. **post-sequence**) of a sequence S at a location l, denoted $S \uparrow l$ (resp. $S \downarrow l$) is defined inductively as follows:*

$$\begin{bmatrix} (\alpha{::}l.S) \uparrow l.L \stackrel{\mathit{def}}{=} \alpha{::}l.(S \uparrow L) \\ \alpha{::}l.S \uparrow l \stackrel{\mathit{def}}{=} \langle \alpha{::}l \rangle \\ S \uparrow L \stackrel{\mathit{def}}{=} \langle\rangle \ \mathit{otherwise} \end{bmatrix} \left(\mathit{resp.} \begin{bmatrix} \alpha{::}l.S \downarrow l.L \stackrel{\mathit{def}}{=} S \downarrow L \\ \alpha{::}l.S \downarrow l \stackrel{\mathit{def}}{=} S \\ \alpha{::}lm.S \downarrow l \stackrel{\mathit{def}}{=} \alpha{::}m.S \\ S \downarrow L \stackrel{\mathit{def}}{=} \langle\rangle \ \mathit{otherwise} \end{bmatrix} \right)$$

Informally, computing the pre-sequence of a sequence S consists in following the path l and extracting the prefix S' of S at this point. Conversely, the post-sequence extracts the suffix after that point. For instance, let us consider trace set $T = \{\langle \alpha{::}\triangleright, \beta{::}\diamond_1^2 \rangle, \langle \alpha{::}\triangleright, \gamma{::}\diamond_2^2 \rangle\}$. Here the pre-sequence of T after absolute location $\epsilon\triangleright$, denoted $T \uparrow \triangleright$, will be $\{\langle \alpha{::}\triangleright \rangle\}$ and the corresponding post-sequence $T \downarrow \triangleright$ will be $\{\langle \beta{::}\diamond_1^2 \rangle, \langle \gamma{::}\diamond_2^2 \rangle\}$. Note that the pre-trace set and the post-trace set do not partition a trace set in the general case: they instead isolate a given branching point in the behaviour.

Definition 7. *A **substitution** of x by y in sequence S, denoted $S\{y/x\}$, consists in the sequence S where all the occurrences of x are replaced by y. A **generic substitution** of any x by any y with respect to φ in sequence S, denoted $S\{y/x \mid \varphi\}$, where φ is a property on x and y, corresponds to applying the substitution to any pair of names satisfying φ.*

[2]The internal actions are not recorded in traces, but their effect on the branching structure has to be recorded, hence the introduction of weak split location.

Trace sets are built from sets of sequences with a few constraints. Unless otherwise stated, all sequence operators naturally extend to trace sets by simply applying to all sequences within them. We define below some useful operators on localised trace sets.

Definition 8. *The **relocation** of a trace set T from location l_1 to location l_2 at location l, denoted $T\{l_1 \leftarrow l_2\}_l$, corresponds to the set $\{S\{l_1 \leftarrow l_2\}_l \mid S \in T\}$ with:*

- $\alpha::kl_1m.S\{l_1 \leftarrow l_2\}_k \stackrel{def}{=} \begin{cases} \alpha::kl_2m.S\{V_{\overline{l_2n}}/V_{\overline{l_1n}} \mid n \in \mathcal{L},\ V \in \{\rho, \nu\}\} \\ \quad \textit{if } \exists \varphi, L,\ kl_2m = L(\varphi, \diamond_i^n) \\ \checkmark::kl_2m \textit{ otherwise} \end{cases}$
- $\alpha::l.S\{l_1 \leftarrow l_2\}_{ll'} \stackrel{def}{=} \alpha::l.(S\{l_1 \leftarrow l_2\}_{l'})$
- $S\{l_1 \leftarrow l_2\}_l \stackrel{def}{=} S$ *otherwise*

Relocation is a very important operator, a little bit technical but conceptually quite simple. The idea is to update a trace set so that one of its location is renamed. But such a local change has a non-local impact on the trace. First, the locations are related in a prefix ordering so all successors must be updated in consequence. Moreover, because trace sets are closed under prefixing, a potentially infinite number of sequences can be concerned by the relocation. The ρ and ν elements in the definition are related to the fresh names generated by the model, which are uniquely characterised by the absolute location where they were created and thus have also to be updated whenever it is relocated. Their exact role will be clarified later on.

As an illustration of the relocation process, we take again the previous example $T = \{\langle \alpha::\triangleright, \beta::\diamond_1^2\rangle,\ \langle \alpha::\triangleright, \gamma::\diamond_2^2\rangle\}$. The relocation $T\{\diamond_1^2 \leftarrow \diamond_1^3\}_\triangleright\{\diamond_2^2 \leftarrow \diamond_2^3\}_\triangleright$ yields $\{\langle \alpha::\triangleright, \beta::\diamond_1^3\rangle,\ \langle \alpha::\triangleright, \gamma::\diamond_2^3\rangle\}$

The relocated set of sequences we obtain is not well-formed because there is no third branch involved in the behaviour. However, this partial trace set can be used by higher-level operators (e.g. for choice or parallel compositions) to recombine correct trace sets.

Definition 9. $T\{(\varphi, l_1) \leftrightarrow (\psi, l_2)\}_l \stackrel{def}{=} T\{(\varphi, l_1) \leftarrow (\varphi, \bullet)\}_l\{(\psi, l_2) \leftarrow (\psi, l_1)\}_l\{(\varphi, \bullet) \leftarrow (\varphi, l_2)\}_l$

Not all sets of sequence are valid trace sets, it is thus important to characterise precisely the structure of the set $\mathcal{T}$ of all possible trace sets. Technically, this is a setoid characterised as follows:

Definition 10. *A trace set T is a set of sequence of the setoid $(\mathcal{T}, =)$ with the following properties:*

fin $\forall S \in T,\ S$ *is finite*
pref $\forall S \in T,\ \forall S' \leq S,\ S' \in T$
move $T\{(\varphi, \diamond_i^n) \leftrightarrow (\psi, \diamond_j^n)\}_l = T$

The axioms [fin] and [pref] are identical to their CSP counterpart. The axiom [move] allows arbitrary commutations of locators: the order among the particular branches of a given location is not significant[3]

For the sake of readability, the trace sets presented in the paper are abbreviated as plain, non-prefixed sets of sequences but we of course assume the trace set axioms unless stated otherwise.

[3]The removal of the axiom [move] from the model leads to a notion of prioritised trace semantics that could be worth studying.

2. Trace Semantics

The key aspect of CSP-like trace semantics is the possibility to construct arbitrarily complex process behaviours from a reduced set of elementary behaviours and composition operators.

Definition 11. *The **empty** behaviour is represented by the empty trace set* $\{\langle\rangle\}$ *which we sometimes denote* $\emptyset$ *for brevity.*

Definition 12. *The **termination** behaviour is the trace set* $\{\langle\checkmark::\epsilon, \langle\rangle\rangle\}$.

The main operator for sequential behaviour is the prefixing of a trace set T by an action α, which we denote $\alpha::l.T$ (because the action must be located somewhere). In the case of an output, every sequences in the trace set is prefixed by the action decorated by $(true, \triangleright)$. The locator $\triangleright$, which is synonymous to $\diamond_1^1$, corresponds to a step forward in time; and since observing that prefix is unconditional, its condition is $true$. We remind the reader that the subject and object of observations are name sets rather than names; however, whenever there is no possible confusion the brackets may be omitted for the sake of brevity.

Definition 13. $c!a.T \stackrel{def}{=} \{\{c\}!\{a\}::(true, \triangleright).S\{\epsilon \leftarrow \epsilon\triangleright\} \mid S \in T\}$

Note that the sequences following the output must be relocated after the initial $\triangleright$.

As hinted previously, a key aspect of our encoding is the absence of any form of symbolism, in particular binders, as that would break the denotational nature of the model. For instance, there is no variable or binder attached to input prefixes: whatever is present at the current branching point will be received. If the data is received by an input occurring at absolute location L, it will be known thereafter as ρ_L. That absolute location will be built with the trace set by the use of the relocation operator.

Definition 14. $c?x.T \stackrel{def}{=} \{\{c\}?::(true, \triangleright).S\{\epsilon \leftarrow \epsilon\triangleright\}\{\rho_{\epsilon\triangleright}/x\} \mid S \in T\}$

A silent or internal action τ can obviously not be observed, this is the main idea underlying this notion. However, since it may have an effect on the branching structure, it is recorded as a weak location $\widetilde{\triangleright}$ attached to the initial action in the continuation.

Definition 15. $\tau.T \stackrel{def}{=} \{\alpha::(true, \widetilde{\triangleright})l.S\{\epsilon \leftarrow \epsilon\widetilde{\triangleright}\} \mid \alpha::l.S \in T\}$

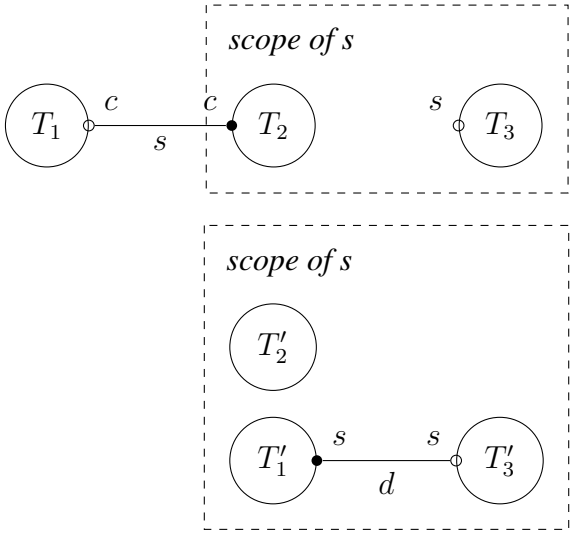
Note that an observation may have any number of *weak* locations, but it has at most one *strong* location that corresponds to the point where the actual observation occurs.

Guarding a trace set by a condition is simply done by guarding the head location of all the initials of its individual sequences, so that their initials may only be observed it that condition is true. However, to take into account the match conditions, e.g. $[a = b]$, we allow equivalence classes of names to be formed, e.g;. to replace both a and b by $\{a, b\}$ in the remainder of the trace set conditioned by the match[4].

Definition 16. $[G]T \stackrel{def}{=} \{\alpha::(G \wedge \varphi, l_0)l.S \mid \alpha::(\varphi, l_0)l.S \in T\{x \cup y/x \mid G \implies x = y\}\}$

Restriction corresponds to declaring a name as private, and thus not allowing to communicate using it as a subject. However, if it is used as the object of an output, it will *escape* its scope and become visible to the outside world. This is the core of mobility, and also what in our opinion is the most involved aspect of the model. The restriction of a name n is recorded in a trace set as an "escape". The effects of restriction is to cut short the sequences from the point where it is not possible to interact using the restricted name anymore (starting

[4]The idea of implementing match condition by equivalence classes of names is developed in [9].



$$T \stackrel{\text{def}}{=} sync(T_1, (\nu s)sync(T_2, T_3))$$
$$T_1 = \{\langle c?::\triangleright, \rho_{\epsilon\triangleright}!d::\triangleright, \checkmark::\epsilon\rangle\}\ \ T_2 = \{\langle c!s::\triangleright, \checkmark::\epsilon\rangle\}\ \ T_3 = \{\langle s?::\triangleright, \checkmark::\epsilon\rangle\}$$
$$T_1' = \{\langle \rho_{\epsilon\triangleright}!d::\triangleright, \checkmark::\epsilon\rangle\}\ \ T_2' = \{\langle \checkmark::\epsilon\rangle\}\ \ T_3' = \{\langle s?::\triangleright, \checkmark::\epsilon\rangle\}$$

Figure 2. Mobile behaviour illustrated

from any action whose subject is an occurrence of the restricted name n that is *not* escaped) and by finding the actions where the restricted name *indeed* escapes. The formal definitions, relatively technical, are given below.

Definition 17.

$$\mathfrak{E}_n(T) \stackrel{def}{=} \{\mathfrak{E}_n(S) \mid S \in T\}$$
$$\mathfrak{E}_n(\langle\rangle) \stackrel{def}{=} \langle\rangle$$
$$\mathfrak{E}_n(\alpha::l.S) \stackrel{def}{=} \begin{cases} \alpha::l\{false/(n=x)\}.S \textit{ if } \mathfrak{grd}(l) \implies n = x \\ \alpha::(false \wedge \varphi, \lambda)L \textit{ if } subj(\alpha) = n \textit{ and } l = (\varphi, \lambda)L \\ \mathfrak{F}_n^{\bar{l}}(\alpha::l.S)\{\nu_{\bar{l}}/n\} \textit{ if } \alpha = c!n \textit{ and } c \neq n \\ \alpha::l.\mathfrak{E}_n(S) \textit{ otherwise} \end{cases}$$
$$\mathfrak{F}_n^L(\langle\rangle) \stackrel{def}{=} \langle\rangle$$
$$\mathfrak{F}_n^L(\alpha::l.S) \stackrel{def}{=} \begin{cases} \alpha::l\{false/(n=x)\}.S \textit{ if } \mathfrak{grd}(l) \implies n = x \textit{ and not } x = \rho_m, L \leq m \\ \textit{or else } \alpha::l'.\mathfrak{F}_n^L(S) \textit{ with} \\ \quad l' = \begin{cases} l\{true/(n \neq x)\} \textit{ if not } x = \rho_m, L \leq m \\ l \textit{ otherwise} \end{cases} \end{cases}$$

Each escape is effected by replacing all the free occurrences of n by a name $\nu_{\bar{l}}$ generated from this point on. Since absolute locations are unique by construction, this name is guaranteed *fresh*.

Figure 2 illustrates how the scope of a restricted channel may evolve in time. At the beginning of the execution, only T_2 and T_3 are in the scope of s. However, T_2 sends s along public channel c, which is illustrated in the second step. This allows s to escape its scope, which now also integrates T_1. Since the name was received at absolute location $\epsilon\triangleright$, this name is called $\rho_{\epsilon\triangleright}$ in the continuation of the trace set. The third step is the communication of d along channel s which is now known by T_1 (as $\rho_{\epsilon\triangleright}$).

The choice between behaviours T_1 and T_2 is denoted $T_1 \oplus T_2$. It corresponds to a disjoint union between both behaviours. All locators are re-numbered in order to create a new, compound trace set.

Table 1. Syntax of the language

$P, Q, \ldots$	::=	VOID | END | $\alpha.P$ | $P + Q$ | $P \mid\mid\mid Q$ | $P \cdot Q$ | $P \parallel Q$ | $(\nu n)P$ | $\mu X.Q$ | X
$\alpha, \ldots$	::=	τ | $c!a$ | $c?x$ | $[\varphi]\alpha$
$\varphi, \psi, \ldots$	::=	$a = b$ | $a \neq b$ | $\varphi \wedge \psi$ | $\varphi \vee \psi$ | $\neg\varphi$

Definition 18. $T_1 \oplus T_2 \stackrel{def}{=} T_1\{(\varphi, s_i^n) \leftarrow (\varphi, s_i^{n+m})\}_\epsilon \cup T_2\{(\psi, s_j^m) \leftarrow (\psi, s_{j+n}^{n+m})\}_\epsilon$

The interleaving of T_1 and T_2 is denoted $ileave(T_1, T_2)$. It corresponds to interleaving the sequences of both behaviours.

Definition 19.

$$ileave(T_1, T_2) \stackrel{def}{=} \bigoplus_{\substack{S_1 \in T_1 \\ S_2 \in T_2}} ileave(S_1, S_2)$$

$$ileave(\alpha_1{::}l_1.S_1, \alpha_2{::}l_2.S_2) \stackrel{def}{=} \alpha_1{::}l_1.ileave(S_1, \alpha_2{::}l_2.S_2) \oplus \alpha_2{::}l_2.ileave(\alpha_1{::}l_1.S_1, S_2)$$

The pure synchronisation between T_1 and T_2 is denoted $sync(T_1, T_2)$. It corresponds to allowing all possible communications between the two behaviours to occur without any interaction with the environment, much like the CSP parallel operator does.

Definition 20.

$$sync(T_1, T_2) \stackrel{def}{=} \bigoplus_{\substack{S_1 \in T_1 \\ S_2 \in T_2}} sync(S_1, S_2)$$

$$sync(a!d{::}l_1.S_1, b?{::}l_2.S_2) = sync(b?{::}l_2.S_2, a!d{::}l_1.S_1) \stackrel{def}{=} sync(S_1, S_2\{d/\rho_{\overline{l_2}}\})\{\epsilon \leftarrow (a = b, \widetilde{\triangleright})\}$$

Since internal synchronisations can not be observed, the trace set of pure communications between processes, if it is computable, can only contain sequences where the only observation is the termination ✓ decorated by a chain of conditioned weak locations.

Both notions of parallelism can be combined into an universal parallel operator $\oplus\!\!\!\!\cdot$ which allows to account for both interleaving and communication between behaviours. It is defined using mutually recursive modifications of $ileave$ and $sync$, which are denoted $interleave$ and $intersync$.

Definition 21.

$$T_1 \oplus\!\!\!\!\cdot\ T_2 \stackrel{def}{=} interleave(T_1, T_2) \oplus intersync(T_1, T_2)$$

$$interleave(T_1, T_2) \stackrel{def}{=} \bigoplus_{\substack{S_1 \in T_1 \\ S_2 \in T_2}} interleave(S_1, S_2)$$

$$intersync(T_1, T_2) \stackrel{def}{=} \bigoplus_{\substack{S_1 \in T_1 \\ S_2 \in T_2}} intersync(S_1, S_2)$$

$$interleave(\alpha_1{::}l_1.S_1, \alpha_2{::}l_2.S_2) \stackrel{def}{=} \alpha_1{::}l_1.(S_1 \oplus\!\!\!\!\cdot\ \alpha_2{::}l_2.S_2) \oplus \alpha_2{::}l_2.(\alpha_1{::}l_1.S_1 \oplus\!\!\!\!\cdot\ S_2)$$

$$intersync(a!d{::}l_1.S_1, b?{::}l_2.S_2) = intersync(b?{::}l_2.S_2, a!d{::}l_1.S_1) \stackrel{def}{=} (S_1 \oplus\!\!\!\!\cdot\ S_2\{d/\rho_{\overline{l_2}}\})\{\epsilon \leftarrow (a = b, \widetilde{\triangleright})\}$$

3. The Language

As explained in the introduction, we think that an important characteristic of CSP is that the syntax of the language follows the denotation and not the converse. The syntactic constructs that, in our opinion, *naturally* emerge from the denotation proposed in the previous sections is summarised in Table 1.

With first-class channels, a dynamic restriction operator and generalised choice operator, the language is at the surface closer to the π-calculus than to CSP. But since we share with CSP the same philosophy (i.e. "denotation speaks") and also many semantic concepts, we

Table 2. Semantics of the language

$[\![\text{VOID}]\!]$	$\stackrel{\text{def}}{=} \emptyset$	$[\![\text{END}]\!]$	$\stackrel{\text{def}}{=} \{\langle\checkmark::\epsilon\rangle\}$
$[\![\alpha.P]\!]$	$\stackrel{\text{def}}{=} \alpha.[\![P]\!]$	$[\![[G]P]\!]$	$\stackrel{\text{def}}{=} [G][\![P]\!]$
$[\![(\nu n)P]\!]$	$\stackrel{\text{def}}{=} \mathfrak{E}_n([\![P]\!])$	$[\![P+Q]\!]$	$\stackrel{\text{def}}{=} [\![P]\!] \oplus [\![Q]\!]$
$[\![P \mid\!\mid\!\mid Q]\!]$	$\stackrel{\text{def}}{=} \mathit{ileave}([\![P]\!],[\![Q]\!])$	$[\![P \cdot Q]\!]$	$\stackrel{\text{def}}{=} \mathit{sync}([\![P]\!],[\![Q]\!])$
$[\![P \parallel Q]\!]$	$\stackrel{\text{def}}{=} [\![P]\!] \circledcirc [\![Q]\!]$	$[\![\mu X.P]\!]$	$\stackrel{\text{def}}{=} [\![P\{\tau.\mu X.P/X\}]\!]$

Table 3. Examples illustrating trace set construction

1. $[\![\tau.a!b.END + c!d.END]\!] = \{\langle a!b::\widetilde{\diamond}^2_1\triangleright, \checkmark::\epsilon\rangle, \langle c!d::\diamond^2_2, \checkmark::\epsilon\rangle\}$
2. $[\![a?x.b?y.[x=y]c!x.END]\!] = \{\langle a?::\triangleright, b?::\triangleright, c!\{\rho_\triangleright, \rho_{\triangleright\triangleright}\}::(\rho_\triangleright = \rho_{\triangleright\triangleright}, \triangleright), \checkmark::\epsilon\rangle\}$
3. $[\![a!b.END \parallel c?x.x!d.END]\!] = \{\langle a!b::\diamond^4_1, c?::\triangleright, \rho_{\diamond^4_1\triangleright}!d::\triangleright, \checkmark::\epsilon\rangle, \langle c?::\diamond^4_2, a!b::\triangleright, \rho_{\diamond^4_2}!d::\triangleright, \checkmark::\epsilon\rangle,$
$\langle c?::\diamond^4_3, \rho_{\diamond^4_3}!d::\triangleright, a!b::\triangleright, \checkmark::\epsilon\rangle, \langle b!d::(a=c, \widetilde{\diamond}^4_4)\triangleright, \checkmark::\epsilon\rangle\}$
4. $[\![(\nu a)a!b.END]\!] = \{\langle\checkmark::\epsilon\rangle\}$ 5. $[\![(\nu a)(a!b.END \parallel z?x.x!c.END)]\!] = \{\langle b!c::(a=z, \widetilde{\triangleright})\triangleright, \checkmark::\epsilon\rangle\}$
6. $[\![(\nu a)c!a.a?x.x!m.END]\!] = \{\langle c!\nu_\triangleright::\triangleright, \nu_\triangleright?::\triangleright, \rho_{\triangleright\triangleright}!m::\triangleright, \checkmark::\epsilon\rangle\}$
7. $[\![\mu P.(\nu n)a!n.P]\!] = \{\langle a!\nu_\triangleright::\triangleright\rangle, \langle a!\nu_\triangleright::\triangleright, a!\nu_{\triangleright\widetilde{\triangleright}\triangleright}::\widetilde{\triangleright}\triangleright\rangle, \langle a!\nu_\triangleright::\triangleright, a!\nu_{\triangleright\widetilde{\triangleright}\triangleright}::\widetilde{\triangleright}\triangleright, a!\nu_{\triangleright\widetilde{\triangleright}\triangleright\widetilde{\triangleright}\triangleright}::\widetilde{\triangleright}\triangleright\rangle, \ldots\}$

rather see this language as a hybrid. To reflect this we adopted when possible the syntactic style of CSP. Note that there is no natural equivalent of the choice operators $\Box$ and $\sqcap$ of CSP in our denotation, because these relate to stable failures while we use locations instead. The generalised choice $+$ is in fact neither deterministic nor non-deterministic. It is deterministic whenever possible, and non-deterministic otherwise. A purely internal choice can be encoded by weak locations (inserted by explicit τ prefixes in the syntax). Mainly because it has simpler denotation and axiomatisation, we also prefer explicit guarding of processes than the if-then-else construct. But it is possible to encode $P \lessgtr \varphi \gtrless Q$ as $[\varphi]P + [\neg\varphi]Q$.

We can now connect the syntax to the semantics.

Definition 22. *The trace set of a process P is $[\![P]\!]$ calculated according to Table 2.*

The semantic encoding of Table 2 illustrates, in our opinion quite demonstratively, the fact that the proposed syntactic constructs *naturally* emerge from the semantics. Each construct has a dedicated operators applying at the semantic level. The encoding of recursive process is, as a first approximation, encoded as a simple unfolding. A subtlety is that we introduce a silent action "before" each unfolding. This has the advantages of taking into account the computational cost of unfolding, and it makes divergences to be observable (as unbounded sequences of weak locations). A more denotational characterisation of recursion as fixed points requires a proper refinement model. This is proposed in Section 5.

To illustrate the calculation of trace sets, we provide a few examples in Table 3. The first example illustrates the sum recorded as a combination of branches built using the operator $\oplus$. Note that the combined branches are correctly renumbered. The second example illustrates the treatment of "binders", i.e. received names recorded together with the absolute location of their reception. The third one is about parallel composition, which composes the possible interleavings and communications of the operand processes. Interleavings are computed by the function $\mathit{interleave}$, communications using the function $\mathit{intersync}$, and those behaviours are joined together by $\oplus$ like a process sum. The next three examples are about restriction, and illustrate that restriction behaves as expected, which can be checked easily by following restriction/escaping function $\mathfrak{E}_n$. Example 4 gives a situation where there is no possible interaction. In Example 5, the process very much behaves like $[a=z]\tau.b!c$ since the only possible interaction is an internal communication. Example 6 is an example of the escape of a private name sent over a public channel. The last example illustrates unfolding recursion on a very simple case.

4. Split-equivalence

The proposed denotational semantics is not as simple or "beautiful" as we would like it to be. Locations represent at the same time its strength and weakness from this point of view. On the positive side, they offer a well integrated encoding of the branching structure of process behaviours, and perhaps most importantly when it comes to mobility, an adequate (i.e. compositional) characterisation of freshness. But in return, they are also quite fine-grained and, not unlike de Bruijn indices in the λ-calculus, uneasy to deal with in the formal definitions[5]. It is thus very important to develop *proof principles* and *techniques* allowing to abstract away from the technical details. The basic step towards that objective is the development of a proper notion of semantic equivalence. The trace model presented in the previous sections naturally underlies an equivalence relation based on the setoid *identity* of Definition 10. This so-called *localised trace equivalence* is denoted $P =_{\mathcal{L}} Q$ and holds if and only if $[\![P]\!] = [\![Q]\!]$. It is not, however, a satisfying equivalence in all situations. For instance, it does not preserve such a basic property as $P + P = P$ (because there is a supplementary split location in the left hand side of the equality).

A first obvious nevertheless useful way to loosen the comparison is to simply forget about locations altogether. This results in the notorious *trace equivalence*, denoted $=_{\mathcal{T}}$, that does not take into account the branching structure of processes. Trace equivalence is enough to address safety issues, but is too imprecise as a general equivalence forgetting about non-deterministic choices, e.g. it equates processes such as $\alpha.(P + Q)$ and $\alpha.P + \alpha.Q$

The trace-based equivalence developed in this section, *split-equivalence*, can be seen as an intermediate between the localised and plain forms of trace equivalences. On the one side it preserves the observational contents as captured by $=_{\mathcal{T}}$ and on the other side it weakens the constraints imposed by $=_{\mathcal{L}}$ on locations.

4.1. Trace transformations

The general idea is to start from the localised equivalence $=_{\mathcal{L}}$ but allow a certain number of transformations that preserve the branching structure and observational semantics.

Table 4. The transformations on trace sets

[merge]	$\forall s, s' \in \{\diamond, \widetilde{\diamond}\},\ T\{(\varphi, s_a^n) \leftarrow (\varphi \vee \psi, s_b^n)\}_l\{(\xi_i, s'^n_i)l_i \leftarrow \xi_i, s'^{n-1}_{i-1})l_i \forall i > a\}_L$ $\{(\xi_i, s'^n_i)l_i \leftarrow \xi_i, s'^{n-1}_i)l_i \forall i < a\}_l$ if $T \downarrow l(\varphi, s_a^n) = T \downarrow l(\psi, s_b^n)$
[perco]	$T\{(\varphi \wedge \psi, l_1) \leftarrow (\psi, l_1)\}_l$ if $\mathfrak{grd}(\bar{l}) \implies \varphi$
[weak false]	$T\{l_f \leftarrow (\psi, s'_{k,n})l_e\}_l$ where $\mathfrak{grd}(\overline{l_f}) = false,\ hd(l_f) = (\varphi, s_k^n)$ and $\mathfrak{f}(\overline{l_f}) = (\psi, s'_{i,j})l_e$
[strong false]	$\forall i > k,\ T \setminus (T \downarrow l.l_f)\{(\varphi, s_i^n) \leftarrow (\varphi, s_i^{n-1}) \forall i < k\}_l\{(\varphi, s_i^n) \leftarrow (\varphi, s_{i-1}^{n-1}) \forall i > k\}_l,$ $s \in \{\diamond, \widetilde{\diamond}\}$ where $l_f = (\psi, s'^n_k)l'_f,\ \alpha :: l_f.S \in T \downarrow l,$ $\mathfrak{grd}(\overline{l_f}) = false$ and $\mathfrak{f}(\overline{l_f}) = \emptyset$

Definition 23.

$$\mathfrak{f}(\emptyset) \stackrel{def}{=} \emptyset$$

$$\mathfrak{f}((\varphi, l).L) \stackrel{def}{=} \begin{cases} \emptyset \textit{ if } \varphi \iff false \\ l.\mathfrak{f}(L) \textit{ otherwise} \end{cases}$$

These transformations are described in Table 4. Note that they rely heavily on trace relocation. Despite their somewhat technical definitions, the transformation are conceptually simple:

[5] Despite the apparent complexity of the proposed formulations on *paper*, most of the proposed definition are designed as simple recursive functions easily implementable.

Merge The transformation merges two distinct branches (identified by two different s_i^n and s_j^n strong or weak split locators) at a given location l whenever they exhibit the same behaviour (wrt. $=_{\mathcal{L}}$). 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 may occur. 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 $\bar{l}$ 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 behaviour. 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, *weak false* transforms $\tau.[false]P+Q$ into $\tau.\text{END}+Q$. Note that in this case END is used to record the possibility of branching into a deadlock, which clearly differs from the usual CSP semantics.

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, *strong false* transforms $[false]P + Q$ into Q.

Definition 24. *A **split-relation** $\mathcal{R}$ is a symmetric binary relation on trace sets such that $T_1 \mathcal{R} T_2$ if and only if $T_1 = T_2$ or there exists a couple of transformations $\mathcal{U}, \mathcal{V}$ such that $\mathcal{U}(T_1)\mathcal{R}\mathcal{V}(T_2)$. The **split-equivalence** is $=_\diamond \stackrel{def}{=} \{\mathcal{R} \mid \mathcal{R}$ is a split-relation$\}$*

We extend the notation to process expressions, considering two processes P and Q as split-equivalent, denoted $P =_\diamond Q$, if and only if $[\![P]\!] =_\diamond [\![Q]\!]$. The rationale is that two processes are equivalent if and only if there trace sets are either identical (according to $=_{\mathcal{L}}$, which means the axioms of Definition 10 hold) or they can be transformed (using the transformations of Table 4) an arbitrary number of times so as to be made identical (still according to $=_{\mathcal{L}}$). The equivalence itself is defined in coinductive terms, which means it encompass infinite behaviours (by allowing to apply an infinite times the transformations).

4.2. Normalisation techniques

The definition of split-equivalence is concise and conceptually simple. Unfortunately, the coinduction proof technique is relatively cumbersome to deal with in practice. It only works well for either very general and simple relations (e.g. showing that $=_{\mathcal{L}}$ is a split-relation and thus included in split-equivalence), or on very simple behaviours (in this case exhibiting a split relation is easy). In general this is not a very practical proof technique. The first reason is that it tells nothing about what transformation to choose for a given context. Also, it gives an operational feel to the semantics because one has to consider each transformation individually. Moreover the straightforward approach does not terminate even for some finite systems, potentially requiring an infinite number of transformations to be applied (e.g. to relate $\mu X.(\tau.X + P)$ and $\mu Y.(P + \tau.Y)$). Indeed, the transformation rules could be added as axioms for the trace setoid. There is however a reason why we maintain these rules outside the setoid: *trace normalisation*. The technique we now discuss is based on the idea of rewrite systems [8].

Definition 25. *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 denote $T \overset{\mathcal{U}::l}{\not\longrightarrow}$. An arbitrary rewrite (at an arbitrary location) is denoted $T \to T'$. If a trace T is such that $T \not\to$ then T is said in **normal form**, which is denoted $\widehat{T}$*

The rewrite rules give a directed and localised interpretation of the transformations of Table 4. The definition of a normal forms is also important it that it provides an alternative characterisation of split-equivalence.

Proposition 1. *$P =_\diamond Q$ iff $\widehat{[\![P]\!]} = \widehat{[\![Q]\!]}$*

This follows naturally from the fact that the normalisation itself is a split-relation. Now, in order to prove that two processes P and Q are split-equivalent, we can compute the normal forms of their trace sets and equate the later using $=_{\mathcal{L}}$. A useful lemma shows that normal forms are unique up-to $=_{\mathcal{L}}$.

Lemma 1. *Let T be a trace set. Suppose T_1 and T_2 such that $T \to^* T_1 \not\to$ and $T \to^* T_2 \not\to$. Then $T_1 = T_2 = \widehat{T}$.*

The proof for this lemma requires a *diamond property*, relatively technical, which is detailed in the technical report [5].

For the moment, we do not gain much by using the normalisation technique to prove split equivalence. It is possible, however, to take advantage of the finitely branching structure of behaviours as well as the well-foundedness of the prefix ordering on locations to uncover a *weak termination* property of the normalisation 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 set. 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 26. *A single parallel simplification of a trace set T is a triple (T, Υ, T') with Υ the set of all the independent rewrites applicable on T. The triple is denoted $T \overset{\Upsilon}{\Longrightarrow} T'$.*

By Lemma 1, we know that the order of application of the individual rewrites $\mathcal{U}::l \in \Upsilon$ is not significant so if we consider such parallel application as atomic, the relation $\Rightarrow$ enjoys a decisive weak termination lemma.

Lemma 2. *The parallel simplification of a trace set T is terminating, i.e the descending chain $T \overset{\Upsilon_1}{\Longrightarrow} T' \overset{\Upsilon_2}{\Longrightarrow} T'' \ldots \overset{\Upsilon_n}{\Longrightarrow} \widehat{T}$ is terminated (i.e. n is finite)*

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

4.3. Laws

Equipped with adequate proof principles, we can now discuss a certain number of laws about the language constructors, interpreted in term of trace properties. The technical report [5] contains more properties of the model, with more thorough proof details. Most notably, it provides a complete axiomatisation of split-equivalence.

We first discuss the *compositional* nature of split-equivalence (i.e. it is a congruence for all language constructors).

Lemma 3. *Let $\alpha::l$ a location, T and T' a couple of trace sets, then $T = T' \implies \alpha::l.T = \alpha::l.T'$*

Proof. This is simple: $\alpha::l.T = \{\alpha::l.S \mid S \in T\}$, $\alpha::l.T' = \{\alpha::l.S' \mid S' \in T'\}$ and $T = T'$ so that $\forall S \in T, \exists S' \in T'$ with $S = S'$ and $\forall S' \in T', \exists S \in T$ with $S = S'$, which trivially gives $\forall \alpha::l.S \in \alpha::l.T, \exists \alpha::l.S' \in \alpha::l.T'$ with $\alpha::l.S = \alpha::l.S'$ and $\forall \alpha::l.S' \in \alpha::l.T', \exists \alpha::l.S \in \alpha::l.T$ with $\alpha::l.S = \alpha::l.S'$ □

Lemma 4. *Let σ an injective substitution of names by other names, T and T' trace sets, then $T = T' \implies T\sigma = T'\sigma$. The same is true for σ an injective substitution of locations by other locations, provided the cosupport of σ only contains fresh split or weak split locations.*

Proof. This is trivial by considering $T\sigma = \{S\sigma \mid S \in T\}$ and $T'\sigma = \{S'\sigma \mid S' \in T'\}$, using a similar proof scheme to that of Lemma 3, we can match all $S\sigma$'s in T to $S'\sigma$'s in T' and vice versa, which is enough to conclude □

Lemma 5. *For any single-hole context C:*

$$[\![P]\!] =_\diamond [\![Q]\!] \implies [\![C[P]]\!] = [\![C[Q]]\!]$$

Proof. Let C be a single-hole context.We proceed by case analysis on C. The common hypothesis in all cases is that $P =_\diamond Q$, i.e., there exists Υ_1 and Υ_2 a couple of simplifications such that $\Upsilon_1([\![P]\!]) = \Upsilon_2([\![Q]\!])$. In all cases, we can separate the issue in first exhibiting a couple of simplifications Υ'_1 and Υ'_2 to recover a comparison up-to $=_\mathcal{L}$, and then secondly to discuss the observable properties of the modified contexts. In many cases, it is enough to delay Υ_1 and Υ_2 to obtain the simplifications we may apply on the contexts.

- Case $C \stackrel{\text{def}}{=} \tau.[.]$: we take $\Upsilon'_1 \stackrel{\text{def}}{=} \Upsilon_1::(true, \widetilde{\triangleright})$ and $\Upsilon'_2 \stackrel{\text{def}}{=} \Upsilon_2::(true, \widetilde{\triangleright})$. We have $[\![\tau.P]\!] \stackrel{\text{def}}{=} [\![P]\!]\{\epsilon \leftarrow (true, \widetilde{\triangleright})\}$ (see Table 2) and also $[\![\tau.Q]\!] \stackrel{\text{def}}{=} [\![Q]\!]\{\epsilon \leftarrow (true, \widetilde{\triangleright})\}$. Moreover, by Definition 25, we obtain $\Upsilon_1::(true, \widetilde{\triangleright})([\![P]\!]\{\epsilon \leftarrow (true, \widetilde{\triangleright})\}) = \Upsilon_1([\![P]\!]\{\epsilon \leftarrow (true, \widetilde{\triangleright})\})$ and $\Upsilon_2::(true, \widetilde{\triangleright})([\![Q]\!]\{\epsilon \leftarrow (true, \widetilde{\triangleright})\}) = \Upsilon_2([\![P]\!]\{\epsilon \leftarrow (true, \widetilde{\triangleright})\})$ and we conclude the case by Lemma 3.
- Case $C \stackrel{\text{def}}{=} \alpha.[.]$: we take $\Upsilon'_1 \stackrel{\text{def}}{=} \Upsilon_1::(true, \triangleright)$ and $\Upsilon'_2 \stackrel{\text{def}}{=} \Upsilon_2::(true, \triangleright)$. We have $[\![\alpha.P]\!] \stackrel{\text{def}}{=} \alpha::(true, \triangleright).[\![P]\!]$ (see Table 2) and also $[\![\alpha.Q]\!] \stackrel{\text{def}}{=} \alpha::(true, \triangleright).[\![Q]\!]$. Moreover, by Definition 25, we obtain $\Upsilon_1::(true, \triangleright)(\alpha::(true, \triangleright).[\![P]\!]) = \alpha::(true, \triangleright).\Upsilon_1([\![P]\!])$ and $\Upsilon_2::(true, \triangleright)(\alpha::(true, \triangleright).[\![Q]\!]) = \alpha::(true, \triangleright).\Upsilon_2([\![P]\!])$ and we conclude the case by Lemma 3.
- Case $C \stackrel{\text{def}}{=} [\varphi][.]$ where φ is a guard: we take $\Upsilon'_1 \stackrel{\text{def}}{=} \Upsilon_1$ and $\Upsilon'_2 \stackrel{\text{def}}{=} \Upsilon_2$. The modification of the observational contents is making the head locations of both $[\![P]\!]$ and $[\![Q]\!]$ guarded by φ, and if $\alpha::(\psi, l) = \beta::(\xi, l)$ then of course $\alpha::(\varphi \wedge \psi, l) = \beta::(\varphi \wedge \xi, l)$.
- Case $C \stackrel{\text{def}}{=} (\nu n)[.]$: we also take $\Upsilon'_1 \stackrel{\text{def}}{=} \Upsilon_1$ and $\Upsilon'_2 \stackrel{\text{def}}{=} \Upsilon_2$. The function $\mathfrak{E}_n$ will be applied on each sequence of $[\![P]\!]$ and $[\![Q]\!]$. It is obvious that $\mathfrak{E}_n(S) = \mathfrak{E}_n(S)$ but here the trace sets are not strictly equal; however, their normalisations are. We now have to prove that for each possible individual simplification θ, $\{\mathfrak{E}_n(S) \mid S \in T\} = \{\mathfrak{E}_n(S') \mid S' \in \theta(T)\}$ and the case will be proved by transitivity on simplifications. We will only consider the sequences possibly modified by each kind of rewrite rule. If θ is a *compression*, the conclusion is obvious by Lemma 4 on both locations and names. If θ is a *merge*, it may transform two sequences into one whose head guard will be the disjunction of the two previous guard conditions. If $\mathfrak{E}_n$ changed both sequences to $\langle \checkmark::l \rangle$ because of the head guard conditions, it will do the same to the result sequence. If it didn't on either sequence, it can't become true on the combined sequence. If it did on one sequence and not on the other, it will obviously not remove the combined

sequence, because $(G_1 \implies X) \wedge \neg(G_2 \implies X) \implies (G_1 \vee G_2 \implies X)$, so we may conclude.

- Case $C \stackrel{\text{def}}{=} [.] + R$ where R is a process expression. Here we generalise the context by considering the case of delayed sums (cf. Def. 28). So the goal becomes first $P +_l R =_\diamond Q +_l^\sigma R$ where l is a location and σ a substitution from names (public names and place-names) to place-names. We have $[\![P +_l^\sigma R]\!] \stackrel{\text{def}}{=} [\![P]\!] \oplus [\![R]\!]$ and $[\![Q +_l^\sigma R]\!] \stackrel{\text{def}}{=} [\![Q]\!] \oplus [\![R]\!]$ if $l = \epsilon$ and σ is the identity, and in this case the conclusion is a simple fact: $\Upsilon_1([\![P]\!]) \oplus [\![R]\!] = \Upsilon_2([\![Q]\!]) \oplus [\![R]\!]$. Now if $l > \epsilon$ then we have $[\![P +_l^\sigma R]\!] \stackrel{\text{def}}{=} [\![P]\!] \oplus_l^\sigma [\![R]\!]$. Complementarily we have $[\![Q +_l^\sigma R]\!] \stackrel{\text{def}}{=} [\![Q]\!] \oplus_l^\sigma [\![R]\!]$. We may now apply the adequate simplification Υ_1 (resp. Υ_2) for each occurrence of P (resp. Q) with the adequate delay and obtain $\Upsilon_1([\![P]\!]) \oplus_l^\sigma [\![R]\!] = \Upsilon_2([\![Q]\!]) \oplus_l^\sigma [\![R]\!]$. Since $+_l^\sigma$ is not symmetric, we need also to consider the second goal $R +_l^\sigma P =_\diamond R +_l^\sigma Q$, whose proof is obvious from this one.
- Case $C \stackrel{\text{def}}{=} \mu(X).C_X$, which is the solution of the equation $Y =_\diamond P\{Y/X\}$. It is easy to show that if $[\![Y]\!] = [\![Z]\!]$ then $[\![P\{Y/X\}]\!] = [\![P\{Z/X\}]\!]$ (by a simple induction on the contexts for Y and Z). Thus, $[\![\mu(X).P]\!]$ is a fixed point of the function f such that $[\![P\{Y/X\}]\!] = f([\![Y]\!])$. The proof thus relies on the existence of such a fixed point for f, which we ensure by the least fixed point lemma (Lemma 11) and the monotonicity of the language constructors (cf. lemma 1).
- Case $C \stackrel{\text{def}}{=} [.] \parallel R$ where R is a process expression. The case is subsumed by the other cases if we apply the expansion law.

This concludes the congruence proofs for $=_\diamond$ □

Lemma 6.
SUM1 $[\![P + Q]\!] =_\diamond [\![Q + P]\!]$
SUM2 $[\![P + (Q + R)]\!] =_\diamond [\![(P + Q) + R]\!]$
SUM3 $[\![P + END]\!] =_\diamond [\![P]\!]$
SUM4 $[\![[\varphi]P + [\psi]P]\!] = [\![[\varphi \vee \psi]P]\!]$

Proof.

SUM1 $[\![P + Q]\!] = [\![P]\!] \oplus [\![Q]\!]_{l\diamond} =_\diamond [\![Q]\!]_{l\diamond} \oplus [\![P]\!]_{l\diamond}$ by Definition 10.
SUM2 $[\![P + (Q + R)]\!] = [\![P]\!] \oplus [\![Q + R]\!] = [\![P]\!] \oplus [\![Q]\!] \oplus [\![R]\!]$ by Definition 10.
SUM3 $[\![P + END]\!] = [\![P]\!] \oplus [\![END]\!]$. We know that $[\![END]\!] = \{\langle\checkmark::l\rangle\}$ so we may conclude.
SUM4 If $[\![P]\!] = \bigcup \alpha::(\xi, l_1)L.S$ then $[\![[\varphi]P + [\psi]P]\!] = \bigcup \alpha::(\varphi \wedge \xi, l_1)L.S \cup \bigcup \alpha::(\varphi \wedge \xi, l_1)L.S \xrightarrow{merge} \bigcup \alpha::((\varphi \vee \psi) \wedge \xi, l_1)L.S = [\![[\varphi \vee \psi]P]\!]_l$

□

Lemma 7.
RES1 $[\![(\nu n)END]\!] =_\diamond [\![END]\!]$
RES2 $[\![(\nu n)[\varphi]P]\!] =_\diamond [\![[\varphi](\nu n)P]\!]$ *if* $n \notin \varphi$
RES3 $[\![(\nu n)(\nu m)P]\!] =_\diamond [\![(\nu m)(\nu n)P]\!]$
RES4 $[\![(\nu n)\alpha.P]\!] =_\diamond [\![\alpha.(\nu n)P]\!]$ *if* $n \notin \alpha$
RES5 $[\![(\nu n)\alpha.P]\!] =_\diamond [\![END]\!]$ *when* $n = subj(\alpha)$
RES6 $[\![(\nu n)(P + Q)]\!] =_\diamond [\![(\nu n)P + (\nu n)Q]\!]$

Proof.

RES1 $[\![(\nu n)END]\!] = \{\mathfrak{E}_n(S) \mid S \in [\![END]\!]\} = \{\mathfrak{E}_n(\langle\rangle), \mathfrak{E}_n(\langle\checkmark::l\rangle)\} = \{\langle\rangle, \langle\checkmark::l\rangle\} = [\![END]\!]$

RES2 $[\![(\nu n)[\varphi]P]\!] = \{\mathfrak{E}_n(S) \mid S \in [\![[\varphi]P]\!]\}$ Let's examine the cases for $\mathfrak{E}_n$. If $subj(hd(S)) = n$ or $S = \langle\rangle$ we have $\mathfrak{E}_n(S) = \langle\checkmark::\mathfrak{s}(l)\rangle$ (resp. $\langle\rangle$) so in this case we do have $\mathfrak{E}_n([\varphi]S) = [\varphi]\mathfrak{E}_n(S)$. If $hd(S) = c!n$ and $c \neq n$ we have $\mathfrak{E}_n(S) = \mathfrak{F}_n^{\emptyset}(S)\{\nu_{\bar{l}}/n\}$ but since $n \notin \varphi$ the substitution $\{\nu_{\bar{l}}/n\}$, where $\bar{l}$ is the absolute location of the action that causes the escape of name n, and the semantic function $\mathfrak{F}_n^L$ will behave as the identity for S, so we will have $\mathfrak{E}_n([\varphi]S) = ([\varphi]S)\{\nu_{\bar{l}}/n\} = [\varphi](S\{\nu_{\bar{l}}/n\}) = [\varphi]\mathfrak{E}_n(S)$. Otherwise, $\mathfrak{E}_n(S) = hd(S).\mathfrak{E}_n(tl(S))$ and we will have $\mathfrak{E}_n([\varphi]S) = \mathfrak{E}_n(S') = hd(S').\mathfrak{E}_n(tl(S')) = [\varphi]\mathfrak{E}_n(S)$. This allows to conclude that $\forall n \notin \varphi, \mathfrak{E}_n([\varphi]S) = [\varphi]\mathfrak{E}_n(S)$

RES3 Since all the conditions of the $\mathfrak{E}_n$ (resp. $\mathfrak{E}_m$) function depend on n (resp. m) the order of applying the two restriction functions can't have any influence on their effect, so $\mathfrak{E}_n \circ \mathfrak{E}_m = \mathfrak{E}_m \circ \mathfrak{E}_n$

RES4 $[\![(\nu n)\alpha.P]\!] = \{\mathfrak{E}_n(S) \mid S \in [\![\alpha.P]\!]\} = \{\mathfrak{E}_n(S) \mid S \in \alpha::l \triangleright .[\![P]\!]\}$. Except for $\langle\rangle$ and $\langle\checkmark::l\rangle$, all the sequences in $\alpha::l \triangleright .[\![P]\!]$ begin by $\alpha::l\triangleright$. We know that $\mathfrak{E}_n$ is the identity for $\langle\checkmark::l\rangle$ and $\langle\rangle$ so those two sequences will not be problems. The other sequences will be of the form $S = \alpha::l \triangleright .S'$. Since $n \notin \alpha$, we will have $\mathfrak{E}_n(S) = \alpha::l \triangleright .\mathfrak{E}_n(S')$

RES5 $[\![(\nu n)\alpha.P]\!] = \{\mathfrak{E}_n(S) \mid S \in [\![\alpha.P]\!]\} = \{\mathfrak{E}_n(S) \mid S \in \alpha::l \triangleright .[\![P]\!]\}$. Except for $\langle\rangle$ and $\langle\checkmark::l\rangle$, all the sequences in $\alpha::l \triangleright .[\![P]\!]$ begin by $\alpha::l\triangleright$. Like before, $\mathfrak{E}_n$ behaves as the identity for those. The other sequences will be of the form $S = \alpha::l \triangleright .S'$, but here, $n = subj(\alpha)$ so $\mathfrak{E}_n(S) = \langle\checkmark::l\rangle$ by definition.

RES6 $[\![(\nu n)(P+Q)]\!] = \{\mathfrak{E}_n(S) \mid S \in [\![P+Q]\!]\} = \{\mathfrak{E}_n(S) \mid S \in [\![P]\!]_{l\diamond} \cup [\![Q]\!]_{l\diamond}\} = \{\mathfrak{E}_n(S) \mid S \in [\![P]\!]_{l\diamond}\} \cup \{\mathfrak{E}_n(S) \mid S \in [\![Q]\!]_{l\diamond}\} = [\![(\nu n)P]\!]_{l\diamond} \cup [\![(\nu n)Q]\!]_{l\diamond} = [\![(\nu n)P + (\nu n)Q]\!]$

□

Lemma 8.
GRD1 $[\![[false]P]\!] =_\diamond [\![END]\!]$
GRD2 $[\![[true]P]\!] =_\diamond [\![P]\!]$
GRD3 $[\![[\varphi]P]\!] =_\diamond [\![[\psi]P]\!]$ *if* $\varphi \iff \psi$
GRD4 $[\![[\varphi](P+Q)]\!] =_\diamond [\![[\varphi]P + [\varphi]Q]\!]$
GRD5 $[\![(\nu a)[a \neq b]P]\!] =_\diamond [\![(\nu a)P]\!]$ *if* $b \neq a$
GRD6 $[\![[\varphi]\alpha.P]\!] =_\diamond [\![[\varphi]\alpha.[\varphi]P]\!]$ *if* $bn(\alpha) \notin \varphi$
GRD7 $[\![[\varphi]\alpha.P]\!] =_\diamond [\![[\varphi]\alpha\{a/b\}.P]\!]$ *if* $\varphi \implies a = b$
GRD8 $[\![[\varphi][\psi]P]\!] =_\diamond [\![[\varphi \wedge \psi]P]\!]$

Proof.

GRD1 $[\![[false]P]\!] = \{\alpha::falseL.S \mid \alpha::lL \in [\![P]\!]\} \xrightarrow{\phi} \{\langle\checkmark::l\rangle\}$ where ϕ is the application of transformation *false* from Table 4.

GRD2 If $[\![P]\!] = \bigcup \alpha::(\varphi, l_1)L.S$ then $[\![[true]P]\!] = \bigcup \alpha::(true \wedge \varphi, l_1)L.S = [\![P]\!]$

GRD3 Our guards being logical expressions, all laws of first order logic apply so φ and ψ are considered the same object

GRD4 If $[\![P]\!] = \bigcup \alpha::(\xi, l_1)L.S$ and $[\![Q]\!] = \bigcup \beta::(\eta, m_1)M.S'$ then $[\![[\varphi](P+Q)]\!] = \bigcup \alpha::(\varphi \wedge \xi, l_1)L.S \oplus \bigcup \beta::(\varphi \wedge \eta, m_1)M.S' = [\![[\varphi]P + [\varphi]Q]\!]$

GRD5 $[\![(\nu a)[a \neq b]P]\!] = \{\mathfrak{E}_a(S) \mid S \in [\![[a \neq b]P]\!]\} = \{\mathfrak{E}_a(\alpha::(a \neq b \wedge \varphi, l)\ldots).S')\} = \{\alpha::(\varphi\{true/a \neq b\}, l)\ldots.\mathfrak{E}_(S')\} = [\![(\nu a)P]\!]$

GRD6 Application of the *perco* rewrite rule removes guards that have already enforced earlier in the sequence

GRD7 From Table 2, any occurrence of a or b will be replaced by $\{a, b\}$ which allows to conclude

GRD8 From Table 2, when calculating the trace set of a guarded process the guard will be put in conjunction with that of the head location of the head observation in all sequences, and the substitutions will be composed. The result is trivial by associativity of conjunction and of function composition

□

Lemma 9.
$[\![P \parallel Q]\!] = [\![\sum \alpha_i{::}l_i.(P_i \parallel \beta_j{::}l_j.Q_j) + \sum \beta_j{::}l_j.(\alpha_i{::}l_i.P_i \parallel Q_j) + [c = d]\tau.(P_i \parallel Q_j\{d/x\})]\!]$

Proof. Soundness of the expansion law is quite easily proved by induction on sequences. The induction hypothesis is that if the property is true for $S_i \oplus S_j$ it is for $\bigcup_{i,j} \alpha_i{::}l_i.S_i \oplus \beta_j{::}l_j.S_j$. Except if one of the processes can only terminate immediately, in which case the other one is the result of the parallel composition (which provides a fixpoint for our induction), we have

$$
\begin{aligned}
[\![P \parallel Q]\!] &= \bigcup\{\alpha_i{::}l_i.S_i\} \oplus \bigcup\{\beta_j{::}l_j.S_j\} \text{ where } \alpha_i{::}l_i.S_i \in [\![P]\!],\ \beta_j{::}l_j.S_j \in [\![Q]\!] \\
&= \bigcup\{interleave(\alpha_i{::}l_i.S_i, \beta_j{::}l_j.S_j)\} \oplus \bigcup\{interleave(\beta_j{::}l_j.S_j, \alpha_i{::}l_i.S_i)\} \\
&\quad \oplus \bigcup\{intersync(\alpha_i{::}l_i.S_i, \beta_j{::}l_j.S_j)\} \oplus \bigcup\{intersync(\beta_j{::}l_j.S_j, \alpha_i{::}l_i.S_i)\} \\
&= \bigcup\{\alpha_i{::}l_i.(S_i \oplus \beta_j{::}l_j.S_j)\} \oplus \bigcup\{\beta_j{::}l_j.(S_j \oplus \alpha_i{::}l_i.S_i)\} \\
&\quad \oplus \bigcup\{\gamma_z{::}w_a w'_b(c_a = d_b \wedge \varphi_a \wedge \psi_b, \widetilde{\triangleright}) l_z.T_z \\
&\quad \mid \bigcup\{\alpha_i{::}l_i.S_i\} \oplus \bigcup\{\beta_j{::}l_j.S_j\}\{e_a/\rho_{\overline{l_j}}\} = \bigcup_{k=1}^{p} \gamma_k{::}l_k.T_k\} \oplus \dots \\
&= [\![\sum \alpha_i{::}l_i.(P_i \parallel \beta_j{::}l_j.Q_j)]\!] \oplus [\![\sum \beta_j{::}l_j.(\alpha_i{::}l_i.P_i \parallel Q_j)]\!] \\
&\quad \oplus [\![[c = d]\tau.(P_i \parallel Q_j\{d/x\})]\!] \text{ where } \alpha_i = c!d \text{ and } \beta_j = d?x \text{ or the converse} \\
&= [\![\sum \alpha_i{::}l_i.(P_i \parallel \beta_j{::}l_j.Q_j) + \sum \beta_j{::}l_j.(\alpha_i{::}l_i.P_i \parallel Q_j) + [c = d]\tau.(P_i \parallel Q_j\{d/x\})]\!]
\end{aligned}
$$

□

For the sake of readability, the above proof elides as ... the converse case of the communication condition since it behaves exactly the same, except for the fact that the sending and receiving processes are reversed.

5. Refinement

One advantage of manipulating prefix-closed sets of sequences as trace sets is that set inclusion then provides a simple yet powerful means for behavioural refinement.

Definition 27. *A process P refines a process Q, which we denote $Q \sqsubseteq P$, iff $\exists T \subseteq [\![Q]\!]$ such that $[\![P]\!] =_\diamond T$. The relation is equivalently denoted $[\![P]\!] \subseteq_{/=_\diamond} [\![Q]\!]$.*

Before investigating the main properties of the order, we introduce a syntactic construction that, indeed, characterises properly the notion of refinement in the proposed model. We remind that for a given trace set T, its pretrace set $T \uparrow l$ (resp. its postrace set $T \downarrow l$) corresponds to the subtrace containing all the prefixes (resp. suffixes) of T before (resp. after) l. A further notation is that of the *trace complement* $\overline{T}(l)$ which is defined as $T \setminus (T \uparrow l^\frown T \downarrow l)$. These are all the sequences that do not go "through" l. It is maybe easier to remind the invariant: $T = \overline{T}(l) \cup T \uparrow l^\frown T \downarrow l$. Note that the complement set may be empty and thus may not be a valid trace set, i.e. its codomain is $\mathcal{T} \uplus \{\emptyset\}$. We may now introduce the notion of *delayed sum*, a strict generalisation of the sum operator, as follows:

Definition 28. *Let P and Q be arbitrary processes, l a location and σ a substitution from names $(a, b, \rho_l, \dots)$ to place names only $(\nu_\triangleright, \rho_{\triangleright\widetilde{\triangleright}}, \dots)$. The delayed sum operator at l, denoted $P +_l^\sigma Q$, is defined as follows:*

$$[\![P +^{\sigma}_{l} Q]\!] \stackrel{def}{=} [\![P]\!] \oplus^{\sigma}_{l} [\![Q]\!]$$
$$T_1 \oplus^{\sigma}_{l} T_2 \stackrel{def}{=} T_1 \uparrow l^{\frown}(\, T_1 \downarrow l\{s^n_i \leftarrow s^{n+m}_i\}_\epsilon \cup T_2\{s^m_j \leftarrow s^{n+m}_{j+n}\}_\epsilon \\ \{V_{lm}/V_m \mid V \in \{\rho,\nu\},\ V_m \notin \mathit{support}(\sigma)\}\sigma\,)$$

Notation 2. *Let Id be the identity substitution. For the sake of brevity, $P +^{Id}_{l} Q$ is denoted $P +_l Q$*

Delayed sums are similar to ordinary sums, except that the effect of the operator – the branching point – is delayed until the path l has been followed in the left operand. For example, $\alpha.P +^{\emptyset}_{\triangleright} Q = \alpha.(P + Q)$. An explicit substitution must be provided when the right-hand process refers to names that have been created before (wrt. the prefix order on locations) the location where it is inserted in the left-hand process behaviour. When the substitution is not necessary it may be omitted. It is important to note that the operator is not symmetric[6]. The delayed sum operator is clearly a generalisation of ordinary sum.

Proposition 2. $P + Q =_\diamond P +_\epsilon Q$

Proof. Here we assert that an ordinary sum is the same as a delayed sum with delay ϵ. If we apply Definition 28, then we have for any location l, $[\![P +_\epsilon Q]\!] \stackrel{\text{def}}{=} \overline{[\![P]\!]}(\epsilon) \cup ([\![P]\!] \uparrow \epsilon^{\frown}([\![P]\!] \downarrow \epsilon\{s^n_i \leftarrow s^{n+m}_i\}_\epsilon \cup [\![Q]\!]\{s^m_j \leftarrow s^{n+m}_{j+n}\}_\epsilon))$. By Definition 6 we have for any trace set T, $T \uparrow \epsilon \stackrel{\text{def}}{=} \{\langle\rangle\}$ and $T \downarrow \epsilon \stackrel{\text{def}}{=} T$. The trace set complement $\overline{T}(\epsilon)$ is thus the empty set, which gives $[\![P +_\epsilon Q]\!] \stackrel{\text{def}}{=} [\![P]\!]\{s^n_i \leftarrow s^{n+m}_i\}_\epsilon \cup [\![Q]\!]\{s^m_j \leftarrow s^{n+m}_{j+n}\}_\epsilon = [\![P]\!] \oplus [\![Q]\!] = [\![P + Q]\!]$. □

There is a tight connection between delayed sums and refinement, as characterised by the following lemma:

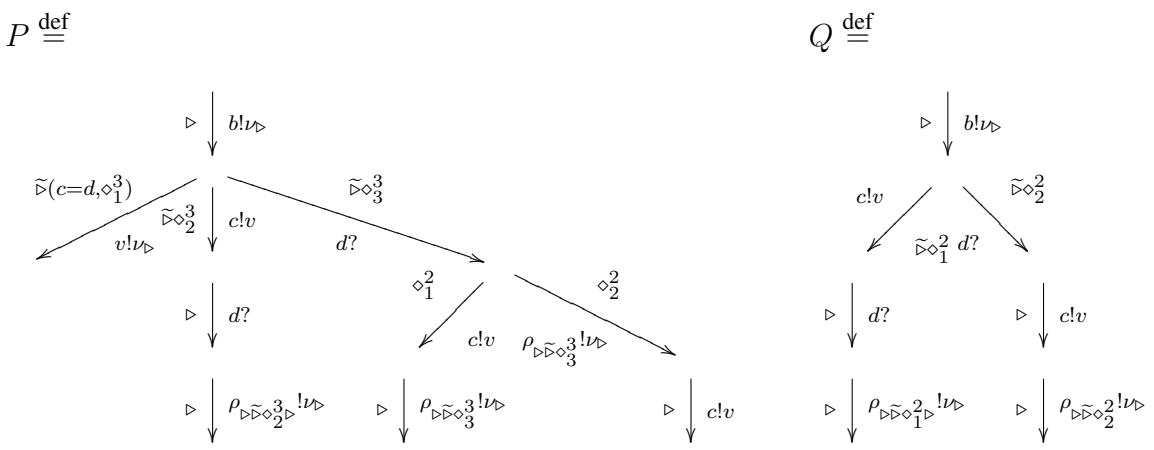
Lemma 10. $P \sqsubseteq Q \Longleftrightarrow \exists \mathcal{RL} = \bigcup^n_{i=1}\{(R_i, l_i, \sigma_i)\}$ *s. t.* $P =_\diamond Q +^{\sigma_1}_{l_1} R_1 \ldots +^{\sigma_n}_{l_n} R_n$

Proof. For the if part, let l' be an arbitrary location and $T \subseteq [\![P]\!]\{\epsilon \leftarrow l'\}$ a trace set such that $T =_\diamond [\![Q]\!]\{\epsilon \leftarrow l'\}$. Such a trace set exists by hypothesis and Definition 27. Now we consider $T' \stackrel{\text{def}}{=} [\![P]\!]\{\epsilon \leftarrow l'\} \setminus T$ the set of all sequences that are in the behaviour of P but not in Q. Note that if T' is empty then we are finished and P and Q exactly match through reflexivity. We now take all the maximal locations $l \in \widetilde{l}$ such that $T' \uparrow l \neq \emptyset$. The set $\widetilde{l}$ is maximal with respect to location prefixing in that if $l_1, l_2 \in \widetilde{l}$ then $l_1 \not\leq l_2$ and $l_2 \not\leq l_1$. For each of such location l, we identify a process R_l such that $[\![P]\!]\{\epsilon \leftarrow l'\} \downarrow l = T' \downarrow l\{s^n_i \leftarrow s^{n+m}_i\} \cup [\![R_l]\!]\{\epsilon \leftarrow l'\}\{s^m_j \leftarrow s^{n+m}_{j+n}\}\{V_{lm}/V_m \mid V \in \{\rho,\nu\},\ V_m \notin \mathit{support}(\sigma)\}\sigma$. Such a process exists since $T' \downarrow l$ is not empty and strictly contained in $[\![P]\!]\{\epsilon \leftarrow l'\}$ by definition. Hence, $[\![P]\!]\{\epsilon \leftarrow l'\} = \bigcup_{l \in \widetilde{l}}([\![P]\!]\{\epsilon \leftarrow l'\} \uparrow l^{\frown}[\![P]\!]\{\epsilon \leftarrow l'\} \downarrow l\{s^n_i \leftarrow s^{n+m}_j\}) \cup T\{s^m_j \leftarrow s^{n+m}_{j+n}\}\{V_{lm}/V_m \mid V \in \{\rho,\nu\},\ V_m \notin \mathit{support}(\sigma)\}\sigma$, which we may finally rephrase as $P =_\diamond Q +^{\sigma_1}_{l_1} R_1 \ldots +^{\sigma_n}_{l_n} R_n$

For the only if part we suppose $P =_\diamond Q +^{\sigma_1}_{l_1} R_1 \ldots +^{\sigma_n}_{l_n} R_n$, and for each $l_i \in \widetilde{l}$, $[\![Q +_\epsilon R]\!]_{l_i} \stackrel{\text{def}}{=} \overline{[\![Q]\!]_{l_i}}(\epsilon) \cup ([\![Q]\!]_{l_i} \uparrow \epsilon^{\frown}([\![Q]\!]_{l_i} \downarrow \epsilon\{s^n_i \leftarrow s^{n+m}_i\}_\epsilon \cup [\![R]\!]_{l_i}\{s^m_j \leftarrow s^{n+m}_{j+n}\}_\epsilon\{V_{lm}/V_m \mid V \in \{\rho,\nu\},\ V_m \notin \mathit{support}(\sigma)\}\sigma))$ (Definition 28), which implies trivially that $[\![Q]\!]\{\epsilon \leftarrow l'\} \subseteq \bigcup_i [\![Q +^{\sigma_i}_{l_i} R_i]\!]\{\epsilon \leftarrow l'\}$ and, following the hypothesis, $[\![Q]\!]\{\epsilon \leftarrow l'\} \subseteq_{/=_\diamond} [\![P]\!]\{\epsilon \leftarrow l'\}$. We thus conclude $P \sqsubseteq Q$ □

We now illustrate the delayed sum characterisation of refinement. Consider the processes of Figure 3. Intuitively, it should be the case that $P \sqsubseteq Q$ (i.e. Q refines P) because P

[6] We think the dual notion of "premature sums" also worth studying but this requires a notion of "reversible locations" that we have to investigate furthermore.



$$\begin{aligned}[\![P]\!] = \{\ & \langle b!\nu_{\triangleright}::\triangleright, v!\nu_{\triangleright}::\widetilde{\triangleright}(c=d,\diamond_1^3)\rangle,\\ & \langle b!\nu_{\triangleright}::\triangleright, c!v::\widetilde{\triangleright}\diamond_2^3, d?::\triangleright, \rho_{\triangleright\widetilde{\triangleright}\diamond_2^3\triangleright}!\nu_{\triangleright}::\triangleright\rangle,\\ & \langle b!\nu_{\triangleright}::\triangleright, d?::\widetilde{\triangleright}\diamond_3^3, c!v::\diamond_1^2, \rho_{\triangleright\widetilde{\triangleright}\diamond_3^3}!\nu_{\triangleright}::\triangleright\rangle,\\ & \langle b!\nu_{\triangleright}::\triangleright, d?::\widetilde{\triangleright}\diamond_3^3, \rho_{\triangleright\widetilde{\triangleright}\diamond_3^3}!\nu_{\triangleright}::\diamond_2^2, c!v::\triangleright\rangle\ \}\end{aligned}$$

$$\begin{aligned}[\![Q]\!] = \{\ & \langle b!\nu_{\triangleright}::\triangleright, c!v::\widetilde{\triangleright}\diamond_1^2, d?::\triangleright, \rho_{\triangleright\widetilde{\triangleright}\diamond_1^2\triangleright}!\nu_{\triangleright}::\triangleright\rangle,\\ & \langle b!\nu_{\triangleright}::\triangleright, d?::\widetilde{\triangleright}\diamond_2^2, c!v::\triangleright, \rho_{\triangleright\widetilde{\triangleright}\diamond_2^2}!\nu_{\triangleright}::\triangleright\rangle\ \}\end{aligned}$$

$$P = (\nu a)b!a.\tau.([c=d]v!a.VOID + c!v.d?x.x!a.VOID + d?x.(c!v.x!a.VOID + x!a.c!v.VOID))$$

$$Q = (\nu a)b!a.\tau.(c!v.d?x.x!a.VOID + d?x.c!v.x!a.VOID)$$

Figure 3. Illustrating delayed sums (1)

may *at least* perform all the actions and non-deterministic choices of Q, but can do even more of course. However, it is not the case that $P =_\diamond P + Q$ (see Fig. 4) so the (standard) sum operator does not characterise refinement in a complete way. As Fig. 5 makes clear Lemma 10 guarantees the existence of a delay l, σ and a delayed sum of processes $\widetilde{R}$ such that $P =_\diamond Q +_l^\sigma \widetilde{R}$.

Refinement, as characterised by delayed sums, is a proper ordering relation with (parametrised) monotone properties on the language constructors.

Theorem 1.
$P \sqsubseteq P$
$P \sqsubseteq Q \wedge Q \sqsubseteq R \implies P \sqsubseteq R$
$P \sqsubseteq Q \wedge Q \sqsubseteq P \implies P =_\diamond Q$
$P \sqsubseteq Q \implies \exists \widetilde{l},\ C_{\widetilde{l}}[P] \sqsubseteq C_{\widetilde{l}}[Q]$ *for any language context* C

All these properties use a similar proof schema that consists in replacing each property in the context of the split equivalence and delayed sums.

Proof.

- (reflexivity) By definition, $P +_\epsilon P \sqsubseteq P$ and since $P +_\epsilon P =_\diamond P + P$ (by lemma 2) and $P + P =_\diamond P$so we conclude $P \sqsubseteq P$.

$P +_\epsilon Q \stackrel{\text{def}}{=}$

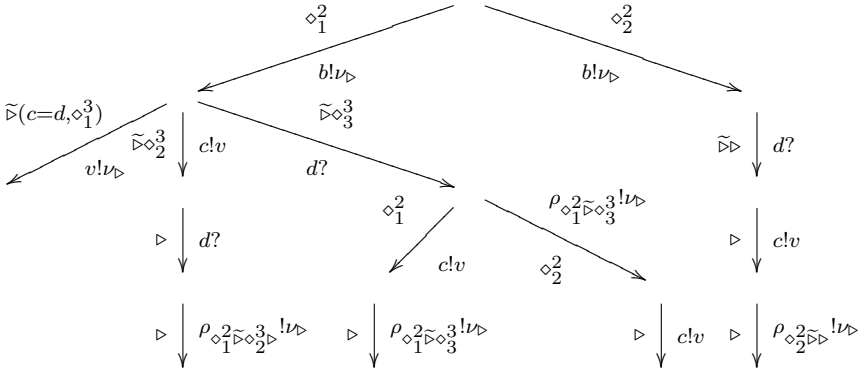
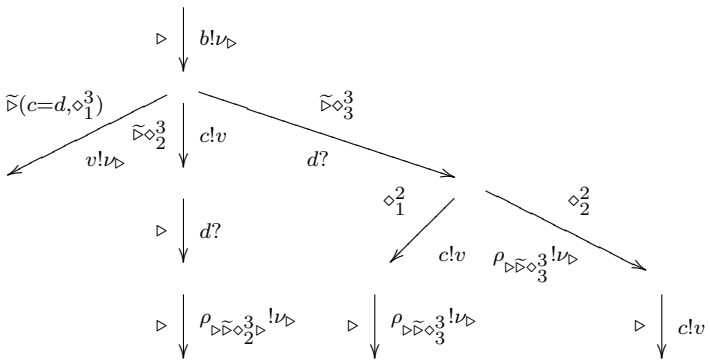


Figure 4. Illustrating delayed sums (2)

$$Q +^{\{\nu_\triangleright/a\}}_{\widetilde{\triangleright\triangleright}} R_1 +^{\{\rho_{\triangleright\widetilde{\triangleright}\diamond_3^3}/a,\ \nu_\triangleright/b\}}_{\triangleright\widetilde{\triangleright}\diamond_3^3} R_2 \stackrel{\text{def}}{=}$$



$$R_1 = [c = d]v!a.VOID$$
$$[\![R_1]\!] = \{\ \langle v!a{::}(c = d, \triangleright)\rangle\}$$

$$R_2 = a!b.c!v.VOID$$
$$[\![R_2]\!] = \{\langle a!b{::}\triangleright, c!v{::}\triangleright\rangle\ \}$$

Figure 5. Illustrating delayed sums (3)

- (transitivity) We have $P \sqsubseteq Q$ and $Q \sqsubseteq R$ so $\exists l_1 \ldots l_n,\ A_1 \ldots A_n$ s. t. $Q +_{l_1} A_1 \ldots +_{l_n} A_n =_\diamond P$ and $\exists l'_1 \ldots l'_n,\ A'_1 \ldots A'_n$ s. t. $R +_{l'_1} A'_1 \ldots +_{l'_n} A'_n =_\diamond Q$. Since split-equivalence is transitive, thus $R +_{l'_1} A'_1 \ldots +_{l'_n} A'_n +_{l_1} A_1 \ldots +_{l_n} A_n =_\diamond P$ which allows us to conclude.
- (antisymmetry) We have $P \sqsubseteq Q$ so $\exists l_1 \ldots l_n,\ A_1 \ldots A_n$ s. t. $Q +_{l_1} A_1 \ldots +_{l_n} A_n =_\diamond P$. And also by hypothesis $Q \sqsubseteq P$ so $\exists l'_1 \ldots l'_m,\ B_1 \ldots B_m$ s. t. $P +_{l'_1} B_1 \ldots +_{l'_m} B_m =_\diamond Q$. In terms of trace sets, it is easy to derive the property that $\bigcup_{i=1}^n \{A_i\} = \bigcup_{i=1}^m \{B_i\}$ and so $P =_\diamond Q$.
- (monotonicity) The property can be rewritten as follows: $\exists l_1 \ldots l_n,\ R_1 \ldots R_n$ s. t.

$Q =_\diamond P +_{l_1} R_1 \ldots +_{l_n} R_n \implies C[Q] =_\diamond C[P +_{l_1} R_1 \ldots +_{l_n} R_n]$ whose proof is a particular case of the congruence property for delayed sums.

This concludes the proofs for Lemma 1. □

Note that unsurprisingly the congruence result is relative to a given set of locations delays, it does not follow from $P \sqsubseteq Q$ that $P +^\sigma_l R \sqsubseteq Q +^\sigma_l R$, since P and Q may have differently ordered branches. Instead, if $P \sqsubseteq Q$ and $\exists P' =_\diamond P$ such that $P' \uparrow l = Q \uparrow l$, then $P' +_l R \sqsubseteq Q +_l R$.

The refinement ordering is trivially bounded by process VOID on one side, and by process RUN such that $[\![\text{RUN}]\!] \stackrel{\text{def}}{=} \mathcal{T}$ on the other side. A simple fact is that for any process P we have RUN $\sqsubseteq P \sqsubseteq$ VOID. A much more general result about $\sqsubseteq$ is the following one:

Theorem 2. *$\sqsubseteq$ is a complete lattice*

The property is relatively easy to exhibit if we interpret it in terms of trace sets. It says, in fact, that $(\mathcal{T}, \subseteq_{/=_\diamond})$ (i.e. the subset relation for the equivalence classes with respect to $=_\diamond$) itself possesses a complete lattice structure. Simple set theoretic arguments (using $\bigcap_{/=_\diamond}$ and $\bigcup_{/=_\diamond}$, the generalised intersection and union operators with respect to $\mathcal{T}_{/=_\diamond}$) then suffice to establish the property.

This leads to the most important result of the section:

Lemma 11. *Let ϕ a function from process expressions to process expressions. If ϕ is monotone with respect to $\sqsubseteq$ then it admits a least fixed point with respect to $=_\diamond$, i.e. $\phi(P) =_\diamond P$ for any process P. Moreover, if ϕ is continuous with respect to $\sqsubseteq$, then the least fixed point is $\bigsqcap\{\phi^n(\textit{VOID}) \mid n \in \mathbb{N}\}$.*

Both the properties correspond to transpositions of Tarski's lemmas in the realm of the proposed framework, considering the complete lattice structure of the refinement order.

Thanks to lemma 11, we may now introduce a general rule for recursion as follows:

[rec] $[\![\mu(X).P]\!]$ is the least fixed point solution of $[\![P\{Y/X\}]\!] =_\diamond f([\![Y]\!])$ with f such that $Y =_\diamond P\{Y/X\}$

Note that the existence of f and the existence or unicity of its least fixed point are not always guaranteed. An example is with unguarded recursions (e.g. $\mu(X).X$) for which nothing is recorded. The explicit recording of divergences would allow for a more thorough treatment of unguarded recursion, which is left as a future work.

6. Related Work

To our knowledge, there are very few investigations aiming at developing mobile extensions for CSP. In [10] the authors propose to encode mobile channels as processes. This makes sense from the point of view of execution environments and close-world semantics, but channel mobility has an important impact on the theory, and thus something must be proposed at that level to be able to reason about such mobile extensions. In a recent yet unpublished paper [11], an interesting proposition is made for an encoding of both a channel-passing version of CSP and of the π-calculus within CSP+, the language of CSP enriched by a construction for exceptional behaviours [12]. The channel-passing variant of the parallel construct is more about dynamic alphabets and the explicit manipulation of read-write access rights on channels, but it is not mobility in the sense of the π-calculus. In particular the scope of name remains static in this variant. Concerning the encoding of the π-calculus itself, the proposition remains mostly informal as of today but the general idea is to encode the effect

of binders as non-deterministic choices among the (potentially infinite) possibilities of name substitutions involved. For the input binder this idea clearly relates to the early semantics for the pi-calculus, and it is shown that for finite-state problem the choices are also finite (using open bisimilarity). However, we do not convey the idea of early semantics in our model because it has a significant cost when conducting proofs or developing verification algorithms. The idea is that one has to consider all the possible substitutions of names in order to solve the problem, which can be infinite (early case) or restricted to the finite number of names actually used in the process being analysed (open case). Moreover this does not solve the compositionality issue because channel names that are not bound are not considered by the substitution. In our case we provide a single, uniquely defined name — attached to the absolute location of the considered observation — to serve the same purpose (and more). The advantage of our denotation, also, is that it is not parametric unlike [11] because of the explicit manipulation of infinite replacement sets to capture freshness. In our case a single location must be recorded instead. Unlike our proposition, the model also seems to suffer from the same compositionality as the "real" π-calculus. A very positive point of the presented model is the natural switch from the operational to the denotational characterisation and vice-versa. In our case we had to exhibit a non-trivial axiomatisation of the denotation, which is quite an involved process (especially the completeness part of the adequacy theorem, cf. our technical report). The positive point is that the axiomatisation gives us a minimal set of laws for the proposed language constructs.

In comparison with the standard failure-divergence (FD) model of CSP, an interesting characteristic of the trace model we propose is that it integrates well the concepts of (standard) trace sets, the encoding of the branching structure and the mobile features. There is no need for separate specifications (traces, failures and divergences) and the resulting denotation conveys the complete lattice structure of the plain trace semantics. This is mostly thanks to the notion of location. In return, the manipulation of the trace sets must ensure the correct (re-)location of the observations, which makes the definitions more intricate than those of FD, even if we remove the mobility part. We show, however, that thanks to trace normalisation we are able to abstract away from the fine-grained nature of the locations. The proof techniques we propose beyond the denotation itself do not suffer from the fine-grained nature of locations. In the same line of idea, we provide in [13] the first sketch of a CSP-like predicate logic built on the present model. The logic allows the practical reasoning on mobile systems without having to deal directly with e.g. explicit locations, escape functions or split-relations.

The study of the π-calculus semantics from a denotational point of view has also been investigated in [14,15] with the objective of characterising full abstraction lemmas wrt. testing preorders. Most interpretations consider set-theoretic trace models built from the operational semantics. In this paper, we adopt a complementary point of view of building trace models directly from process expressions, with the goal of providing proof principles and techniques directly applicable on trace models as in CSP. While [14,15] relate trace-based denotations, so-called *acceptance traces*, with tests based on the operational semantics in order to provide full abstraction lemmas, we address complementary questions at the intersection of denotational and axiomatic semantics. Nevertheless, there are some affinities between the two approaches, most notably the development of behavioural preorder relations, and set-inclusion as the general driving principle. The trace model of [15] is also different in that it implements early commitments – ours are late in comparison. Must testing equivalence is weaker than split-equivalence, with tau-laws releasing even more constraints than weak late congruence [16]. This obviously raises compositionality issues.

7. Conclusion and Future Work

In this paper we have shown that it is possible to model mobility in an observational and compositional way. The resulting model is not as concise and elegant as the standard CSP model but this is, in our opinion, the price to pay for the characterisation of mobility. There is, however, a form of minimalism in the model. The single concept of *location* plays for example quite a versatile role. They are used to encode the branching structure of the process within trace sets (which is why they were introduced at first in [2]) but they are also used to give fresh identities to dynamic names, which are in our opinion the central characteristic of mobile — dynamic — behaviours. The issue with locations is that, similarly to de Bruijn indices, we have to ensure their consistency, especially when composing trace sets. From a mathematical point of view, there may exist cleaner foundations where, for example, permutations of branches and trace transformations would come "for free". But this is not a certitude because concurrent systems are not "pure" mathematical objects to start with. What we propose in this paper is the simplest model we were able to develop. Our intuition is that its complexity is inherent to the phenomena we try to characterise. A lesson we learnt from experience is that it is much better to go from the denotation to the language, as in CSP, than the converse, although it is of course necessary to have some intuitions about the language at first. Initially we tried the other way around (starting from the π-calculus directly) and it generally led to dead ends.

From a practical point of view, the trace normalisation principles of the model appear as quite attractive. We are now developing algorithmic principles based on normalisation that will hopefully lead to the development of an equivalence and refinement checking tool for the proposed language.

References

[1] Robin Milner. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.
[2] Frederic Peschanski. On Linear Time and Congruence in Channel-Passing Calculi. In *Communicating Process Architectures 2004*, pages 39–54. IOS Press, 2004.
[3] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
[4] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.
[5] Frédéric Peschanski and Joël-Alexis Bialkiewicz. A denotational model for mobile processes. Technical report, LIP6, `http://www-poleia.lip6.fr/~pesch/data/tracepitr08.pdf`, 2008.
[6] Ugo Montanari and Marco Pistore. History-dependent automata: An introduction. In Marco Bernardo and Alessandro Bogliolo, editors, *SFM*, volume 3465 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2005.
[7] M. Hennessy and H. Lin. Symbolic bisimulations. *Theor. Comput. Sci.*, 138(2):353–389, 1995.
[8] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
[9] Frédéric Peschanski and Joël-Alexis Bialkiewicz. Modelling and verifying mobile systems using pi-graphs. In Mogens Nielsen, Antonín Kucera, Peter Bro Miltersen, Catuscia Palamidessi, Petr Tuma, and Frank D. Valencia, editors, *SOFSEM*, volume 5404 of *Lecture Notes in Computer Science*, pages 437–448. Springer, 2009.
[10] Frederick R. M. Barnes and Peter H. Welch. A CSP Model for Mobile Channels. In Frederick R. M. Barnes, Jan F. Broenink, Alistair A. McEwan, Adam Sampson, G. S. Stiles, and Peter H. Welch, editors, *Communicating Process Architectures 2008*, pages –, sep 2008.
[11] A. W. Roscoe. On the expressiveness of csp. `http://www.comlab.ox.ac.uk/publications/publication2766-abstract.html`.
[12] A. W. Roscoe. The three platonic models of divergence-strict csp. In *Proceedings of the 5th international colloquium on Theoretical Aspects of Computing*, pages 23–49, Berlin, Heidelberg, 2008. Springer-Verlag.
[13] J-A.Bialkiewicz and F.Peschanski. Logic for mobility: a denotational approach. In *Logic, Agents and Mobility (LAM'09)*, pages 44–59. Technical report (Duhram University), 2009.
[14] Michele Boreale and Rocco De Nicola. Testing equivalence for mobile processes. *Inf. Comput.*, 120(2):279–303, 1995.

[15] Matthew Hennessy. A fully abstract denotational semantics for the pi-calculus. *Theor. Comput. Sci.*, 278(1-2):53–89, 2002.
[16] Huimin Lin. Complete inference systems for weak bisimulation equivalences in the pi-calculus. *Inf. Comput.*, 180(1):1–29, 2003.