pNets: an Expressive Model for Parameterised Networks of Processes (Extended Version)

Ludovic Henrio, Eric Madelaine, Min Zhang

pNets: an Expressive Model for Parameterised Networks of Processes (Extended Version)

Ludovic HENRIO , Eric MADELAINE , Min ZHANG

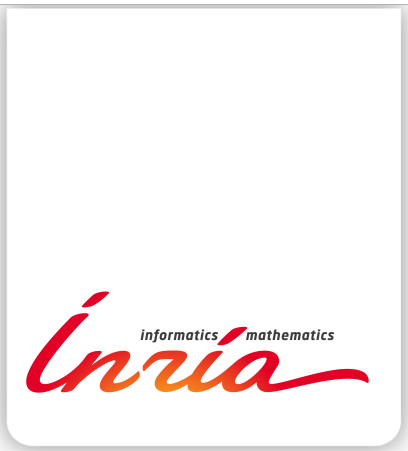
RESEARCH REPORT

N° 8579

July 2014

Common Project-Team Scale

ISSN 0249-6399 ISRN INRIA/RR--8579--FR+ENG



pNets: an Expressive Model for Parameterised Networks of Processes (Extended Version)

Ludovic HENRIO *, Eric MADELAINE † *, Min ZHANG ‡

Common Project-Team Scale

Research Report n° 8579 — version 2 — initial version July 2014 — revised version November 2014 — 23 pages

Abstract: This article studies Parameterised Networks of Automata (pNets) from a theoretical perspective. We illustrate the expressiveness of pNets by showing how to express a wide range of classical constructs of (value-passing) process calculi, but also how we can easily express complex interaction patterns used in modern distributed systems. Our framework can model full systems, using (closed) hierarchies of pNets; we can also build (open) pNet systems expressing composition operators. Concerning more fundamental aspects, we define a strong bisimulation theory specifically for the pNet model, prove its properties, and illustrate it on some examples. One of the original aspects of the approach is to relate the compositional nature of pNets with the notion of bisimulation; this is exemplified by studying the properties of a flattening operator.

Key-words: Formal methods, Distributed systems, Behavioral semantics, Structured labelled transition systems, Bisimulation

SCALE is a common team between the I3S Lab (Univ. of Nice Sophia-Antipolis and CNRS) and INRIA Sophia Antipolis Méditerranée

* University of Nice Sophia Antipolis, CNRS, UMR 7271, 06900 Sophia Antipolis, France
† INRIA Sophia Antipolis Méditérannée, BP 93, 06902 Sophia Antipolis, France
‡ MoE Engineering Research Center for Software/Hardware Co-design Technology and Application, East China Normal University, 200062, Shanghai, China

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

pNets: un modéle expressif pour les réseaux de processus paramétrés (Version étendue)

Résumé : Cet article étudie les Réseaux Paramétrés d'Automates Synchronisés (pNets) d'un point de vue théorique. Nous illustrons l'expressivité du modèle pNets en montrant comment encoder un large éventail de constructions classiques des calculs de processus avec passage de donnés, mais aussi la façon dont nous pouvons facilement exprimer les schémas d'interaction complexes utilisés dans distribués systèmes modernes. Notre formalisme permet de modéliser des systèmes complets, utilisant une hiérarchie (fermée) de pNets; mais nous pouvons également construire des systèmes ouverts exprimant des opérateurs de composition. Concernant les aspects plus fondamentaux, nous définissons une théorie de bisimulation forte spécifiquement adpatée à notre modèle, prouvons ses propriétés, et l'illustrons sur certains exemples. Un des aspects originaux de la démarche est de relier la composition des pNets avec la notion de bisimulation; ceci est illustré par l'étude des propriétés d'un opérateur d'aplatissement.

Mots-clés : Méthodes formelles, Systèmes distribués, Sémantique comportementale, Systèmes de transition structurés, Bisimulation

1 Introduction

This paper contributes to the study of formal modeling languages. In previous research, we defined pNets (Parameterised Networks of Synchronised Automata), an intermediate language targeted towards the behavioural specification of distributed systems. pNets [6] have been used to provide a behavioural specification formalism in different contexts. A (closed) pNet model can be used as the semantics of a program or a system; while open pNets define the semantics of composition operators in specification or programming languages. The pNet formalism is particularly powerful for expressing the behaviour of distributed objects [10] and distributed components [6], in particular distributed systems using queues, futures, and one-to-many communications [1]. In our previous work, we heavily used pNets in our verification environment, called Vercors.[1] It is a platform that assists the programmer in the specification and the verification of his/her component-based application. From the specification, we generate a pNet representing the application behaviour; from this behavioural model, we are able to verify properties ranging from absence of deadlocks, to reachability of some actions, and "any" temporal logic property (including safety and liveness) specific to the application. In order to do this, we choose a finite instantiation domain for the parameters of the pNets, and generate a tree of finite labelled transition systems on which we can prove the properties of the application by hierarchical verification, using state-of-the art model-checkers. The long history of research we conducted towards the fully automatised behavioural verification of distributed objects and components has been based on the pNets formalism. The numerous contributions[1, 8, 7, 10, 11] we made in the domain somehow prove the effectiveness of the formalism, but we never studied formally the theoretical foundation of pNets, or their expressiveness. In this report, we will not show any formal expressiveness result (such as an equivalence with a full encoding of CCS or other calculi). Instead we chose to demonstrate the flexibility of the model with a range of examples from various calculi, and application domains, into pNets.

Positioning. Our middle-term target is a generic framework for modeling the behaviour of distributed software systems, with synchronous and asynchronous execution and communication. We do not want to build yet another syntactic language or calculus, but rather provide a semantic level framework, based on (value-passing) transition systems, with a much richer structure.

When building such a framework, we could be tempted to address some features of the *dynamic* calculi, including dynamic discovery of new objects, channel passing *a la* π-calculus, or *ensembles* theories [5]. In pNets, we chose a different approach: we restricted dynamic aspects to the use of parameters; those parameters can however impact the states of the processes, the process topology, but also the communication patterns (see Section 3.4). Expressiveness is lower than in π-calculus as no name passing is allowed, but still the model is more dynamic than finite and static topologies. Bigraph models of Robin Milner [18] for ubiquitous systems are kinds of general models focusing on describing location and links, whereas pNets focus more on the synchronisation of processes. Bigraphs can encode most dynamic calculi, like e.g., π-calculus, and we consider it too expressive for our purpose. The strength of the pNets resides in the

[1] http://www-sop.inria.fr/oasis/index.php?page=vercors

expression of parametric architectures which are more restrictive than fully dynamic features, but easier to verify and still very expressive. Indeed, those architectures are expressive enough to model cloud-oriented applications with dynamic reconfiguration and group communication, but somehow static enough to allow for model-checking finite abstractions of their models. Our challenge is to find the best compromise between a model flexible enough to encode parametric and dynamic architectures, while allowing automatic model-checking, modulo well-defined data abstractions.

Inspiration for this model clearly comes from early work by Arnold and Nivat on synchronisation vectors [3, 4], on top of which we have constructed our pNets. But it is also very close to R. de Simone's work on Meije-SCCS[12], which provides a notion of "Architectural expressions" sharing many features with our open pNets. Meije-SCCS has a universality result with respect to calculi defined by some class of operational semantic rules, even if it was based on a more complex synchronisation mechanism (using auxiliary signals and a powerful action monoid) than the one offered by synchronisation vectors. Moreover, even if the semantics of our pNets has a shape similar to the one of Meije-SCCS, it is technically different, because the number of holes, and of subnets cooperating in a synchronisation, is unbounded in pNets, while it was finite in Meije-SCCS. As a consequence, we cannot reuse directly the congruence result of [12], and we will show our own result here.

Contributions. This paper proposes a first step in the theoretical foundation for pNets. After providing a concise and precise definition of the core of pNets, we express different coordination mechanisms that illustrate their expressive capacities. We first show that it is easy to express in pNets many constructs of classical value-passing algebras, including CCS and Lotos; pNets handle e.g., the usual binding mechanism within CCS communication, as well as the *gate negotiation* semantics of Lotos. Then we show that pNets allow the expression of more advanced synchronisation patterns. In particular, we show how to express in a simple and natural manner broadcasting interactions, but also indexing, i.e., using an index given in a communication parameter to decide which process should receive the communication, or mechanisms for synchronisation barriers.

Then we define a strong (early) bisimulation equivalence for closed pNets, and prove that it is a congruence for pNet contexts. This bisimulation is illustrated by two applications: one inspired from value-passing CCS, the other is a generic proof on an operator flattening a pNet structure.

This paper is organised as follows. Section 2 provides a formal definition of pNets, and of their operational semantics. Section 3 illustrates the expressiveness of pNets by providing examples expressing either classical constructs of process calculi, or more advanced coordination mechanisms. Section 4 proposes a strong bisimulation theory for pNets. Finally Section 5 gives an overview of related works and Section 6 concludes this paper.

2 Parameterised Networks (pNets): definition

In this first section we define pNets and the notations we will use in this paper; we conclude this section by providing an operational semantics for pNets.

Notations. We extensively use indexed structures over some countable indexed sets, which are equivalent to mapping over the countable set. $a_i^{i\in I}$, or equivalently $(i{\mapsto}a_i)^{i\in I}$ denotes a family of elements a_i indexed over the set I. Consequently, $a_i^{i\in I}$ defines both I the set over which the family is indexed (called *range*), and a_i the elements of the family. E.g., $a^{i\in\{3\}}$ is the mapping with a single entry a at index 3; it will also be denoted $(3{\mapsto}a)$ in the following. When this is not ambiguous, we shall use notations for sets, and typically write "indexed set over I" when formally we should speak of multisets, and write $x \in a_i^{i\in I}$ to mean $\exists i \in I.\, x = a_i$. An empty family is denoted $\emptyset$. $\uplus$ is a disjoint union operator on indexed sets: it requires that the two sets are indexed over disjoint sets. The elements of the union are accessed by using an index of one of the two families. When convenient, we will use $\bar{a}$ to abbreviate for $a_i^{i\in I}$, for some I.

Term algebra. Our models rely on the notion of parameterised actions. We leave unspecified the constructors of the algebra that will allow building actions and expressions. We denote ¶ the set of parameters (or variables) [2]. We denote by Σ the signature of those constructors and by $\mathcal{T}_¶$ the term algebra of Σ over the set of variables ¶. We suppose that we are able to distinguish inside $\mathcal{T}_¶$ a set of *action terms* denoted $\mathcal{A}_¶$ (*parameterised actions*), a set of *expression terms* denoted $\mathcal{E}_¶$, and, among expressions, a set of *Boolean expressions* (guards) denoted $\mathcal{B}_¶$ (with: $\mathcal{E}_P \cap \mathcal{A}_P = \emptyset \wedge \mathcal{B}_¶ \subseteq \mathcal{E}_¶ \wedge \mathcal{A}_P \cup \mathcal{E}_¶ \subseteq \mathcal{T}_¶$). For each term $t \in \mathcal{T}_¶$ we define $fv(t)$ the set of free variables of t. For $\alpha \in \mathcal{A}_¶$, $iv(\alpha)$ returns a subset of $fv(\alpha)$ which are the input variables of α, i.e., the variables newly defined by reception of their value during the action α.

pNets can be used with any term algebra. As a consequence our examples will show different notations for actions inspired from different litterature sources, like classical input-output interactions with parameters, *à la* value-passing CCS [17]. These will be written $?a(x_1, ..., x_n)$ for inputs, $!a(v_1, .., v_n)$ for outputs, or $a(v_1, .., v_n)$ and τ for synchronised actions; in that case we have $fv(?a(x)) = iv(?a(x)) = \{x\}$, and for a value v $iv(!a(v)) = \emptyset$. In other cases we use Lotos-style offers, with specific δ and $accept(x)$ action terms, or Meije-SCCS action monoids, like in $a.b$, $a^{f(n)}$. Naturally the expressiveness of the synchronisation constructs will depend significantly on the action algebra.

We allow countable indexed sets to depend upon variables, and denote $\mathcal{I}_¶$ the set of indexed sets with a range depending on variables of ¶. For example, simple intervals of the form $[1..n]$, for some $n \in$ ¶, represents a set of parameterised and variable size; the instantiation of such a set, i.e., the value of the set when the value of n is known, will unsurprisingly give the expected result. The set of expressible parameterised sets and the decidability of some results depend on the term algebra. For simplicity and at first, the reader can consider that all the parameterised sets are intervals where bounds are linear expression of the parameters, typically of the form $[1..m + 2.\, n]$. Those correspond to the parameterised intervals we used in our practical use cases; for these intervals the interpretation of the parameter valuation, including dynamic change of parameter value, raises no particular difficulty. In that case, variables in ¶ will be given a value that is a natural number. We suppose the existence of an inclusion relationship $\subseteq$ over $\mathcal{I}_¶$, with the natural guarantee that this operation ensures set inclusion when one replaces variables by

[2] We will use bothequivalently the term "variable", as usual in term algebras, and the word "parameter" that is more usual for "parameterised systems", to denote elements of ¶

their values.

2.1 The pNets Model

The first comprehensive definition of pNets was published in [6]. The definition of pNets provided here is a simplified[3] version that is convenient for providing a concise formal definition of pNets and to reason about them. pNets are networks of processes: they provide a hierarchical structure to organise processes. At the leaves of the structure we have pLTSs (parameterised labelled transition systems) described in Definition 2.1. Definition 2.2 describes the hierarchical composition. pNets are parameterised by a set ¶ of variables, used both in value-passing and in the processes topology. We suppose the set ¶ is defined globally, but it could also be defined locally in each pNet.

A pLTS is a labelled transition system with variables that can be wrote and read by the actions performed in the transitions; a pLTS can have guards and assignment of variables on transitions. Variables can be manipulated, defined, or accessed inside states, actions, guards, and assignments.

Definition 2.1 (pLTS)**.** *A pLTS is a tuple* $pLTS \triangleq \langle\!\langle S, s_0, \rightarrow \rangle\!\rangle$ *where:*

- S *is a set of states.*
- $s_0 \in S$ *is the initial state.*
- $\rightarrow \subseteq S \times T \times S$ *is the transition relation. Where*

$$T = \{\langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J}\rangle | \alpha \in \mathcal{A}_{\P} \wedge e_b \in \mathcal{B}_{\P} \wedge x_j \in \P \wedge e_j \in \mathcal{E}_{\P}\}$$

α are actions, e_b are guards, and variables in $iv(\alpha)$ are assigned when the transition is performed (as the "input variables" in traditional value-passing calculi). There may be also side-effects on variables, these will be specified by the additional assignments.

Note that we make no assumption on finiteness of S or of branching in $\rightarrow$. A *sort* is a set of actions. It can be viewed as the signature of the pNet. In the case of a pLTS, it is the set of labels appearing on the transitions:

$$\mathrm{Sort}(\langle\!\langle S, s_0, \rightarrow \rangle\!\rangle) = \{\alpha | \exists s, s', e_b, e_j^{j\in J}, t, x_j^{j\in J}.(s,t,s') \in \rightarrow \wedge\ t = \langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J}\rangle\}$$

pNets are constructors for hierarchical behavioural structures: a pNet is formed of other pNets, or pLTSs at the bottom of the hierarchy tree. A composite pNet consists of a set of subnets, each exposing a set of actions according to its sort. The synchronisation between a global action of the pNet and actions of the subnets is given by *synchronisation vectors* of the form $SV_k = \alpha_l^{l\in L_k} \rightarrow \alpha'_k$: a synchronisation vector synchronises one or several actions of subnets, and exposes a single resulting global action (α'_k). A pNet can either compose sub-pNets given explicitly, or be used as an operator accepting other pNets as parameters. Placeholders for the pNets that will be provided later are called *holes*, that play a role similar to process variables in usual process algebras. Actions synchronised in synchronisation vectors are both actions performed by sub-pNets and by holes.

[3] The complete pNet model includes elements that are convenient for behavioural specification and model-checking like request queues and the possibility to fill a single hole with a family of pNets. Those elements are not necessary for the formal study we present here, expressiveness is indeed the same.

Definition 2.2 (pNets). *A pNet is a hierarchical structure where leaves are pLTSs:*
$pNet \triangleq pLTS \mid \langle\!\langle pNet_i^{i\in I}, \mathcal{S}o_j^{j\in J}, SV_k^{k\in K}\rangle\!\rangle$ *where*

- $pNet_i^{i\in I}$ *is the family of sub-pNets where* $I \in \mathcal{I}_\P$ *is the set over which sub-pNets are indexed.*
- $\mathcal{H} \in \mathcal{I}_\P$ *is the set over which holes are indexed, each hole* h *is of sort* $\mathcal{S}o_h$. I *and* $\mathcal{H}$ *must be* disjoint*:* $I \cap \mathcal{H} = \emptyset$
- $SV_k^{k\in K}$ *is a set of synchronisation vectors* ($K \in \mathcal{I}_\P$). $\forall k \in K, SV_k = \alpha_{lk}^{l\in L_k} \rightarrow \alpha'_k$ *where* $\alpha'_k \in \mathcal{A}_\P$, $L_k \in \mathcal{I}_\P$, $\emptyset \subset L_k \subseteq I \uplus \mathcal{H}$, $\forall l \in L_k \cap I.\, \alpha_{lk} \in \mathrm{Sort}(pNet_l)$, *and* $\forall l \in L_k \cap \mathcal{H}.\, \alpha_{lk} \in \mathcal{S}o_l$. *We denote* $\mathrm{Label}(SV_k) = \alpha'_k$.
- *The set of holes of the sub-pNets, and of local holes of the pNet are all disjoint:*

$$\forall i \in I.\ \mathrm{Holes}(pNet_i) \cap \mathcal{H} = \emptyset$$
$$\forall i_1, i_2 \in I.\, i_1 \neq i_2 \Rightarrow \mathrm{Holes}(pNet_{i_1}) \cap \mathrm{Holes}(pNet_{i_2}) = \emptyset$$

The preceding definition relies on the two functions defined below.

Definition 2.3 (Sorts). *The sort of a pNet is the set of actions that a pNet can perform:*

$$\mathrm{Sort}(\langle\!\langle \overline{pNet}, \overline{\mathcal{S}o}, SV_k^{k\in K}\rangle\!\rangle) = \{Label(SV_k) | k \in K\}$$

Definition 2.4 (Holes). *The set of holes of a pNet is defined inductively as follows:*

$$\mathrm{Holes}(\langle\!\langle S, s_0, \rightarrow\rangle\!\rangle) = \emptyset \qquad \mathrm{Holes}(\langle\!\langle pNet_i^{i\in I}, \mathcal{S}o_h^{h\in\mathcal{H}}, \overline{SV}\rangle\!\rangle) = \mathcal{H} \cup \bigcup_{i\in I} \mathrm{Holes}(pNet_i)$$

Vocabulary. A *subnet* is either a sub-pNet or a hole.

A pNet Q is *closed* if it has no hole: $\mathrm{Holes}(Q) = \emptyset$; else it is said to be *open*.

When $I \cup \mathcal{H} = [0..n]$ we denote synchronisation vectors: $< \alpha_1, .., \alpha_n > \!\!\rightarrowtail \alpha$, and elements not taking part in the synchronisation are denoted $-$ as in: $< -, -, \alpha, -, - > \!\!\rightarrowtail \alpha$.

Definition 2.5 (pNet composition). *An open pNet:* $pNet = \langle\!\langle pNet_i^{i\in I}, \mathcal{S}o_h^{h\in\mathcal{H}}, \overline{SV}\rangle\!\rangle$ *can be (partially) filled by providing a family of pNets* $(pNet'_l)^{l\in L}$ *of the right sort to fill its holes.*

Suppose $\forall l \in \mathcal{H} \cap L.\ \mathrm{Sort}(pNet'_l) \subseteq \mathcal{S}o_h$, *then*[4]*:*

$$pNet\left[(pNet'_l)^{l\in L}\right] = \langle\!\langle \left(pNet_i\left[(pNet'_l)^{l\in L}\right]\right)^{i\in I} \uplus (pNet'_l)^{l\in J\cap L}, \mathcal{S}o_h^{h\in\mathcal{H}\setminus L}, \overline{SV}\rangle\!\rangle$$

Note that L can be larger than $\mathcal{H}$. In such a case, the elements of $(pNet'_l)^{l\in L\setminus\mathcal{H}}$ can fill holes at an arbitrary depth inside the tree structure of *pNet*. Also, if *pNet* is closed, then $pNet[\overline{pNet'}] = pNet$.

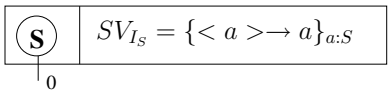
A trivial example. Let us first define a trivial open pNet that can encapsulate another pNet and transmit its communications/synchronisation actions unchanged. We also use this first example to define a graphical format for the pNets presentation.

Example 2.6 (Identity pNet). For a sort S, an identity *pNet*, I_S, is defined as:

$$I_S := \langle\!\langle \emptyset, (0 \mapsto S), \{(0 \mapsto a) \rightarrow a | a \in S\}\rangle\!\rangle$$

The corresponding graphical definition is:

[4] else the composition is undefined



This I_S pNet has a single hole, indexed by 0, of sort S. For any action of S, it transmits this action between the external interface of I_S (to/from processes in its environment), and the pNet that fills the hole. Overall, the behaviour of this pNet is that, for any pNet Q, $I_S[Q]$ behaves as Q.

For the examples in this paper, we use a graphical representation of pNets, certainly easier to read in most cases than the full mathematical notation. A pNet is represented by a box with subnets represented as circles on the left of the box. When a subnet is a hole, the sort is given in the circle and there is a line leading to nothing, decorated with the index below the circle. If the subnet is given, the line is connected to a pNet (see e.g. example 3.2 in page 9). On the right part we put the synchronisation vectors, often as a reference to a term outside the box.

2.2 An operational semantics for pNets

This section provides an operational semantics for the pNet model; it is based on a valuation domain for the variables of the pNet, that can be finite or infinite. The operational semantics is defined for *closed* pNets only; the semantics for a pNet with a hole is more complicated, it should be a function of the pNet put in the hole. We suppose the existence of a unique *valuation domain* $\mathcal{D}$. This domain can be a countable instantiation domain for each variable. We require that for a boolean expression e, if all the variables of e are given a value in $\mathcal{D}$ it is possible to decide whether e is true or false. Similarly, it must be possible to decide whether two expressions have the same value (e.g., when two action labels are the same).

We let $\phi = \{x_j \rightarrow V_j | j \in J\}$ be a valuation function, mapping x_j, which range over all variables of the considered pNet, to $V_j \in \mathcal{D}$, values. For a term $t \in \mathcal{T}_{\P}$, $t\phi \in \mathcal{D}$ is the value of the term obtained by replacing each variable by their values given by ϕ. A valuation can also be applied to indexed sets. In all cases, the variables are replaced by their value and the new expression is evaluated. We define $\hat{\Phi}$ as the set of all possible valuations. We choose Φ a subset of $\hat{\Phi}$ the set of valuations that can be explored: only valuation functions such that $\phi \in \Phi$ are considered. This allows us to restrict the valuation domain and, for example, consider only finite valuations or letting some variables range only on a small domain. While in $\hat{\Phi}$ the valuation domain is the same for all variables, in Φ each variable can range over a different sub-domain. We define an update operator $+$ on valuations, where $\phi_1 + \phi_2$ replaces some of the values defined in ϕ_1 by the ones in ϕ_2; ϕ_2 might also define new entries:
$\{x_j \rightarrow V_j | j \in J\} + \{x'_j \rightarrow V'_j | j \in J'\} = \{x'_j \rightarrow V'_j | j \in J'\} \cup \{x_j \rightarrow V_j | j \in J \setminus J'\}$

We suppose that variable names are unique but, as variables are used locally to each pNet/pLTS, it would also be possible to use qualified names to avoid collision. The semantics of a pNet is denoted $\|-\|_\Phi$, which is a LTS[5]. It relies on a set of states for closed pNets $S_\Phi(-)$ that are hierarchical composition of states reflecting the structure of the pNet. Then, $[\![-]\!]_\Phi$ defines a set of transitions between such states.

[5] A (classical) LTS is indeed a pLTS with no parameter

Definition 2.7 (Operational semantics of closed pNets)**.** *Let $\phi_0 \in \Phi$ be an initial valuation associating a value to each variable of ¶. The semantics of pNet is a LTS $\|pNet\|_\Phi = \langle\!\langle S_\Phi(pNet), S_0(pNet), \rightarrow \rangle\!\rangle$ where:*

- *The set of states of $\|pNet\|_\Phi$, denoted $S_\Phi(pNet)$ is defined as follows*[6]*:*

$$S_\Phi(\langle\!\langle S, s_0, \rightarrow \rangle\!\rangle) = \{(s, \phi) | s \in S \ \wedge \ \phi \in \Phi\}$$
$$S_\Phi(\langle\!\langle pNet_i^{i\in I}, \emptyset, SV_k^{k\in K} \rangle\!\rangle) = \{\lhd s_i^{i\in I\phi} \rhd \mid \phi \in \Phi \wedge \forall i \in I\phi.\, s_i \in S_\Phi(pNet_i)\}$$

- *The initial state $S_0(pNet)$ is the composition of initial states:*

$$S_0(\langle\!\langle S, s_0, \rightarrow \rangle\!\rangle) = (s_0, \phi_0) \qquad S_0(\langle\!\langle pNet_i^{i\in I}, \emptyset, SV_k^{k\in K} \rangle\!\rangle) = \lhd S_0(pNet_i) \rhd^{i\in I\phi_0}$$

- *labels are $\{\alpha\phi | \alpha \in \mathrm{Sort}(pNet) \wedge \phi \in \Phi\}$;*
- *and transitions are defined as $[\![pNet]\!]_\Phi$, the smallest set of transitions satisfying the rules below.*

$$\frac{\phi, \phi', \phi'' \in \Phi \qquad s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J} \rangle} s' \in \rightarrow \qquad \phi' = \phi + \{x_i' \rightarrow V_i | x_i' \in iv(\alpha) \wedge V_i \in \mathcal{D}\} \qquad e_b\phi' = True \qquad \phi'' = \phi' + \{x_j \rightarrow e_j\phi' | j \in J\}}{(s, \phi) \xrightarrow{\alpha\phi'} (s', \phi'') \in [\![\langle\!\langle S, s_0, \rightarrow \rangle\!\rangle]\!]_\Phi} \ \textbf{Tr1}$$

$$\frac{\phi, \phi' \in \Phi \qquad \alpha_l^{l\in L} \rightarrow \alpha \in \overline{SV} \qquad \forall i \in I\phi \setminus L\phi.\, s_i' = s_i \qquad \forall l \in L\phi.\, \phi_l \in \Phi \wedge s_l \xrightarrow{\alpha_l\phi_l} s_l' \in [\![pNet_l]\!]_\Phi \qquad \phi' = \phi + \uplus_{l\in L\phi}\phi_l}{\lhd s_i^{i\in I\phi} \rhd \xrightarrow{\alpha\phi'} \lhd s_i'^{i\in I\phi'} \rhd \in [\![\langle\!\langle pNet_i^{i\in I}, \emptyset, \overline{SV} \rangle\!\rangle]\!]_\Phi} \ \textbf{Tr2}$$

In these rules, any variable can be assigned any value in the corresponding data domain. Then the effects of transitions (reception of valeus, effects of assignments) will restrict the possible values in subsequent states of the pNet, while constraints introduced by the synchronisation vectors, in rule **Tr2**, will restrict the set of possible valuations.

Note that only states of pLTSs are associated with a valuation, but states of the pNets still use the valuation to decide (expand) the set of sub-pNets embedded in the pNet. Rule **Tr1** deals with transitions between states of the pLTS; only input variables and assigned variables are allowed to change value in the valuation; the resulting valuation is obtained by the successive updates from the input variable assignments, and the explicit assignments from the transition. Remark that the predicate in a transition is evaluated in a valuation that includes the values carried by the communication action, allowing for expressing non-local decisions as in Lotos *gate negotiation*. Note also that there is *a priori* no predefined direction for the flow of data because it is dependent on the chosen term algebra.

Rule **Tr2** deals with transitions between states of the pNet; A given synchronisation is picked, it involves the set $L \subseteq I$ of sub-pNets. The resulting valuation is obtained by the combination of all updates happening in the sub-pNets involved in the synchronisation ($\phi \uplus \{\phi_l\}^{l\in L\phi}$ in the rule); sub-pNets not involved in the transition keep the same state.

The fact that each variable has a (finite or infinite) complete instantiation ensures that the semantics is a (possibly infinite) fully instantiated LTS that has no more variable. This condition

[6] We use the $\lhd \ldots \rhd$ notation to easily identify states of the operational semantics in the rest of the paper.

is crucial for the decidability of strong bisimilarity presented in Section 4. If we choose a finite domain for each variable and if each pLTS has a finite set of states, the semantics of the pNet will be a finite LTS that can be used in a finite-state model-checker. The semantics defined above is illustrated in Section 3.2, with an example of a closed pNet encoding a n-places buffer. Giving a denotational semantics for open pNets is out of the scope of this paper, however, an open pNet can be seen as a function that accepts pNets to fill its hole. Once all the holes are filled, we can use the semantics of closed pNets defined above.

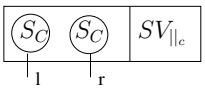
3 Examples and Expressiveness

This section illustrates the expressiveness of pNets by exhibiting pNets for several classical constructs of process calculi. These pNets "behave" similarly to the original construct: the reachable states and traces simulate faithfully the original construct. Proving richer equivalence between the modelled construct and the pNet or providing a general results on expressiveness of pNets with other existing models is out of the scope of this paper.

3.1 Constructions from classical (value-passing) algebras

We first focus on value passing algebras. The first example models the parallel operator of CCS, synchronising input and output actions, or transmitting those actions unchanged to the outside.

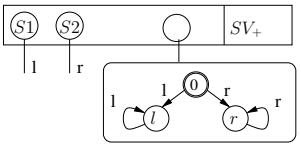
Example 3.1 (CCS-like Synchronisation)**.** Let C be a set of channel names, V a set of values. We denote S_C the sort: $S_C = \{\tau\} \cup \{?a(x) | a \in C,\, x \in \P\} \cup \{!a(v) | a \in C,\, v \in V\}$. The pNet of the synchronisation operator, $||_C$, is defined as:



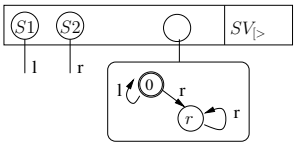
$$
\begin{aligned}
SV_{||_c} = {} & \{<!a(x), ?a(x) > \rightarrow \tau\}_{a:C,x:P,xfresh} \\
& \cup \{<?a(x), !a(x) > \rightarrow \tau\}_{a:C,x:P,xfresh} \\
& \cup \{< b, - > \rightarrow b\}_{b:S_C} \\
& \cup \{< -, b > \rightarrow b\}_{b:S_C}
\end{aligned}
$$

There are two holes indexed l and r (synchronisation vectors are written taking $l = 0$ and $r = 1$). There are four families of synchronisation vectors: the first two sets of vectors synchronise input and output actions from the process to the right (resp. left) to the one to the left (resp. right), it is visible as a τ action; the two last sets allow any action of a process to be visible at the level of the parallel operator, and potentially to be synchronised by another parallel operator. The two first vectors necessitate two fresh variables for each channel name and for each occurrence of the parallel operator. Value passing is obtained by an adequate definition of the valuation domains of the free variables: for each fresh variable x introduced by the synchronisation vector for channel c, the valuation domain should include the set of values that can be transmitted over the channel c. This set can be any over-approximation, and can be refined by typing on channels. Note that input and output actions have a symmetric role in the synchronisation vector, they will be distinguished in the way they can appear in pLTSs, and in the way they are handled by rule **Tr1**.

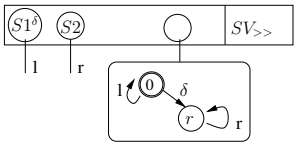
Example 3.2 (Choice, Disruption, and Sequence)**.** Next we show how to encode various artefacts for combining 2 processes. These are a classical non-deterministic choice as in CCS (here denoted "+"), a disruption operator similar to the Lotos *Disable* ([>) , and a sequencing operator with value passing similar to the Lotos *Enable* (>>).



$SV_+ = \{< a, -, l > \rightarrow a\}_{a:S1}$
$\cup \{< -, b, r > \rightarrow b\}_{b:S2}\}$



$SV_{[>} = \{< a, -, l > \rightarrow a\}_{a:S1}$
$\cup \{< -, b, r > \rightarrow b\}_{b:S2}\}$



$SV_{>>} = \{< a, -, l > \rightarrow a\}_{a:S1}$
$\cup \{< \delta(x), accept(x), \delta > \rightarrow \tau\}\}$
$\cup \{< -, b, r > \rightarrow b\}_{b:S2}\}$

These three pNets have a similar structure with two holes. The left hole, indexed l, has sort $S1$, the right hole has sort $S2$. The third subnet, contains an LTS encoding their control part. These control parts, and the synchronisation vectors, are the elements defining their specific behaviour.

For the *Choice* pNet, the first action of one of the holes decides which branch of the LTS is activated; all subsequent actions will be from the same side. Concerning the *Disable* pNet, the left sub-process evolves alone for any number of steps; but at any time the right sub-process can interrupt it and from this point keep the control. Finally the *Enable* pNet is a sequencing operator with data transmission: the left sub-process emits a specific exit signal δ (we denote $S1^\delta = S1 \cup \{\delta\}$ its sort), transmitting data values to the right sub-process through an initial "accept" action.

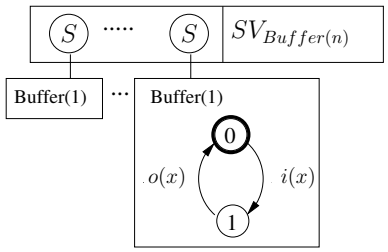
Those examples rely on some pLTSs dedicated to control of the rest of the pNet: they encode the state of the assembly and enable different transitions depending on this state; we call these pLTSs *controllers*. Controllers are typical artefacts managing an assembly of processes, with dynamic evolution of the control part. A similar idea has been used for giving a compositional behaviour semantics to open process expressions in Lotos [15].

3.2 A closed pNet system: the n-places Buffer

Now we give an example of a closed pNet representing a well-known toy example: an n-places buffer. We also use this example to illustrate the operational semantics of closed pNets.

The standard way to build a n-places buffer in CCS is to define a 1-place buffer as a 2 states process: $B = ?i(x).!o(x).B$, and to compose n instances of B using parallel, renaming, and restriction operators: $(B[a1/o]||B[a1/i, a2/o]||...)\backslash_{a_1,a_2,...}$. We could have used a direct encoding of each of the CCS operators using open pNets, and build a (rather complicated) pNet system to express the n-places buffer. But we prefer to show the expressiveness of pNets by building the full (parameterised) system using a single pNet node, synchronising all one-place buffers in one step.

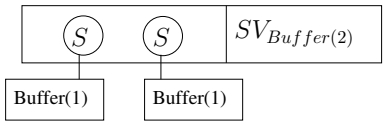
Example 3.3. Using pNets, we express such a construct using a parameterised pNet with n subnets, and as many synchronisation vectors expressing binary synchronisation between the subnets:



$S = \{?i(x), !o(x)\}$
$SV_{Buffer(n)} = \{<?i(x), -, ..., - > \rightarrow ?i(x) >,$
$< -, ..., -, !o(x), ?i(x), -, ..., - > \rightarrow \tau >,$
$< -, ..., -, !o(x) > \rightarrow !o(x) >\}$

Computing the operational semantics of the n-places Buffer Computing the possible transitions of such a pNet involves using the rules of the operational semantics from Section 2.2, building all possible proof trees combining transitions of the pLTSs at the leaves of our system, with all adequate synchronisation vectors of the pNet node.

For simplicity of notations, consider a Buffer of size 2:



$S = \{?i(x), !o(x)\}$
$SV_{Buffer(2)} = \{<?i(x), - > \rightarrow ?i(x) >,$
$<!o(x), ?i(x) > \rightarrow \tau >,$
$< -, !o(x) > \rightarrow !o(x) >\}$

We start with a (global) parameter set $P = \{x_1, x_2\}$, and an initial valuation ϕ_0. Chose a valuation Φ containing ϕ_0, and all possible valuations of the variables in P. The initial state of $[\![Buffer(2)]\!]_\Phi$ is $\lhd(s_0, \phi_0), (s_0, \phi_0)\rhd$.

The only way to build a transition from this state is to use rule Tr1 with the first synchronisation vector $<?i(x_1), - > \rightarrow ?i(x_1)$ in which only the first sub-pNet of Buffer(2) is working, and to use Tr2 to prove the initial transition of this pLTS:

$$\frac{s_0 \xrightarrow{?i(x_1)\phi_1} s_1 \in \rightarrow_{Buffer(1)} \qquad iv(?i(x_1)) = \{x_1\} \qquad v_1 \in \mathcal{D} \qquad \phi_1 = x_1 \rightarrow v_1}{(s_0, \phi_0) \xrightarrow{?i(x_1)\phi_1} (s_1, \phi_0 + \phi_1) \in [\![Buffer(1)]\!]_\Phi} \textbf{ Tr2}$$

$$\frac{\begin{array}{c}(s_0, \phi_0) \xrightarrow{?i(x_1)\phi_1} (s_1, \phi_0 + \phi_1) \\ \phi \in \Phi \qquad k = 1 \qquad SV_1 = <?i(x_1), - > \rightarrow ?i(x_1) \qquad \mathcal{H} = \{1\} \qquad \phi_1 = \{x_1 \rightarrow v_1\}\end{array}}{\lhd(s_0, \phi_0), (s_0, \phi_0)\rhd \xrightarrow{?i(x_1)\phi_1} \lhd(s_1, \phi_0 + \phi_1), (s_0, \phi_0)\rhd \in [\![Buffer(2)]\!]_\Phi} \textbf{ Tr1}$$

A similar construction leads to the full transition relation of $[\![Buffer(2)]\!]_\Phi$:

$$\begin{array}{ll}
\lhd(s_0, \phi_0), (s_0, \phi_0)\rhd \xrightarrow{?i(v_1)} \lhd(s_1, \phi'_1), (s_0, \phi'_1)\rhd & \phi'_1 = \phi_0 + \{x_1 \rightarrow v_1\} \\
\lhd(s_1, \phi'_1), (s_0, \phi'_1)\rhd \xrightarrow{\tau} \lhd(s_0, \phi'_2), (s_1, \phi'_2)\rhd & \phi'_2 = \phi'_1 + \{x_2 \rightarrow v_1\} \\
\lhd(s_0, \phi'_2), (s_1, \phi'_2)\rhd \xrightarrow{!o(v_1)} \lhd(s_0, \phi'_3), (s_0, \phi'_3)\rhd & \phi'_3 = \phi'_2 \\
\lhd(s_0, \phi'_2), (s_1, \phi'_2)\rhd \xrightarrow{?i(v_2)} \lhd(s_1, \phi'_4), (s_1, \phi'_4)\rhd & \phi'_4 = \phi'_2 + \{x_1 \rightarrow v_2\} \\
\lhd(s_1, \phi'_4), (s_1, \phi'_4)\rhd \xrightarrow{!o(v_2)} \lhd(s_1, \phi'_5), (s_0, \phi'_5)\rhd & \phi'_5 = \phi'_4 \\
...
\end{array}$$

If the domain of x is infinite, then this transition relation will also infinite, but all transitions are of the same form as one of the 5 above, with only the valuation values changing.

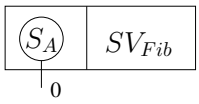
Naturally in our modeling platform we have specific syntax to express the parameterised structures and the instantiation. This is beyond the scope of this paper.

3.3 Richer action algebras

In this section we investigate action algebras with expressive constructs and their use in the synchronisation vectors of pNets, lifting the richness of these constructs at the level of processes.

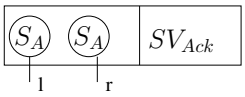
The two following examples are borrowed from Meije-SCCS [12], where the action algebra includes a simultaneity product, and has globally the structure of a commutative monoid.

Example 3.4. Fibonacci. Here we have a single hole with sort $S_A = \{a^n | a \in A, n \in \mathbb{N}\}$ with A a set of action names. Then the open pNet Fib transforms any action of the subnet by increasing its "intensity".



$$SV_{Fib} = \{< a^n > \rightarrow a^{fib(n)}\}_{a:A,n:Nat}$$

Example 3.5. Ackermann. The Ack pNet is similar, though it synthesises a signal of "intensity" $Ack(m,n)$, synchronising inputs of the same base action from 2 subnets.



$$SV_{Ack} = \{< a^m, a^n > \rightarrow a^{Ack(n,m)}\}_{a:A,n,m:Nat}$$

These two open pNets are examples of the richness of computations that can be expressed in synchronisation vectors, mixed with synchronisation capacities in the Ackermann case. Such mechanisms in the Meije-SCCS structure were proved decidable (though computationally expensive) using the theory of semi-linear sets. One could imagine further extensions using some satisfiability modulo theory (SMT), taking into account the properties of the data-types embedded in the action algebra; in that case decidability may have to be discussed.

3.4 Many-to-many Communications

pNets offer richer and higher-level synchronisation capacities than classical one-to-one communications. pNets are particularly convenient for expressing one-to-many and many-to-one communications that are frequently used in modern grid or cloud middleware. The examples of this section all have the same top-level structure, with a number of holes with identical sort S, but they differ in the way subnets are synchronised. We present here pNets modelling either one-to-many schemas (Unicast, Broadcast, and Scatter), or many-to-one synchronisation (Synchronisation Barrier).

We first present one-to-many communications. For each of the 3 variants, we build a pNet assembling a group of processes (subnets). A process communicating with this group synchronises with some of the subnets, and the argument of the communication (the "payload" of the message) can be distributed in different ways. In the Unicast pNet the payload is delivered to a single

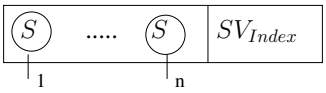
element of the group, only one subnet synchronises. When broadcasting, each element receives its own copy of the payload, while in the Scattering pNet each subnet receives a different piece of the payload. Typically, the pNet representing the group will then be assembled with some data provider *pNet* as $pNet||Group_{S,N}$ in which *Group* can be a unicast, broadcast or scatter pNet. The provider would be a process sending messages of type $!a(arg)$, in which arg is a value to be passed to each/all of the subnets in the group (for unicast and broadcast), or a vector of such values in the case of scatter, with the length of the vector equal to the length of the group.

In the 3 cases, elements of the groups (the holes) all have a sort containing reception events $?a \in S$, that carry some value ($iv(a) = \{v\}, v \in \P$).

Example 3.6 (Unicast)**.** A unicast pNet dispatches actions to a set of pNets that fill its holes, all of sort S. The resulting unicast pNet is of sort $\{?a(n) |\, ?a \in S \wedge n \in N\}$:

$$Unicast_{S,N} := \ \langle\!\langle \emptyset, (n \mapsto S)_n^{n \in N}, \{(n \mapsto a) \rightarrow a(n) |\, a \in S \wedge n \in N\} \ \rangle\!\rangle$$

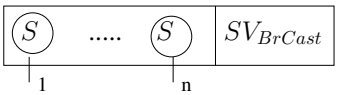
We provide below the graphical definition for this pNet when $N = [1..n]$.



$$SV_{Index} = \{< -, ..., a, ..., - > \rightarrow a(n)\}_{a:S,n:N}$$

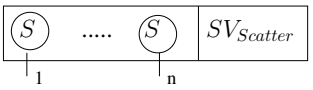
This pNet has card(N) holes, each of sort S. For each action of S it allows the synchronisation of one hole with the external world. The index of the hole is matched with the exported action. This way, it is possible to choose which subnet performs the action, or to allow any of them to perform the action, and to know which of them performed it.

Example 3.7 (Broadcast)**.** A broadcast pNet dispatches actions to a set of pNets that fill its holes. The holes are indexed over a set N. Each hole of the pNet has sort S, where S only contains reception actions, $?a \in S$, that intuitively carry some value ($iv(a) = \{v\}, v \in \P$). We build a broadcasting pNet of sort S:



$$SV_{BrCast} = \{<?a, ..., ?a, ..., ?a > \rightarrow ?a\}_{?a:S}$$

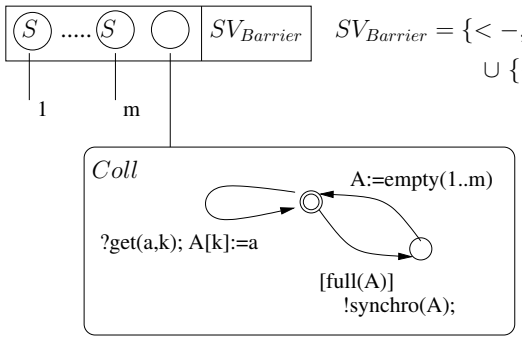
Example 3.8 (Scatter)**.** The Scattering pNet is similar to the Broadcast pNet, except that instead of duplicating the argument, it splits the argument in as many pieces as there are elements in the target group. The sort of the scattering pNet is S^N, i.e., vectors of terms, each subnet, indexed j, will receive its own piece of the global action term $?a$, denoted here $?a.j$.



$$SV_{Scatter} = \{<?a.1, ..., ?a.j, ..., ?a.n > \rightarrow ?a\}_{?a:S^N}$$

Reversely, we define a many-to-one communication acting as a synchronisation barrier. It is a synchronising pattern in an asynchronous system: it waits for events from a set of processes, and when all events have arrived it creates some sort of "global data" from data received, and sends it to an external process.

Example 3.9 (Synchronisation Barrier)**.** This pNet collects actions from a set of m pNets. Its holes, of sort S, are indexed over a set $N = [1..m]$. The pNet also contains a collecting process *Coll* that gathers individual 'a' messages from the holes, and stores them in an array (if any hole sends 'a' twice, the new value replaces the older one); when the array is full it transmits the resulting array as an argument of the 'synchro' global action. The sort of the SBarrier pNet is $\{?get(a,k)|a \in S \wedge k \in [1..m]\} \cup \{!synchro(A)|A \in S'\}$, where S' is the set of arrays of length m containing elements in S.



$$SV_{Barrier} = \{< -, ..., a, ..., -, ?get(a,k) > \rightarrow ?get(a,k)\}_{k:[1..m],a:S} \cup \{< -, ..., -, !synchro(A) > \rightarrow !synchro(A)\}_{A:S'}$$

This section has illustrated the expressiveness of pNets through examples. We have shown their capacity to express rich composition patterns mimicking both classical synchronisations, and modern many-to-many interactions.

4 Strong Bisimulation for pNets

In this section, we define a convenient notion of (strong) bisimulation for closed pNets, and prove that this equivalence is a congruence w.r.t. any kind of pNet context. We first define a context in the pNet formalism. Defining a context syntactically as one would do in usual process algebras is possible in our case. However, pNets already provide a way to define expressions with holes. A context pNet is a pNet C with a single hole, we index this hole by $\bullet$. To alleviate proofs, we distinguish contexts featuring a single hole at the top-level of the pNet structure, and call them *top1-contexts* from contexts having a single hole at any depth.

Definition 4.1 (top1-context)**.** *A top1-context of sort $\mathcal{S}o$ is a pNet C of the form $\langle\!\langle \overline{pNet}, (\bullet \mapsto \mathcal{S}o), \overline{SV} \rangle\!\rangle$ where $\overline{pNet}$ is a family of closed pNets.*

Definition 4.2 (1-context)**.** *A 1-context of sort $\mathcal{S}o$ is either:*

- *a top1-context pNet of sort $\mathcal{S}o$: $\langle\!\langle \overline{pNet}, (\bullet \mapsto \mathcal{S}o), \overline{SV} \rangle\!\rangle$,*
- *or a pNet $\langle\!\langle \overline{pNet}, \emptyset, \overline{SV} \rangle\!\rangle$, in which one (and only one) of the $pNet_i$ is a 1-context of sort $\mathcal{S}o$.*

Recall that the operation $pNet[\{P_i\}]$ fills some holes of $pNet$, with $\{P_i\}$ a family of pNets indexed with the indexes of the holes. Then, provided $\mathrm{Sort}(Q) \subseteq \mathcal{S}o$, $C[(\bullet \mapsto Q)]$ is the closed pNet where the hole is replaced by Q (e.g., $C[(\bullet \mapsto Q)] = \langle\!\langle \overline{pNet} \cup (\bullet \mapsto Q), \emptyset, \overline{SV} \rangle\!\rangle$ in the case of a top1-context).

4.1 Strong bisimulation

Strong bisimulation of pNets is relatively classical except that it relates instantiated states, and that the two bisimilar states of pNets can perform different transitions with different valuations provided the application of the two valuations on the two labels give the same result.

Definition 4.3 (Strong Bisimulation)**.** *Let P and Q be two closed pNets. A relation $R \subseteq S_\Phi(P) \times S_\Phi(Q)$ is a* strong simulation *if for each $(s,t) \in R$, for any valuation set $\phi \in \Phi$, whenever $s \xrightarrow{\alpha\phi} s' \in [\![P]\!]_\Phi$, then $\exists \phi' \in \Phi, \beta, t'$, such that $t \xrightarrow{\beta\phi'}_\Phi t' \in [\![Q]\!]_\Phi$ and $(s',t') \in R$ and $\alpha\phi = \beta\phi'$.*

A relation R is a strong bisimulation *if R and R^{-1} are strong simulations.*

Consider two closed pNets P and Q, two states $s \in S_\Phi(P)$ and $t \in S_\Phi(Q)$ are strongly bisimilar, *denoted by $(\!|s,P|\!) \sim (\!|t,Q|\!)$ iff $(s,t) \in R$, for some strong bisimulation $R \subseteq S_\Phi(P) \times S_\Phi(Q)$.*

Two closed pNets P and Q are strongly bisimilar, *denoted by $P \sim Q$, iff their initial states are strongly bisimilar: $(\!|S_0(P),P|\!) \sim (\!|S_0(Q),Q|\!)$*

Observe that the bisimilarity between pNets only implies that their reachable states are equivalent.

Proposition 4.4 (Strong Bisimulation is a congruence for top1-contexts)**.** *Let P and Q be two closed pNets and $s \in S_\Phi(P)$, $t \in S_\Phi(Q)$ two strongly bisimilar states $(\!|s,P|\!) \sim (\!|t,Q|\!)$.*

Then for any top1-context C of sort S, i.e., $C = \langle\!| pNet_i^{i\in I}, (\bullet\mapsto S), \overline{SV} |\!\rangle$, such that $\mathrm{Sort}(P) \cup \mathrm{Sort}(Q) \subseteq \mathcal{S}o$*, we can build two closed pNets $C[(\bullet\mapsto P)]$ and $C[(\bullet\mapsto Q)]$ ensuring that:*

For all s', t' of the form $s' = \lhd s_i^{i\in I\phi} \uplus (\bullet\mapsto s)\rhd \in S_\Phi(C[\bullet\mapsto P])$, $t' = \lhd s_i^{i\in I\phi} \uplus (\bullet\mapsto t)\rhd \in S_\Phi(C[\bullet\mapsto Q])$ with $\forall i \in I\phi.\, s_i \in S_\Phi(pNet_i)$; we have $(\!|s',P|\!) \sim (\!|t',Q|\!)$.

Consequently, provided $\mathrm{Sort}(P) \cup \mathrm{Sort}(Q) \subseteq \mathcal{S}o$*, for any top1-context C of sort $\mathcal{S}o$:*
$P \sim Q \Rightarrow C[(\bullet\mapsto P)] \sim C[(\bullet\mapsto Q)]$.

The proof (in Appendix 7.1) is done using classical bisimulation techniques and distinguishing synchronisation vectors involving actions of the hole from the others. We use Property 4.4 to show that strong bisimulation is a congruence, for 1-contexts where the hole can occur at arbitrary depth.

Theorem 4.5 (Strong bisimulation is a congruence)**.** *Let P and Q be two bisimilar closed pNets: $P \sim Q$. Then for any 1-context C of sort $\mathcal{S}o$, such that* $\mathrm{Sort}(P) \cup \mathrm{Sort}(Q) \subseteq \mathcal{S}o$*, we have $C[(\bullet\mapsto P)] \sim C[(\bullet\mapsto Q)]$.*

The proof follows from Proposition 4.4 by induction on the structure of contexts (see Appendix 7.2).

This result can be extended to n-ary contexts (by replacing one hole at a time), allowing to compose pNets in any possible way.

Example of properties. To illustrate the interplay between variable valuations, input variables in communication, and the strong bisimulation, we run here a small classical example,

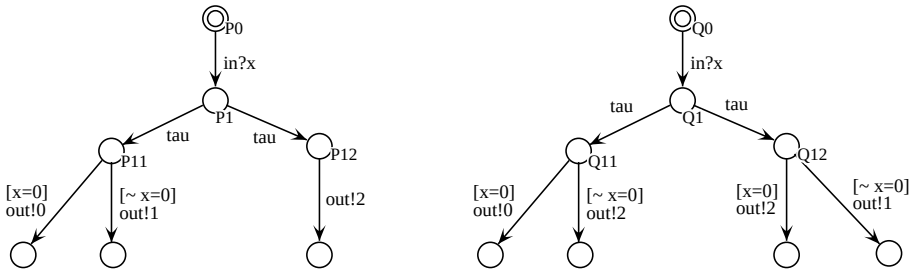


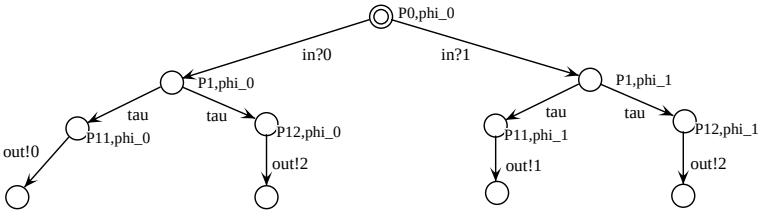
Figure 1: Process P and Process Q.

inspired from [14], where it was used to discuss about early, late, and symbolic equivalences in value-passing CCS. Let us prove that the two pLTSs P and Q in Figure 1 are strongly bisimilar, even if matching transitions in their semantics do not come from the same symbolic branch of the pLTS.

Computing the operational semantics of these two pLTS requires fixing a valuation set Φ. It is enough to use only two possible values for variable x, distinguishing the possible evaluations of the guard `[x=0]`. We use $\{0, 1\}$, corresponding to the most abstract domain for x preserving the semantics. Then we compute $\|P\|_\Phi$ and $\|Q\|_\Phi$ by unfolding different possible valuations and transitions of the process.

Then checking the equivalence of the two LTSs classically involves the building of equivalence classes of states, where each state includes an explicit valuation. This requires exhibiting pairs of equivalent states as in any classical bisimulation algorithm, only here we have to exhibit explicitly the valuations of variables.

The most possible abstract domain for the variable x preserving the semantics is $\{0, 1\}$, so we fix $\Phi = \{\phi_0, \phi_1\}$, with two possible valuations $\phi_0 = (x \to 0)$ and $\phi_1 = (x \to 1)$. Running rule **Tr1** on process P, choosing arbitrarily ϕ_0 as the initial valuation, unfolds the semantic LTS shown in Figure 2.



Bisimilar states:
$((P_0, \phi_0)\ (Q_0, \phi_0))$
$((P_1, \phi_0)\ (Q_1, \phi_0))$
$((P_1, \phi_1)\ (Q_1, \phi_1))$
$((P_{11}, \phi_0)\ (Q_{11}, \phi_0))$
$((P_{12}, \phi_0)\ (P_{12}, \phi_1))$
$((Q_{12}, \phi_0)\ (Q_{11}, \phi_1))$
$((P_{11}, \phi_1)\ (Q_{12}, \phi_1))$

Figure 2: Behaviour unfolding of process P, and classes of bisimilar states of P and Q

We leave the semantics of Q as an exercise for the reader. The last step is to prove equivalence of P and Q. This is a simple exercise, we only have to exhibit a bisimulation relation between the states of the semantic LTSs, being careful of transitions that do not come from the same

branches of the original pLTSs (see right part of Figure 2).

4.2 Application to Flattening Operator

Finally, this section presents a flattening operation for pNet contexts and proves strong bisimilarity. The purpose of this section is to illustrate further the interaction between bisimulation and pNets structure on an example that is useful both from a practical and from a theoretical point of view. The flattening operator can be used to flatten a pNet hierarchy, it is used when building a model for a large system (to reduce the state-space) and can also be used to ease compositional formal reasoning. We suppose here that the action algebras limits the parameters of the actions to be only variables. This restriction avoids us to have to deal with unification between action terms here; it is possible to add a pNet in the flattened structure dealing with the unification.

Definition 4.6 (Flattening)**.** *Let* $C' = \langle\!\langle \overline{pNet'}, (\bullet \mapsto \mathcal{S}o'), \overline{SV'} \rangle\!\rangle$*, and* $C = \langle\!\langle \overline{pNet}, (\bullet \mapsto \mathcal{S}o), \overline{SV} \rangle\!\rangle$ *with* $\mathrm{Sort}(C') \subseteq \mathcal{S}o$*; we define* $\mathrm{Flat}(C, C')$ *as follows:*

$$\mathrm{Flat}(C, C') = \langle\!\langle \overline{pNet} \uplus \overline{pNet'}, (\bullet \mapsto \mathcal{S}o'), \overline{SV''} \rangle\!\rangle \qquad \textit{where}$$

$$\begin{aligned}
\overline{SV''} \;=\; & \{(\alpha_l \varphi)^{l \in L \setminus \{\bullet\}} \uplus \beta_l^{l \in L'} \to \alpha' \mid \alpha_l^{l \in L} \to \alpha' \varphi \in \overline{SV} \wedge \beta_l^{l \in L'} \to \beta \in \overline{SV'} \\
& \quad \wedge \bullet \in L \wedge \alpha_\bullet \varphi = \beta \wedge \varphi = (x_n \to y_n)^{n \in N} \wedge \forall n \in N.\, (x_n \in fv(\alpha_\bullet) \wedge y_n \in fv(\beta))\} \\
& \cup \{(\alpha_l)^{l \in L} \to \alpha' \mid (\alpha_l)^{l \in L} \to \alpha' \in \overline{SV} \wedge \bullet \notin L\}
\end{aligned}$$

This definition uses the fact that action parameters can only be variables, and thus an action β triggered by a subnet can only match the action $\alpha_\bullet$ in the synchronisation vector if there exist φ that assigns to each free variable of $\alpha_\bullet$ a variable of β such that the two actions are the same. φ is then applied to all the actions α_i of the original synchronisation vector. Now we prove the most interesting property of the flattening: it preserves strong bisimulation:

Proposition 4.7 (Flattening and strong bisimulation)**.**

$$\forall P \textit{ closed } pNet, C[\bullet \mapsto C'[\bullet \mapsto P]] \sim Flat(C, C')[\bullet \mapsto P]$$

The full proof is in Appendix 7.3. This property has several consequences, as it shows that it is sufficient to prove congruence for the top-level contexts instead of checking holes occurring deeper in the pNet tree. We can then derive a Flatten operator that flattens a closed pNet P hierarchy, it produces a pNet composing only pLTSs as explained below.

Observe that the hole in the context can be indexed by an index other than $\bullet$. We define a function F_0 as follows: Let $P = \langle\!\langle pNet_i^{i \in I}, \emptyset, \overline{SV} \rangle\!\rangle$. Either P is flat (i.e. $\forall i \in I.\, pNet_i$ is a pLTS then $F_0(P) = P$); or $\exists i_0 \in I.\, pNet_{i_0} = \langle\!\langle {pNet'}_i^{i \in I'}, \emptyset, \overline{SV'} \rangle\!\rangle$, then let $C = \langle\!\langle pNet_i^{i \in I \setminus \{i_0\}}, (i_0 \mapsto S_0), \overline{SV} \rangle\!\rangle$. Take $i_1 \in I'$ and have $C' = \langle\!\langle pNet_i^{i \in I' \setminus \{i_1\}}, (i_1 \mapsto S_1), \overline{SV'} \rangle\!\rangle$. Then $P = C[i_0 \mapsto C'[i_1 \mapsto {pNet'}_{i_1}]] \sim Flat(C, C')[i_1 \mapsto {pNet'}_{i_1}] = F_0(P)$.[7] The recursive application of F_0 on P reaches a fixed-point if all the sub-pNets are indexed over finite set, in that case we call *Flatten*(P) this fixed-point and have *Flatten*$(P) \sim P$.

[7] To make F_0 a function, we need to choose i_0 and i_1 deterministically, which can be done by choosing an arbitrary order on the indexing sets without loss of generality.

On the practical side, Property 4.7 justifies some strategies used to optimize the state-space generation for hierarchical pNet structures before model-checking. More precisely, we often use a hierarchical approach for state-space generation, starting bottom-up with state-space (= LTS) of pLTSs at the leaves, then computing products for each level of the pNet tree, reducing each intermediate LTS using bisimulation. But this brute force approach happen to be expensive (in term of the size of intermediate LTSs) when some subsystems explode because of the lack of information about their execution contexts. In such cases it can be very useful to apply first flattening (removing some of the hierarchical levels of the pNet) before applying the hierarchical stat-space generation.

Concerning generic proofs, this operator can help in proving various properties, like associativity of the CCS operators $\|$ and $+$. Indeed, proving that $\mathit{Flatten}(P\|(Q\|R))$ is bisimilar to $\mathit{Flatten}((P\|Q)\|R)$ becomes trivial because the two flattened pNets are equal. The case of $+$ is less trivial because the flattened pNets are not the same for $(P+Q)+R$ and $P+(Q+R)$, proving their bisimilarity requires arguments similar to the direct study. On the contrary the enabling operator from Lotos (denoted $>>$) is not associative, and $\mathit{Flatten}(P >> (Q >> R))$ and $\mathit{Flatten}((P >> Q) >> R)$ are not the same and cannot be proved to be bisimilar in the general case.

Finally, we expect this operator to help us in the definition of more complex semantic notions to reason on pNets, e.g., a symbolic semantics for pNets, or some notion of barbed congruence. Indeed, thanks to flattening, it will be sufficient to define a semantics on a flat pNet only synchronising LTSs.

5 Related Works

Synchronisation vectors. As mentioned in the introduction, this research naturally inherits from the seminal work by Arnold and Nivat on synchronisation vectors [3, 4]. The theoretical expressiveness of (closed) pNets is the same, but their tree structure is essential to search for compositionality, and for the definition of open structures. Moreover, Arnold/Nivat synchronisation vectors did not include explicit data values; in our approach, this ability is very important for usability in verification tools.

Formats for SOS semantic rules. The closest work to this paper may be the efforts in the nineties [13] to define the tyft/tyxt format and its extensions. The expressive power of synchronisation vectors is similar to that of SOS rules with positive hypothesis and structured transition labels. Close to the limits of what pNets (and tyft/tytx) can express, you would find replication mechanisms: in pNets this is encoded as the dynamic activation of a new instance inside one (unbounded) parameterised hole. One significant difference is that pNets allow for synchronisation of an unbounded number of subnets, while in the tyft/tyxt format and its variants (to the best of our knowledge), this is not possible. We already had a similar distinction when comparing with the MEIJE-SCCS format in Section 1. Our approach with pNets is more constructive and pragmatic, as our main interests are for defining behavioural semantics for various programming languages in the parallel and distributed computation area, and we want our model, its expressiveness, and its limits to be compatible with an efficient implementation in a modeling and

verification environment.

Other semantic-level frameworks. In the family of "low-level" formalisms for distributed systems, we must cite Lotos-NT and BIP.

Lotos NT [16] is a low-level language designed as an efficient execution language for Lotos or E-Lotos, but also as an intermediate formalism for the model-checking of these languages. It was originally designed to express easily the complex synchronisation mechanisms of E-Lotos, but later used successfully to encode the Pi-calculus. Still it relies strongly on specific parallel operators; the direct multi-way and parameterised synchronisation vectors of pNets allow for a much more flexible and direct encoding of other mechanisms.

BIP [9] is a formal framework that allows building and analysing complex component-based systems. It applies to both synchronous (reactive) and asynchronous (distributed) systems by coordinating the behaviour of a set of primitive and heterogeneous components. A component's behaviour is described as a Petri net extended with data and functions, whereas coordination is described as interactions between components and scheduling policies between interactions. Even if the BIP framework allows powerful compositional reasoning on the system, it does not support the definition of parameterised components, nor does it allow (in its basic versions) explicit data transfer between components. On the other side, the priority functionality of BIP would require to use some kind of "negative premisses", that we cannot do in the current version of pNets.

One aspect we have not tried yet, but that seems to fit naturally with the pNet appraoch, is formalisms dedicated to the external coordination of components, as e.g. with the REO connectors language[2]. This would certainly be an interesting basis for studying integration of component description formalisms with exogeneous coordination or choregraphy.

6 Conclusion and Discussion

pNets have been used to provide a general behavioural specification formalism in different contexts, in particular it proved to be particularly powerful for expressing the behaviour of distributed objects and of distributed components.

In this paper, we illustrate the expressiveness of pNets by showing how they can be used to express a wide range of classical synchronisation patterns, including constructs of value-passing algebras. We have also exhibited more complex interaction and synchronisation patterns used for many-to-many communication schemes in distributed component-based applications. In that case, pNet parameters are used both as data values and as topology indexes.

We propose a behavioural semantics and a bisimulation theory, starting with a strong bisimulation relation for closed pNets. Both the semantics and the equivalence are based on all possible valuations, and can be thought of *early* versions. We prove that the strong bisimulation indeed is a congruence for all pNets contexts, and we provide a more constructive approach for combining pNets using a flattening operator.

Our next goal is to look for appropriate bisimulation relations for *open pNets*. Such equivalences should be considering hypotheses on the behaviour of holes, as was used in so-called

FH-bisimulations in [12], but a particular challenge will be to find more constructive equivalences that will give us efficient approaches to build proofs in a compositional (congruent) manner. Such compositional theories and proof methods are of course essential in the analysis of hierarchical component systems. Another promising perspective is to investigate the proof of "symbolic" properties that are valid independently of the parameter valuations and to identify conditions on data and indexes in which one can prove such properties, such an approach could rely on previous works on symbolic reasoning for value passing processes [14]. The properties of congruence and of the flattening operator are promising first steps toward a bisimulation theory for open pNets.

References

[1] R. Ameur-Boulifa, R. Halalai, L. Henrio, and E. Madelaine. Verifying safety of fault-tolerant distributed components. In *International Workshop on Formal Aspects of Component Software (FACS'11)*, Oslo, Sept 2011.

[2] F. Arbab. A channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14:329–366, 2004.

[3] A. Arnold. *Finite transition systems. Semantics of communicating sytems.* Prentice-Hall, 1994. ISBN 0-13-092990-5.

[4] A. Arnold. Nivat's processes and their synchronization. *Theor. Comput. Sci.*, 281(1-2):31–36, 2002.

[5] J. Barnat, N. Beneš, T. Bureš, I. Černá, J. Keznikl, and F. Plášil. Towards Verification of Ensemble-Based Component Systems. In *To appear in Formal Aspects of Component Software (FACS)*, LNCS. Springer, 2013.

[6] T. Barros, R. Ameur-Boulifa, A. Cansado, L. Henrio, and E. Madelaine. Behavioural models for distributed fractal components. *Annals of Télécommunications*, 64(1-2):25–43, 2009.

[7] T. Barros, R. Boulifa, and E. Madelaine. Parameterized Models for Distributed Java Objects. In *International Conference on Formal Techniques for Networked and Distributed Systems FORTE'04*, 2004. LNCS 3235.

[8] T. Barros, L. Henrio, and E. Madelaine. Behavioural models for hierarchical components. In *Proceedings of SPIN'05*. Spinger Verlag, 2005.

[9] A. Basu, B. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis. Rigorous component-based system design using the bip framework. *IEEE Softw.*, 28(3):41–48, May 2011.

[10] R. Boulifa, L. Henrio, and E. Madelaine. Behavioural models for group communications. In *WCSI-10: International Workshop on Component and Service Interoperability*, number 37 in EPTCS, pages 42–56, 2010.

[11] R. Boulifa and E. Madelaine. Finite model generation for distributed Java programs. In *Workshop on Model-Checking for Dependable Software-Intensive Systems*, San-Francisco, June 2003. North-Holland, IEEE. ISBN 0-7695-1952-0.

[12] R. de Simone. Higher-level synchronizing devices in Meije-SCCS. *Theoretical Computer Science*, 40, 1985.

[13] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, october 1992.

[14] Anna Ingólfsdóttir and Huimin Lin. A symbolic approach to value-passing processes. In *Handbook of Process Algebra, chapter 7. Elsevier Science*. Elsevier, 2001.

[15] A. Lakas. *Les Transformations Lotomaton : une contribution à la pré-implémentation des systèmes Lotos*. PhD thesis, Univ. Paris VI, june 1996.

[16] R. Mateescu and G. Salaün. Translating Pi-Calculus into LOTOS NT. In *IFM*, pages 229–244, 2010.

[17] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989. ISBN 0-13-114984-9.

[18] R. Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.

7 Appendix

7.1 Proof of proposition 4.4:

Strong Bisimulation is a congruence for top1-contexts Let P and Q be two closed pNets and $s \in S_\Phi(P)$, $t \in S_\Phi(Q)$ two strongly bisimilar states $(\!|s,P|\!) \sim (\!|t,Q|\!)$. Then for any top1-context C of sort $\mathcal{S}o$, i.e., $C = \langle\!\langle pNet_i^{i\in I}, (\bullet\!\!\mapsto\!\mathcal{S}o), \overline{SV}\rangle\!\rangle$, such that $\mathrm{Sort}(P) \cup \mathrm{Sort}(Q) \subseteq \mathcal{S}o$, we can build two closed pNets $C[(\bullet\!\!\mapsto\!P)]$ and $C[(\bullet\!\!\mapsto\!Q)]$ ensuring that:
for all s', t' of the form $s' = \triangleleft s_i^{i\in I\phi} \uplus (\bullet\!\!\mapsto\!s)\triangleright \in S_\Phi(C[\bullet\!\!\mapsto\!P])$, $t' = \triangleleft s_i^{i\in I\phi} \uplus (\bullet\!\!\mapsto\!t)\triangleright \in S_\Phi(C[\bullet\!\!\mapsto\!Q])$ with $\forall i \in I\phi.\, s_i \in S_\Phi(pNet_i)$; we have $(\!|s',P|\!) \sim (\!|t',Q|\!)$.

Proof. Because $(\!|s,P|\!) \sim (\!|t,Q|\!)$, then there exists a strong bisimulation $R \subseteq S_\Phi(P) \times S_\Phi(Q)$ such that $(s,t) \in R$.

The states of $C[(\bullet\!\!\mapsto\!P)]$ (resp. $C[(\bullet\!\!\mapsto\!Q)]$) are tuples of the form $\triangleleft s_i^{i\in I\phi} \uplus u\triangleright$, where the last element u is a state of P (resp. Q).

Define $\hat{R} = \{(\triangleleft s_i^{i\in I\phi} \uplus (\bullet\!\!\mapsto\!s)\triangleright, \triangleleft s_i^{i\in I\phi} \uplus (\bullet\!\!\mapsto\!t)\triangleright) | (s,t) \in R, \forall i \in I\phi.\, s_i \in S_\Phi(pNet_i)\}$. We will prove that $\hat{R}$ is a strong bisimulation.

Suppose $(\triangleleft s_i^{i\in I\phi} \uplus (\bullet\!\!\mapsto\!s)\triangleright, \triangleleft s_i^{i\in I\phi} \uplus (\bullet\!\!\mapsto\!t)\triangleright) \in \hat{R}$; then $(s,t) \in R$. For any valuation ϕ_1, for any transition $\triangleleft s_i^{i\in I\phi} \uplus (\bullet\!\!\mapsto\!s)\triangleright \xrightarrow{\alpha\phi_1} \triangleleft (s_i')^{i\in I\phi'} \uplus (\bullet\!\!\mapsto\!s')\triangleright$, let us prove (by case analysis on the form of the synchronisation vector used to generate the global action α) that there is a corresponding transition in $\|C[(\bullet\!\!\mapsto\!Q)]\|_\Phi$, and that their target states are bisimilar:

- If the action α only involves some $pNet_i^{i\in I}$, then there is a synchronisation vector, $\alpha_l^{l\in L} \to \alpha$, with $L \subseteq I$. We have $\forall l \in L\phi.\, s_l \xrightarrow{\alpha_l\phi_l} s_l'$ in $[\![pNet_i^{i\in I}]\!]_\Phi$ (for some of the sub-pNets, but not involving P), and $\triangleleft s_i^{i\in I\phi} \uplus (\bullet\!\!\mapsto\!s)\triangleright \xrightarrow{\alpha\phi'} \triangleleft (s_i')^{i\in I\phi'} \uplus (\bullet\!\!\mapsto\!s)\triangleright$, where $\phi' = \phi \uplus \{\phi_l\}^{l\in L\phi}$ and $\forall i \notin L.\, s_i' = s_i$. According to rule **Tr2**, similarly $\triangleleft s_i^{i\in I\phi} \uplus (\bullet\!\!\mapsto\!t)\triangleright \xrightarrow{\alpha\phi'} \triangleleft (s_i')^{i\in I\phi'} \uplus (\bullet\!\!\mapsto\!t)\triangleright$, and by definition of $\hat{R}$, $(\triangleleft (s_i')^{i\in I\phi'} \uplus (\bullet\!\!\mapsto\!s)\triangleright, \triangleleft (s_i')^{i\in I\phi'} \uplus (\bullet\!\!\mapsto\!t)\triangleright) \in \hat{R}$.
- If the interaction α involves both $pNet_i^{i\in I}$ and the hole, then rule **Tr2** is applicable and there is a synchronisation vector, $\alpha_l^{l\in L} \uplus (\bullet\!\!\mapsto\!\alpha_P) \to \alpha$, with $L \subseteq I$ (potentially $L = \emptyset$). We have $\forall l \in L\phi.\, s_l \xrightarrow{\alpha_l\phi_l} s_l'$ in $[\![pNet_i^{i\in I}]\!]_\Phi$ and $s \xrightarrow{\alpha_P\phi_P} s'$ in $[\![P]\!]_\Phi$, and $\triangleleft s_i^{i\in I\phi} \uplus (\bullet\!\!\mapsto\!s)\triangleright \xrightarrow{\alpha\phi'} \triangleleft (s_i')^{i\in I\phi'} \uplus (\bullet\!\!\mapsto\!s')\triangleright$, where $\phi' = \phi \uplus \{\phi_l\}^{l\in L\phi} \uplus \phi_P)$ and $\forall i \notin L.\, s_i' = s_i$. Since $(s,t) \in R$, then there exists ϕ_Q such that $t \xrightarrow{\beta\phi_Q} t'$ in $[\![Q]\!]_\Phi$, $(s',t') \in R$, and $\beta\phi_Q = \alpha_P\phi_P$. According to rule **Tr2**, $\triangleleft s_i^{i\in I\phi} \uplus (\bullet\!\!\mapsto\!t)\triangleright \xrightarrow{\alpha\phi''} \triangleleft (s_i')^{i\in I\phi''} \uplus (\bullet\!\!\mapsto\!t')\triangleright$, where $\phi'' = \phi \uplus \{\phi_l\}^{l\in L\phi} \uplus \phi_Q$. Finally, as $(s',t') \in R$, by definition of $\hat{R}$, $(\triangleleft (s_i')^{i\in I\phi'} \uplus (\bullet\!\!\mapsto\!s'\triangleright, \triangleleft (s_i')^{i\in I\phi'} \uplus (\bullet\!\!\mapsto\!t')\triangleright) \in \hat{R}$.

The proof for $\hat{R}^{-1}$ is similar, so we have proved that $\hat{R}$ is a strong bisimulation. □

7.2 Proof of theorem 4.5

Strong bisimulation is a congruence Let P and Q be two bisimilar closed pNets: $P \sim Q$. Then for any 1-context C of sort $\mathcal{S}o$, such that $\mathrm{Sort}(P) \cup \mathrm{Sort}(Q) \subseteq \mathcal{S}o$, we have $C[(\bullet\!\!\mapsto\!P)] \sim C[(\bullet\!\!\mapsto\!Q)]$.

Proof. We prove theorem 4.5 by induction on the structure of C:

- either C is a top1-context, we can apply Proposition 4.4, and conclude $C[(\bullet\mapsto P)] \sim C[(\bullet\mapsto Q)]$.
- or C is a 1-context of the form $\langle\!\langle pNet_i^{i\in I}, \emptyset, \overline{SV}\rangle\!\rangle$, with $pNet_{i_0}$ a 1-context for a given $i_0 \in I$, with $Sort(pNet_{i_0}) = \mathcal{S}o_{i_0}$, and all other $pNet_i^{i\in I\setminus i_0}$ are closed pNets.
  By induction hypothesis we have $pNet_{i_0}[(\bullet\mapsto P)] \sim pNet_{i_0}[(\bullet\mapsto Q)]$.
  Now consider the top1-context $C'[(\bullet\mapsto\mathcal{S}o_{i_0})] = \langle\!\langle pNet_i^{i\in I\setminus i_0}, \{(\bullet\mapsto\mathcal{S}o_{i_0})\}, \overline{SV}\rangle\!\rangle$.
  Then using Property 4.4 we have:
  $C[(\bullet\mapsto P)] = C'[(\bullet\mapsto(pNet_{i_0}[(\bullet\mapsto P])) \sim C'[(\bullet\mapsto(pNet_{i_0}[(\bullet\mapsto Q])) = C[(\bullet\mapsto Q)]$

□

7.3 Proof of Proposition 4.7

Flattening and strong bisimulation

$$\forall P \text{ closed } pNet, C[\bullet\mapsto C'[\bullet\mapsto P]]\sim Flat(C, C')[\bullet\mapsto P]$$

Proof. Let P be a closed pNet and C and C' two contexts; $C = \langle\!\langle\overline{pNet}, (\bullet\mapsto\mathcal{S}o), \overline{SV}\rangle\!\rangle$ and $C' = \langle\!\langle\overline{pNet'}, (\bullet\mapsto\mathcal{S}o'), \overline{SV'}\rangle\!\rangle$ with $\mathrm{Sort}(C') \subseteq \mathcal{S}o$. Then

$$Flat(C, C') = \langle\!\langle\overline{pNet} \uplus \overline{pNet'}, (\bullet\mapsto\mathcal{S}o'), \overline{SV''}\rangle\!\rangle$$

with $\overline{SV''}$ defined below:

$$\begin{array}{rcl}\overline{SV''} & = & \{(\alpha_l\varphi)^{l\in L\setminus\{\bullet\}} \uplus \beta_l^{l\in L'} \to\alpha' \,|\alpha_l^{l\in L} \to\alpha'\varphi \in \overline{SV} \wedge \beta_l^{l\in L'} \to \beta \in \overline{SV'} \\ & & \quad \wedge\bullet \in L \wedge \alpha_\bullet\varphi = \beta \wedge \varphi = (x_n \to y_n)^{n\in N} \wedge \forall n \in N.\, (x_n \in fv(\alpha_\bullet) \wedge y_n \in fv(\beta))\} \\ & & \cup\{(\alpha_l)^{l\in L} \to \alpha' \,|(\alpha_l)^{l\in L} \to \alpha' \in \overline{SV} \wedge \bullet \notin L\}\end{array}$$

First, we let $\overline{pNet} = pNet_i^{i\in I}$ and $\overline{pNet'} = pNet'^{i\in I'}_i$. To prove $C[\bullet\mapsto C'[\bullet\mapsto P]]\sim Flat(C, C')[\bullet\mapsto P]$, we only need to prove their initial states are bisimilar. Consider the closed pNet P and an initial valuation $\phi_0 \in \Phi$ associating a value to each variable of P. We will make a binary relation

$$R = \Big\{ \Big(\triangleleft s_i^{i\in I\phi} \uplus (\bullet\mapsto \triangleleft (s'_i)^{i\in I'\phi'} \uplus (\bullet\mapsto s)\triangleright\triangleright\,,\ \triangleleft s_i^{i\in I\phi} \uplus s_i'^{i\in I'\phi'} \uplus \bullet\mapsto s\triangleright\Big) \,| \\ \forall i \in I\phi.\, s_i \in S_\Phi(pNet_i) \wedge \forall i \in I'\phi'.\, s'_i \in S_\Phi(pNet'_i) \wedge s \in S_\Phi(P)\Big\}$$

It is easy to see that the initial states composed from an initial valuation ϕ_0 and initial states of the sub-pNets are related according to R.

Then, first for any two states related by R:
$\Big(\triangleleft s_i^{i\in I\phi} \uplus (\bullet\mapsto \triangleleft s_i'^{i\in I'\phi'} \uplus (\bullet\mapsto s)\triangleright\triangleright\,,\ \triangleleft s_i^{i\in I\phi} \uplus s_i'^{i\in I'\phi'} \uplus \bullet\mapsto s\triangleright\Big) \in R$, suppose that

$$\triangleleft s_i^{i\in I\phi} \uplus (\bullet\mapsto \triangleleft s_i'^{i\in I'\phi'} \uplus (\bullet\mapsto s)\triangleright\triangleright \xrightarrow{\alpha\phi_1} \triangleleft t_i^{i\in I\phi_2} \uplus (\bullet\mapsto \triangleleft t_i'^{i\in I'\phi'_2} \uplus (\bullet\mapsto t)\triangleright\triangleright$$

We will prove that $\triangleleft s_i^{i\in I\phi} \uplus s_i'^{i\in I'\phi'} \uplus (\bullet\mapsto s \uplus \varphi_0)\triangleright \xrightarrow{\alpha\phi_1} \triangleleft t_i^{i\in I\phi_2} \uplus t_i'^{i\in I'\phi'_2} \uplus (\bullet\mapsto t)\triangleright$, which is sufficient to ensure that R is a simulation because additionally, $\triangleleft t_i^{i\in I\phi_2} \uplus (\bullet\mapsto \triangleleft (t'_i)^{i\in I'\phi'_2} \uplus (\bullet\mapsto t)\triangleright$

$\triangleright, \lhd t_i^{i\in I\phi_2} \uplus (t'_i)^{i\in I'\phi'_2} \uplus (\bullet\mapsto t)\triangleright \in R$. We focus on the case involving actions of C, C', and P at the same time; the other cases are either special case of this one or similar to the proof of Proposition 4.4.

As the action $\alpha\phi_1$ can be performed in $[\![C[\bullet\mapsto C'[\bullet\mapsto P]]]\!]_\Phi$, based on the definition of rule **Tr2**, there must be a synchronisation vector of the form $\alpha_l^{l\in L\uplus\{\bullet\}} \rightarrow \alpha$ in the definition of $C[\bullet\mapsto C'[\bullet\mapsto P]]$ with $L \subseteq I$ (in the case involving C, C', and P). Also, this implies that by rule **Tr2**, there is ψ such that $\beta\psi$ is a transition of $\|C'[\bullet\mapsto P]\|_\Phi$, and thus there is a synchronisation vector in C' of the form $\beta_l^{l\in L'\uplus\{\bullet\}} \rightarrow \beta$ with $L' \subseteq I'$ (remember we consider that P is involved in the synchronisation), additionally $\beta\psi = \alpha_\bullet\phi_1$ by applicability of **Tr2**. And there must exist φ such that $\beta = \alpha_\bullet\varphi$ because action parameters must be variables. Consequently, by definition of $Flat(C, C')$, $\alpha_l\varphi^{l\in L} \uplus \beta_l^{l\in L'\uplus\bullet} \rightarrow \alpha\varphi$ is a synchronisation vector of $Flat(C, C')$. Thus the action $\alpha\phi_1$ can be performed in $[\![Flat(C, C')[\bullet\mapsto P]]\!]_\Phi$; it leads to the expected state by construction (the two **Tr2** rules are replaced by a single one): $\lhd s_i^{i\in I\phi} \uplus s'^{i\in I'\phi'}_i \uplus (\bullet\mapsto s)\triangleright \xrightarrow{\alpha\phi_1} \lhd t_i^{i\in I\phi_2} \uplus t'^{i\in I'\phi'_2}_i \uplus (\bullet\mapsto t)\triangleright$.

Conversely, when proving that R^{-1} is a simulation, the most representative case is when all the processes are involved. The proof is similar:
if $\left(\lhd s_i^{i\in I\phi} \uplus (\bullet\mapsto \lhd s'^{i\in I'\phi'}_i \triangleright) \uplus (\bullet\mapsto s) \triangleright \ , \ \lhd s_i^{i\in I\phi} \uplus s'^{i\in I'\phi'}_i \uplus \bullet\mapsto s\triangleright\right) \in R$, and a transition can be triggered in $[\![Flat(C, C')[\bullet\mapsto P]]\!]_\Phi$, then:

$$\lhd s_i^{i\in I\phi} \uplus s'^{i\in I'\phi'}_i \uplus (\bullet\mapsto s)\triangleright \xrightarrow{\alpha\phi_1} \lhd t_i^{i\in I\phi_2} \uplus t'^{i\in I'\phi'_2}_i \uplus (\bullet\mapsto t)\triangleright$$

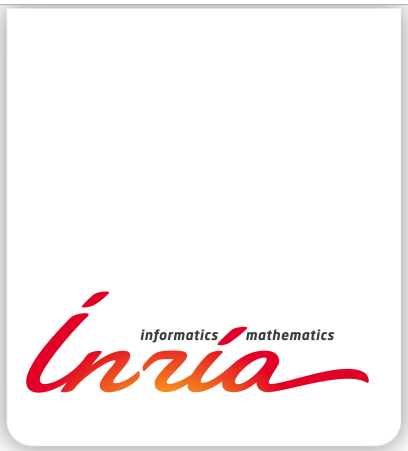
then we prove that

$$\lhd s_i^{i\in I\phi} \uplus (\bullet\mapsto \lhd s'^{i\in I'\phi'}_i \triangleright) \uplus (\bullet\mapsto s)\triangleright \xrightarrow{\alpha\phi_1} \lhd t_i{}^{i\in I\phi_2} \uplus (\bullet\mapsto \lhd t'^{i\in I'\phi'_2}_i \triangleright) \uplus (\bullet\mapsto t)\triangleright$$

with, by definition of R: $\lhd t_i^{i\in I\phi_2} \uplus (\bullet\mapsto \lhd t'^{i\in I'\phi'_2}_i \triangleright) \uplus (\bullet\mapsto t)\triangleright, \lhd t_i^{i\in I\phi_2} \uplus t'^{i\in I'\phi'_2}_i \uplus (\bullet\mapsto t)\triangleright \in R$

Similarly to the case above, the important point is to notice that, if a transition is triggered in $[\![Flat(C, C')[\bullet\mapsto P]]\!]_\Phi$ then the rule **Tr2** can be triggered and there must exist a synchronisation in $Flat(C, C')[\bullet\mapsto P]$ of the form: $\alpha_l^{l\in L} \uplus \beta_l^{l\in L'\uplus\{\bullet\}} \rightarrow \alpha$; then (by definition of the flattening operator) there must exist a synchronisation vector in C of the form: $(\alpha_l\varphi^{-1})^{l\in L\uplus\bullet} \rightarrow \alpha\varphi^{-1}$ and in C' of the form: $\beta_l^{l\in L'\uplus\bullet} \rightarrow \beta$ where φ is the assignment that was used to build the synchronisation vector($\beta = \alpha_\bullet\varphi$). Then it is easy to see that the action $\alpha\phi_1$ can be performed in $[\![C[\bullet\mapsto C'[\bullet\mapsto P]]]\!]_\Phi$ by two applications of the **Tr2** rule.

□



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93

06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399