# Compositional equivalences based on Open pNets

Rabéa Ameur-Boulifa

*LTCI, Télécom Paris, Institut Polytechnique de Paris, France*

Ludovic Henrio*

*Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France.*

Eric Madelaine

*INRIA Sophia Antipolis Méditérannée, UCA, BP 93, 06902 Sophia Antipolis, France*

**Abstract**

Establishing equivalences between programs or systems is crucial both for verifying correctness of programs and for justifying optimisations and program transformations. There exist several equivalence relations for programs, and bisimulations are among the most versatile of these equivalences. Among bisimulations one distinguishes strong bisimulation that requires that each action of a program is simulated by a single action of the equivalent program, and weak bisimulation that allows some of the actions to be invisible, and thus not simulated by the equivalent program.

pNet is a generalisation of automata that model open systems. They feature variables and hierarchical composition. Open pNets are pNets with holes, i.e. placeholders inside the hierarchical structure that can be filled later by subsystems. However, there is no standard tool for defining the semantics of an open system in this context. This article first defines *open automata* that are labelled transition systems with parameters and holes.

Relying on open automata, this article defines bisimulation relations for the comparison of systems specified as pNets. We first define a strong bisimulation for open pNets. We then define an equivalence relation similar to the classical *weak bisimulation*, and study its properties. Among these properties we are interested in compositionality: if two systems are proven equivalent they will be undistinguishable by their context, and they will also be undistinguishable when their holes are filled with equivalent systems. We identify sufficient conditions to ensure compositionality of strong and weak bisimulation. The article is illustrated with a transport protocol running example.

*Keywords:* Bisimulation, compositionality, automata, semantics.

*Corresponding author

*Email addresses:* rabea.ameur-boulifa@telecom-paris.fr (Rabéa Ameur-Boulifa), ludovic.henrio@cnrs.fr (Ludovic Henrio), eric.madelaine@inria.fr (Eric Madelaine)

## 1. Introduction

In the nineties, several works extended the basic behavioural models based on labelled transition systems to address value-passing or parameterised systems, using various symbolic encodings of the transitions [1, 2]. These works use the term *parameter* to designate systems where variables that have a strong influence the system structure and behaviour. In parameterised systems, parameters can typically be the number of processes in the system or the way they interact. In [3, 4], Lin, Ingolfsdottir and Hennessy developed a full hierarchy of bisimulation equivalences, together with a proof system, for value passing CCS, including notions of symbolic behavioural semantics and various symbolic bisimulations (early and late, strong and weak, and their congruent versions). They also extended this work to models with explicit assignments [5]. Separately Rathke [6] defined another symbolic semantics for a parameterised broadcast calculus, together with strong and weak bisimulation equivalences, and developed a symbolic model-checker based on a tableau method for these processes. 30 years later, no practical verification approach and no verification platform are using this kind of approaches to provide proof methods for value-passing processes or open process expressions.

This article provides a theoretical background that allows us to implement such a verification platform. We build upon the concept of pNets that allowed us to give a behavioural semantics of distributed components and verify the correctness of distributed applications in the past 15 years. pNets is a low level semantic framework for expressing the behaviour of various classes of distributed languages, and as a common internal format for our tools. pNets allow the specification of parameterised hierarchical labelled transition systems: labelled transition systems with parameters can be combined hierarchically.

We develop here a semantics for a model of interacting processes with parameters and holes. Our approach is originally inspired from Structured Operational Semantics with conditional premisses as in [7, 8]. But we aim at a more constructive and implementable approach to compute the semantics (intuitively transitions including first order predicates) and to check equivalences for these open systems. The main interest of our symbolic approach is to define a method to prove properties directly on open structures; these properties will then be preserved by any correct instantiation of the holes. As a consequence, our model allows us to reason on composition operators as well as on full-size distributed systems. The parametric nature of the model and the properties of compositionability of the equivalence relations are thus the main strengths of our approach.

*pNets.* pNet is a convenient model to model concurrent systems in a hierarchical and parameterised way. The coordination between processes is expressed as synchronisation vectors that allow the definition of complex and expressive synchronisation patterns. Open pNets are pNets for which some elements in the

hierarchy are still undefined. Such undefined elements are called *holes* and a hole can be *filled* later by providing another pNet. This article first defines pNets and illustrates with an example how they can be used to provide the model of a communicating system.

*A semantics for Open pNets based on open automata.* The semantics of pNets can be expressed as a translation to a labelled transition system, but only if the pNet has no parameter and no hole. Adding parameters to a LTS is quite standard but enabling holes inside LTSs is not a well-defined notion.

To define a semantics for open pNets we thus need LTSs that have both standard variable parameters, and process parameters, i.e. holes that can be filled by processes. We call such LTSs with parameters and holes ~~and equiped with a composition operation~~, *open automata*. The main goal of this article is to define the theory behind open automata and to use them to provide a semantics and prove composition properties for open pNets. The transitions of open automata are much more complex than transitions of an LTS as the firing of a transition depends on parameters and actions that are symbolic. This article defines the notion of *open transition*, a transition that is symbolic in terms of parameters and coordinated actions.

Contrarily to pNets, open automata are not hierarchical structures, we consider them here as a mathematical structure convenient for formal reasoning but not adapted to the definition of a complex and structured system. Open transitions are expressed in terms of logics while the communication in pNets is specified as synchronisation vectors that specify synchronised actions. Open automata could alternatively be seen as an algebra that can be studied independently from its application to pNets, but their composition properties make more sense in a hierarchical model like pNets.

*Previous Works and Contribution*

While most of our previous works relied on closed, fully-instantiated semantics [9, 10, 11], it is only recently that we could design a first version of a parameterised semantics for pNets with a strong bisimulation equivalence [12]. This article builds upon this previous parameterised semantics and provides a clean and complete version of the semantics with a slightly simplified formalism that makes proofs easier. It also adds a notion of global state to automata. Also, in [12] the study of compositionality was only partial, and in particular the proof that bisimulation is an equivalence is one new contribution of this article and provides a particularly interesting insight on the semantic model we use. The new formalism allowed us to extend the work and define weak bisimulation for open automata, which is entirely new. This allows us to define a weak bisimulation equivalence for open pNets with valuable properties of compositionality. To summarise, the contribution of this paper are the following:

- The definition of open automata: an algebra of parameterised automata with holes, and a strong bisimulation equivalence. This is an adaptation of [12] with an additional property stating that strong bisimulation equivalence is indeed an equivalence relation.

- A semantics for open pNets expressed as translation to open automata. This is an adaptation of [12] with a complete proof that strong bisimulation is compositional.
- A theory of weak bisimulation for open automata, and its properties. It relies on the definition of weak open transitions that are derived from transitions of the open automaton by concatenating invisible action transitions with one (visible or not) action transition. The precise and sound definition of the concatenation is also a major contribution of this article.
- A resulting weak bisimulation equivalence for open pNets and a simple static condition on synchronisation vectors inside pNets that is sufficient to ensure that weak bisimulation is compositional.
- An illustrative example based on a simple transport protocol, showing the construction of the weak open transitions, and the proof of weak bisimulation.

*What is new about open automata bisimulation?*

Bisimulation over a symbolic and open model like open pNets or open automata is different from the classical notion of bisimulation because it cannot rely on the equality over a finite set of action labels. Classical bisimulations require to exhibit, for each transition of one system, a transition of the other system that simulates it. Instead, bisimulation for open automata relies on the simulation of each open transition of one automaton by a set of open transitions of the other one, that should cover all the cases where the original transition can be triggered.

Compositionality of bisimulation in our model come from the specification of the interactions, including actions of the holes. In pNets, synchronisation vectors define the possible interactions between the pNet that fills the hole and the surrounding pNets. In open automata, this is reflected by symbolic hypotheses that depend on the actions of the holes. This additional specification is the price to pay to obtain the compositionality of bisimulation that cannot be guaranteed in traditional process algebras.

This approach also allows us to specify a sufficient condition on allowed transitions to make weak bisimulation compositional; namely it is not possible to synchronise on invisible actions from the holes or prevent them to occur.

*Structure*

This article is organised as follows. Section 2 provides the definition of pNets and introduces the notations used in this paper, including the definition of open pNets. Section 3 defines open automata, i.e. automata with parameters and transitions conditioned by the behaviour of "holes"; a strong bisimulation equivalence for open automata is also presented in this section. Section 4 gives the semantics of open pNets expressed as open automata, and states compositional properties ~~on the~~ strong bisimulation for open pNets. Section 5 defines a weak bisimulation equivalence on open automata and derives weak bisimilarity for

pNets, together with properties ~~on~~ compositionality of weak bisimulation ~~for~~ open pNets. Finally, Section 6 discusses ~~related~~ works and Section 7 concludes the paper.

## 2. Background and notations

This section introduces the notations we will use in this article, and recalls the definition of pNets [12] with an informal semantics of the pNet constructs. The only significant difference compared to our previous definitions is that we remove here the restriction that was stating that variables should be local to a state of a labelled transition system.

### 2.1. Notations

*Term algebra.* Our models rely on a notion of parameterised actions, ~~that~~ are symbolic expressions using data types and variables. As our model aims at encoding the low-level behaviour of possibly very different programming languages, we do not want to impose one specific algebra for denoting actions, nor any specific communication mechanism. So we leave unspecified the constructors of the algebra that will ~~allow building~~ expressions and actions. Moreover, we use a generic *action interaction* mechanism, based on (some sort of) unification between two or more action expressions, to express various kinds of communication or synchronisation mechanisms.

Formally, we assume the existence of a term algebra $\mathbb{T}$, and denote as $\Sigma$ the signature of the data and action constructors. Within $\mathbb{T}$, we distinguish a set of data expressions $\mathbb{E}$, including a set of boolean expressions $\mathbb{B}$ ($\mathbb{B} \subseteq \mathbb{E}$), and a set of action expressions $\mathbb{A}$ called the action algebra $\mathbb{A}$, with $\mathbb{A} \subseteq \mathbb{T}, \mathbb{E} \cap \mathbb{A} = \emptyset$; naturally action terms will use data expressions as sub-terms. The function $vars(t)$ identifies the set of variables in a term $t \in \mathbb{T}$.

We let $e_i$ range over expressions ($e_i \in \mathbb{E}$), $a$ range over action labels, $op$ be operators, and $x_i$ and $y_i$ range over variable names.

We define two kinds of parameterised actions. The first kind distinguishes input variables and the second kind does not. The actions that distinguish input variables will be used in the definition of *pLTS* below and are defined as follows:

| | | | |
|---|---|---|---|
| $\alpha \in \mathbb{A}$ | ::= | $a(p_1, \ldots, p_n)$ | action terms |
| $p_i$ | ::= | $?x \mid e_i$ | parameters (input var or expression) |
| $e_i$ | ::= | $Value \mid x \mid op(e_1, .., e_n)$ | Expressions |

The *input variables* in an action term are those marked with a *?*. We additionally impose that each input variable does not appear ~~somewhere~~ else in the same action term: $p_i = ?x \Rightarrow \forall j \neq i.\, x \notin vars(p_j)$. We define $iv(t)$ as the set of input variables of a term $t$ (without the '?' marker). Action algebras can encode naturally usual point-to-point message passing calculi (using $a(?x_1, ..., ?x_n)$ for inputs, $a(v_1, .., v_n)$ for outputs), but it also allows for more general synchronisation mechanisms, like gate negotiation in Lotos, or broadcast communications.

The set of actions that do not distinguish input variables is denoted $\mathbb{A}_S$, it will be used in synchronisation vectors of pNets:

$$\alpha \in \mathbb{A}_S \quad ::= \quad a(e_1, \ldots, e_n)$$

*Indexed sets.* This article extensively uses indexed structures (maps) over some countable indexed sets. The indices can typically be integers, bounded or not. We use indexed sets in pNets because we want to consider a set of processes, and specify separately how to synchronise them. Roughly this could also be realised using tuples, however indexed sets are more general, can be infinite, and give a more compact representation than using the position in a possibly long tuple.

An indexed family is denoted as follows: $t_i^{i\in I}$ is a family of elements $t_i$ indexed over the set $I$. Such a family is equivalent to the mapping $(i \mapsto t_i)^{i\in I}$, and we will also use mapping notations to manipulate indexed sets. To specify the set over which the structure is indexed, indexed structures are always denoted with an exponent of the form $i \in I$.

Consequently, $t_i^{i\in I}$ defines first $I$ the set over which the family is indexed, and then $t_i$ the elements of the family. For example $t_i^{i\in\{3\}}$ is the mapping with a single entry $t_3$ at index 3; exceptionally, for mappings with only a few entries we use the notation $(3 \mapsto t_3)$ instead. In this article, sentences of the form "there exists $t_i^{i\in I}$" means there exists $I$ and a function that maps each element of $I$ to a term $t_i$.

When this is not ambiguous, we shall use abusive notations for sets, and typically write "indexed set over I" when formally we should speak of multisets, and "$x \in A_i^{i\in I}$" to mean $\exists i \in I.\, x = A_i$. To simplify equations, an indexed set can be denoted $\bar{t}$ instead of $t_i^{i\in I}$ when $I$ is irrelevant.

The disjoint union on sets is $\uplus$. We extend it to disjoint union of indexed sets defined by the merge of the two sets provided they are indexed on disjoint families. The elements of the union of two indexed sets are then accessed by using an index of one of the two joined families. The standard subtraction operation on indexed sets is $\setminus$, with $\mathrm{dom}(A \setminus B) = \mathrm{dom}(A) \setminus B$.

*Substitutions.* This article also uses substitutions. Applying a substitution inside a term $t$ is denoted $t\{\!\{y_i \leftarrow e_i\}\!\}^{i\in I}$ and consists in replacing in parallel all the occurrences of variables $y_i$ in the term $t$ by the terms $e_i$. Note that a substitution is defined by a partial function that is applied on the variables inside a term. We let *Post* range over partial functions that are used as substitution and use the notation $\{y_i \leftarrow e_i\}^{i\in I}$ to define such a partial function[1]. These partial functions are sometimes called *substitution functions* in the following. Thus, $\{\!\{Post\}\!\}$ is the operation that applies, in a parallel manner, the substitution defined by the partial function *Post*. $\odot$ is a composition operator on these partial functions, such that for any term $t$ we have: $t\{\!\{Post \odot Post'\}\!\} = (t\{\!\{Post'\}\!\})\{\!\{Post\}\!\}$. This property must also be valid when the substitution does not operate on all

[1] When using this notation, we suppose, without loss of generality that each $y_i$ is different.

variables. We thus define a composition operation as follows:

$$(x_k \leftarrow e_k)^{k\in K} \odot (x'_{k'} \leftarrow e'_{k'})^{k'\in K'} = (x_k \leftarrow e_k\{\!\{(x'_{k'} \leftarrow e'_{k'})^{k'\in K'}\}\!\})^{k\in K} \cup (x'_{k'} \leftarrow e'_{k'})^{k'\in K''}$$

where $K'' = \{k' \in K' | x'_{k'} \notin \{x_k\}^{k\in K}\}$

*2.2. Parameterised Networks (pNets)*

pNets are tree-like structures, where the leaves are either *parameterised labelled transition systems (pLTSs)*, expressing the behaviour of basic processes, or *holes*, used as placeholders for unknown processes. Nodes of the tree (pNet nodes) are synchronising artefacts, using a set of *synchronisation vectors* that express the possible synchronisation between the parameterised actions of a subset of the sub-trees.

A pLTS is a labelled transition system with variables; variables can be used inside states, actions, guards, and assignments. Note that we make no assumption on finiteness of the set of states nor on finite branching of the transition relation. Compared to our previous works [12, 10] we extend the expressiveness of the model by making variables global.

**Definition 1 (pLTS).** A pLTS is a tuple $pLTS \triangleq \langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle$ where:

- $S$ is a set of states.
- $s_0 \in S$ is the initial state.
- $V$ is a set of global variables for the pLTS.
- $\rightarrow \subseteq S \times L \times S$ is the transition relation and $L$ is the set of labels of the form: $\langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J} \rangle$, where $\alpha \in \mathbb{A}$ is a parameterised action, $e_b \in \mathbb{B}$ is a guard, and the variables $x_j$ are assigned the expressions $e_j \in \mathbb{E}$. If $s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J} \rangle} s' \in \rightarrow$ then $vars(\alpha) \setminus iv(\alpha) \subseteq V$, $vars(e_b) \subseteq vars(s) \cup vars(\alpha)$, and $\forall j \in J.\ (vars(e_j) \subseteq V \cup iv(\alpha) \wedge x_j \in V)$.

A set of assignments between two states is performed in parallel so that their order do not matter and they all use the values of variables before the transition or the values received as action parameters.

Now we define pNet nodes as constructors for hierarchical behavioural structures. A pNet has a set of sub-pNets that can be either pNets or pLTSs, and a set of holes, playing the role of process parameters. A pNet is thus a composition operator that can receive processes as parameters; it expresses how the actions of the sub-processes synchronise.

Each sub-pNet exposes a set of actions, called *internal actions*. The synchronisation between global actions exposed by the pNet and internal actions of its sub-pNets is given by *synchronisation vectors*: a synchronisation vector

synchronises one or several internal actions, and exposes a single resulting global action.

We now define the structure of pNets, the following definition relies on the definition of holes, leaves and sorts formalised below in Definition 3. Informally, holes are process parameters, leaves provide the set of pLTSs at the leaves of the hierarchical structure of a pNet, and sorts give the signature of a pNet, i.e. the actions it exposes.

**Definition 2 (pNets).** A pNet $P$ is a hierarchical structure where leaves are pLTSs and holes

$P \triangleq pLTS \mid \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, SV_k^{k\in K} \rangle\!\rangle$ where:

- $I$ is a set of indices and $P_i^{i\in I}$ is the family of sub-pNets indexed over $I$. $vars(P_i)$ and $vars(P_j)$ must be disjoint for $i \neq j$.
- $J$ is a set of indices, called *holes*. $I$ and $J$ are *disjoint*: $I \cap J = \emptyset$, $I \cup J \neq \emptyset$.
- $Sort_j \subseteq \mathbb{A}_S$ is a set of action terms, denoting the *sort* of hole $j$.
- $SV_k^{k\in K}$ is a set of synchronisation vectors.
  $\forall k \in K.\, SV_k = \alpha_l^{l\in I_k \uplus J_k} \rightarrow \alpha'_k[e_k]$ where $\alpha'_k \in \mathbb{A}_S$, $I_k \subseteq I$, $J_k \subseteq J$, $\forall i \in I_k.\, \alpha_i \in Sort(P_i)$, $\forall j \in J_k.\, \alpha_j \in Sort_j$, and $vars(\alpha'_k) \subseteq \bigcup_{l\in I_k\uplus J_k} vars(\alpha_l)$. The global action of a vector $SV_k$ is $\alpha'_k$. $e_k \in \mathbb{B}$ is a guard associated to the vector such that $vars(e_k) \subseteq \bigcup_{l\in I_k\uplus J_k} vars(\alpha_l)$.

Synchronisation vectors are identified modulo renaming of variables that appear in their action terms.

The preceding definition relies on the auxiliary functions defined below:

**Definition 3 (Sorts, Holes, Leaves, Variables of pNets).**

- The sort of a pNet is its signature, i.e. the set of actions in $\mathbb{A}_S$ it can perform, where each action signature is an action label plus the arity of the action.

$$\mathrm{Sort}(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \{\mathrm{Sort}(\alpha) | s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J} \rangle} s' \in \rightarrow\}$$
$$\mathrm{Sort}(\langle\!\langle \overline{P}, \overline{Sort}, \overline{SV} \rangle\!\rangle) = \{\mathrm{Sort}(\alpha') | \overline{\alpha} \rightarrow \alpha'[e_b] \in \overline{SV}\}$$
$$\mathrm{Sort}(\alpha(p_1, .., p_n)) = (\alpha, n)$$

- The set of variables of a pNet $P$, denoted $vars(P)$ is disjoint union the set of variables of all pLTSs that compose $P$.
- The set of holes Holes($P$) of a pNet is the indices of the holes of the pNet itself plus the indices of all the holes of its sub-pNets. It is defined inductively (we suppose those indices disjoints):

$$\mathrm{Holes}(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \emptyset$$
$$\mathrm{Holes}(\langle\!\langle P_i^{i\in I}, \overline{Sort}, \overline{SV} \rangle\!\rangle) = J \uplus \bigcup_{i\in I} \mathrm{Holes}(P_i)$$
$$\forall i \in I.\ \mathrm{Holes}(P_i) \cap J = \emptyset$$
$$\forall i_1, i_2 \in I.\, i_1 \neq i_2 \Rightarrow \mathrm{Holes}(P_{i_1}) \cap \mathrm{Holes}(P_{i_2}) = \emptyset$$

typesetting is a little odd.

I don't think you said what these are. You only do it in Def. 3.

A comparison with Larsen & Liu's context systems and Larsen & Hüttel's "static constructs" (1989) might be appropriate.

Explain better. A small pictorial example illustrating Def. 2 might be in order here.

set of

index sets are

← This "hides" the origin of J.

- The set of leaves of a pNet is the set of all pLTSs occurring in the structure, as an indexed family of the form $\text{Leaves}(P) = \langle\!\langle P_i \rangle\!\rangle^{i \in L}$.

$$\text{Leaves}(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \emptyset$$
$$\text{Leaves}(\langle\!\langle P_i^{i \in I}, \overline{Sort}, \overline{SV} \rangle\!\rangle) = \biguplus_{i \in I} \text{Leaves}(P_i) \uplus \{i \mapsto P_i | P_i \text{ is a } pLTS\}$$

A pNet $Q$ is *closed* if it has no hole: $\text{Holes}(Q) = \emptyset$; else it is said to be *open*. Sort comes naturally with a compatibility relation that is similar to a type-compatibility check. We simply say that two sorts are compatible if they consist of the same actions with the same arity. In practice, it is sufficient to check the equality of the two sets of action signatures of the two sorts[2].

The informal semantics of pNets is as follows. pLTSs behave more or less like classical automata with conditional branching and variables. The actions on the *pLTS*s can send or receive values, potentially modifying the value of variables. pNets are synchronisation entities: a pNet node composes several sub-pNets and define how the sub-pNets interact, where a sub-pNet is either a pNet or a pLTS. The synchronisation between sub-pNets is defined by synchronisation vectors (originally introduced by [13]) that express how an action of a sub-pNet can be synchronised with actions of other sub-pNet, and how the resulting synchronised action is visible from outside of the pNet. The synchronisation mechanism is very expressive, including pattern-matching/unification between the parameterized actions within the vector, and an additional predicate over their variables. Consider a pNet node that assembles several pLTSs, the synchronisation vectors specify the way that transitions of the composed pNet are built from the transitions of the sub-nets. This can be seen as "conditional transitions" of the pNet, or alternatively, as a syntax to encode structural operational semantics (SOS rules) of the system: each vector expresses not only the actions emitted by the pNet but also what transitions of the composed pLTSs must occur to trigger this global transition. Synchronisation vectors can also express the exportation of an action of a sub-pNet to the next level, or to hide an interaction and make it non-observable. Finally, a pNet can leave sub-pNets undefined and instead declare holes with a well-defined signature. Holes can then be filled with a sub-pNet. This is defined as follows.

**Definition 4 (pNet composition).** An open pNet: $P = \langle\!\langle P_i^{i \in I}, Sort_j^{j \in J}, \overline{SV} \rangle\!\rangle$ can be (partially) filled by providing a pNet $Q$ to fill one of its holes. Suppose $j_0 \in J$ and $\text{Sort}(Q) \subseteq Sort_{j_0}$, then:

$$P\,[Q]_{j_0} = \langle\!\langle P_i^{i \in I} \uplus \{j_0 \mapsto Q\}, Sort_j^{j \in J \setminus \{j_0\}}, \overline{SV} \rangle\!\rangle$$

pNets are composition entities equipped with a rich synchronisation mechanism: synchronisation vectors allow the expression of synchronisation between

[2] A more complex compatibility relation could be defined, but this is out of the scope of this article.

any number of entities and at the same time the passing of data between processes. Their strongest feature is that the data emitted by processes can be used inside the synchronisation vector to do addressing: it is easy to synchronise a process indexed by $n$ with the action $a(v, n)$ of another process. This is very convenient to model systems and encode futures or message routing.

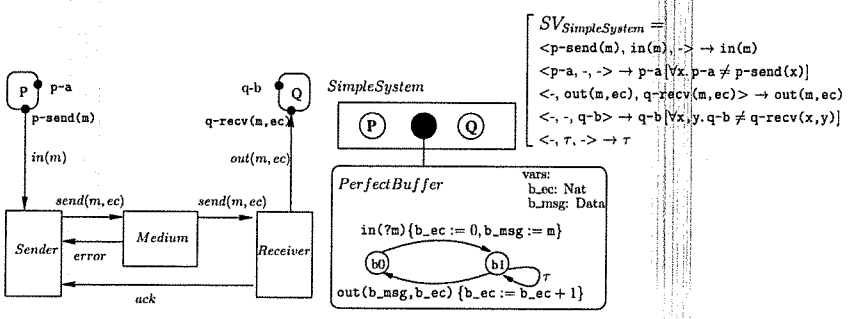
pNets have been used to model GCM distributed component systems, illustrating the expressiveness of the model [10]. These works show that pNets are convenient to express the behaviour of a system in a compositional way. Unfortunately, the semantics of pNets and the existing tools at this point were only able to deal with a closed system completely instantiated: pNets could be used as composition operator in the definition of the semantics, which was sufficient to perform finite-state model checking on a closed system, but there was no theory for the use of pNets as operators and no tool for proving properties on open system. Consequently, much of the formalisation efforts did not use holes and the interplay between holes, sorts, and synchronisation vector was not formalised. In previous works [10], only closed pNets were equipped with a semantics, it was defined as labelled transition systems. The theory of pNets as operators able to fully take into account open systems is given in the following sections. Comparing formally the existing direct operational semantics and the semantics derived from open automata in the current article would be an interesting partial proof of soundness for our semantics. The proof could only be partial as the formal semantics that exists only consider closed and fully instantiated pNets. Proving an equivalence between the semantics presented in this article and the operational one shown in [10] is outside the scope of this article because we focus here on the modelisation of holes that were not modelled in the previous semantics.

### 2.3. Running Example

To illustrate this work, we use a simple communication protocol, that provides safe transport of data between two processes, over unsafe media.

Figure 1 (left) shows the example principle, which corresponds to the hierarchical structure of a pNet: two unspecified processes $P$ and $Q$ (holes) communicate messages, with a data value argument, through the two protocol entities. Process $P$ sends an `p-send(m)` message to the *Sender*; this communication is denoted as `in(m)`. At the other end, process $Q$ receives the message from the *Receiver*. The holes $P$ and $Q$ can also have other interactions with their environment, represented here by actions `p-a` and `q-b`. The underlying network is modelled by a medium entity transporting messages from the sender to the receiver, and that is able to detect transport errors and signal them to the sender. The return *ack* message from *Receiver* to *Sender* is supposed to be safe. The final transmission of the message to the recipient (the hole $Q$) includes the value of the "error counter" *ec*.

Figure 1 (right) shows a graphical view of the pNet *SimpleProtocolSpec* that specifies the system. The pNet is made of the composition of two pNets: a *SimpleSystem* node, and a *PerfectBuffer* sub-pNet. The full system implementation should be equivalent (e.g. weakly bisimilar) to this *SimpleProtocolSpec*.




Having a larger figure would help many of your readers.

Figure 1: pNet structure of the example and its specification expressed as a pNet called *SimpleProtocolSpec*

The pNet has a tree-like structure. The root node of the tree *SimpleSystem* is the top level of the pNet structure. It acts as the parallel operator. It consists of three nodes: two holes $P$ and $Q$ and one sub-pNet, denoted *PerfectBuffer*. Nodes of the tree are synchronised using four synchronisation vectors, that express the possible synchronisations between the parameterised actions of a subset of the nodes. For instance, in the vector $<$ `p-send(m)`, `in(m)`, `_` $>\!\!\rightarrow$ `in(m)` only $P$ and *PerfectBuffer* nodes are involved in the synchronisation. The synchronisation between these processes occurs when process $P$ performs `p-send(m)` action sending a message, and the *PerfectBuffer* accepts the message through an `in(m)` action at the same time; the result that will be returned at upper level is the action `in(m)`.

Figure 2 shows the pNet model of the protocol implementation, called *SimpleProtocolImpl*. When the *Medium* detects an error (modelled by a local $\tau$ action), it sends back a `m-error` message to the *Sender*. The *Sender* increments its local error counter *ec*, and resends the message (including *ec*) to the *Medium*, that will, eventually, transmit $m, ec$ to the *Receiver*.

## 3. A model of process composition

The semantics of open pNets will be defined as an open automaton. An open automaton is an automaton where each transition composes transitions of several LTSs with action of some holes, the transition occurs if some predicates hold, and can involve a set of state modifications. This section defines open automata and a bisimulation theory for them. This section is an improved version of the formalism described in [12], extending the automata with a notion of global variable, which makes the state of the automaton more explicit. We also adopt a semantics and logical interpretation of the automata that intuitively can be stated as follows: "if a transition belongs to an open automaton, any refinement of this transition also belongs to the automaton".

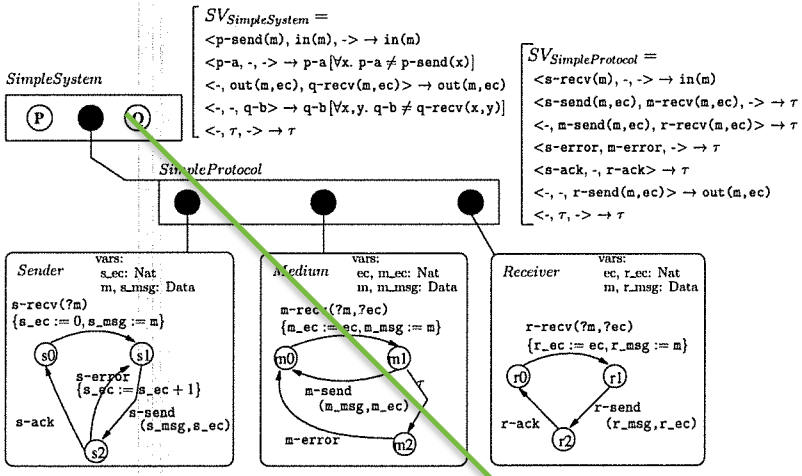



Figure 2: The *SimpleProtocolImpl* pNet resulting from the composition of the *SimpleSystem* and the *SimpleProtocol* pNets.

### *3.1. Open Automata*

Open automata (OA) are not composition structures but they are made of transitions that are dependent of the actions of the holes, and they can reason on a set of variables (potentially with only symbolic values).

**Definition 5 (Open transitions).** An *open transition* (OT) over a set $J$ of holes with sorts $Sort_j^{j\in J}$, a set $V$ of variables, and a set of states $\mathcal{S}$ is a structure of the form:

$$\frac{\beta_j^{j\in J'}, Pred, Post}{s \xrightarrow{\alpha} s'}$$

Where $J' \subseteq J$ is the set of holes involved in the transition; $s, s' \in \mathcal{S}$ are states of the automaton; and $\beta_j$ is a transition of the hole $j$, with $\text{Sort}(\beta_j) \in Sort_j$. $\alpha$ is the resulting action of this open transition. *Pred* is a predicate, *Post* is a set of assignments that are effective after the open transition, they are represented as a substitution function: $(x_k \leftarrow e_k)^{k\in K}$. Predicates and expressions of an open transition can refer to the variables in $V$, and in the different terms $\beta_j$ and $\alpha$. More precisely:

$$vars(Pred) \subseteq V \cup vars(\alpha) \cup \bigcup_{j\in J'} vars(\beta_j) \quad \wedge$$
$$\forall k.\, x_k \in V \quad \wedge \quad \forall k.\, vars(e_k) \subseteq V \cup vars(\alpha) \cup \bigcup_{j\in J'} vars(\beta_j)$$

The assignments are applied simultaneously because the variables in $V$ can be in both sides ($x_k$s are distinct). Open transitions are identified modulo logical equivalence on their predicate.

It is important to understand the difference between the red dotted rule and a classical inference rule. They correspond to two different logical levels. On one side, classical (black) inference rules use an expressive logic (like any other computer science article). On the other side, open transition rules (with dotted lines) are logical implications, but using a logic with a specific syntax and that can be mechanized (this logic includes the boolean expressions $\mathbb{B}$, boolean operators, and term equality).

An open automaton is an automaton where each transition is an open transition.

**Definition 6 (Open automaton).** An *open automaton* is a structure $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$ where:

- $J$ is a set of indices.
- $\mathcal{S}$ is a set of states and $s_0$ an initial state ~~among~~ $\mathcal{S}$.
- $V$ is a set of variables of the automaton and each $v \in V$ may have an initial value $init(v)$.
- $\mathcal{T}$ is a set of open transitions and for each $t \in \mathcal{T}$ there exists $J'$ with $J' \subseteq J$, such that $t$ is an open transition over $J'$ and $\mathcal{S}$.

While the definition and usage of the open transition can be formalised and taken in a pure syntactic acceptance, we take in this article a semantics and logical understanding of open automata. ~~Formally,~~ the open transition sets in open automata are closed by a simple form of refinement that allows us to refine the predicate, or substitute any free variable by an expression. Formally, for ~~all~~ each predicate $Pred$ for ~~all~~ partial function $Post$, if $V \cap \mathrm{dom}(Post) = \emptyset$, we have:

$$\frac{\overline{\beta}, Pred', Post'}{t \xrightarrow{\alpha} t'} \in \mathcal{T} \implies \frac{\overline{\beta}\{\{Post\}\}, Pred'\{\{Post\}\} \wedge Pred, Post \odot Post'}{t \xrightarrow{\alpha\{\{Post\}\}} t'} \in \mathcal{T}$$

Because of the semantic interpretation of open automata, the set of open transition of an open automaton is infinite (for example because every free variable can be renamed). However an open automaton is characterized by a subset of these open transitions which is sufficient to generate, by substitution the other ones. In the following, we will abusively write that we define an "open automaton" when we provide only the set of open transitions that is sufficient to generate a proper open automaton by saturating each open transition by all possible substitutions and refinements.

Another aspect of the logical interpretation of the formulas is that we make no distinction between the equality and the equivalence on boolean formulas, i.e. equivalence of two predicates $Pred$ and $Pred'$ can be denoted $Pred = Pred'$, where the $=$ symbol is not interpreted in a syntactical way.



I'm not so sure I get this differently. Perhaps you can compare de Simone's rules in his TCS article with yours.

I found this a bit unclear.

is contained in

each

each

Note that there are no finiteness requirements in Def. 6. What you write here might be relevant for finite open automata.

so you consider predicates up to semantic equivalence

Though the definition is simple, the fact that transitions are complex structures relating events must not be underestimated. The first element of theory for open automata, i.e. the definition of a strong bisimulation, is given below.

### 3.2. *Bisimulation for open Automata*

The equivalence we need is a strong bisimulation between open automata having exactly the same holes (same indices and same sorts), but using a flexible matching between open transitions, this will allow us to compare pNets with different architectures.

We define now a bisimulation relation tailored to open automata and their parametric nature. This relation relates states of the open automata and guarantees that the related states are observationally equivalent, i.e. equivalent states can trigger transitions with identical action labels. Its key characteristics are 1) the introduction of predicates in the bisimulation relation: the relation between states may depend on the value of the variables; 2) the bisimulation property relates elements of the open transitions and takes into account predicates over variables, actions of the holes, and state modifications. We name it FH-bisimulation, as a short cut for the "Formal Hypotheses" over the holes behaviour manipulated in the transitions, but also as a reference to the work of De Simone [1], that pioneered this idea.

One of the original aspects of FH-bisimulation is due to the symbolic nature of open automata. Indeed, a single state of the automaton represents a potentially infinite number of concrete states, depending on the value of the automaton variables, and a single open transition of the automaton may also be instantiated with an unbounded number of values for the transition parameters. Consequently it would be too restrictive to impose that each transition of one automaton is matched by exactly one transition of the bisimilar automaton. Thus the definition of bisimulation requires that, for each open transition of one automaton, there exists a matching set of open transitions covering the original one. Indeed depending on the value of action parameters or automaton variables, different open transitions might simulate the same one.

The parametric nature of the automata entails a second original aspect of FH-bisimulation: the nature of the bisimulation relation itself. A classical relation between states can be seen as a function mapping pairs of state to a boolean value (true if the states are related, false if they are not). An FH-bisimulation relation maps pairs of states to boolean expressions that use variables of the two systems. Formally, a relation over the states of two open automata $\langle\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1\rangle\rangle$ and $\langle\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2\rangle\rangle$ has the signature $\mathcal{S}_1 \times \mathcal{S}_2 \to \mathbb{B}$. We suppose without loss of generality that the variables of the two open automata are disjoint. We adopt a notation similar to standard relations and denote it $\mathcal{R} = \{(s,t|Pred_{s,t})\}$, where: 1) For any pair $(s,t) \in \mathcal{S}_1 \times \mathcal{S}_2$, there is a single $(s,t|Pred_{s,t}) \in \mathcal{R}$ stating that $s$ and $t$ are related if $Pred_{s,t}$ is True, i.e. the states are related when the value of the automata variables verify the predicate $Pred_{s,t}$. 2) The free variables of $Pred_{s,t}$ belong to $V_1$ and $V_2$, i.e.

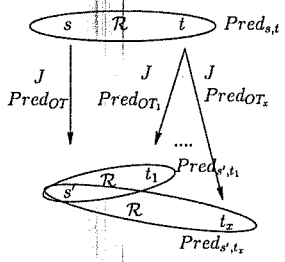
$vars(Pred_{s,t}) \subseteq V_1 \cup V_2$. FH-bisimulation is defined formally[3]:

**Definition 7 (Strong FH-bisimulation).**
Suppose $A_1 = \langle\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1\rangle\rangle$ and $A_2 = \langle\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2\rangle\rangle$ are open automata with identical holes of the same sort, with disjoint sets of variables ($V_1 \cap V_2 = \emptyset$).

Then $\mathcal{R}$ is an FH-bisimulation if and only if for any states $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$, $(s, t|Pred_{s,t}) \in \mathcal{R}$, we have the following:

- For any open transition $OT$ in $\mathcal{T}_1$:

$$\frac{\beta_j^{j\in J'}, Pred_{OT}, Post_{OT}}{s \xrightarrow{\alpha} s'}$$




  there exists an indexed set of open transitions $OT_x^{x\in X} \subseteq \mathcal{T}_2$:

$$\frac{\beta_{jx}^{j\in J_x}, Pred_{OT_x}, Post_{OT_x}}{t \xrightarrow{\alpha_x} t_x}$$

  such that $\forall x.\, J' = J_x$ and there exists $Pred_{s',t_x}$ such that $(s', t_x|Pred_{s',t_x}) \in \mathcal{R}$ and

$$Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x\in X} (\forall j.\beta_j = \beta_{jx} \wedge Pred_{OT_x} \wedge \alpha{=}\alpha_x \wedge Pred_{s',t_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})$$

- and symmetrically any open transition from $t$ in $\mathcal{T}_2$ can be covered by a set of transitions from $s$ in $\mathcal{T}_1$.

Classically, $Pred_{s',t_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}$ applies in parallel the substitution defined by the partial functions $Post_{OT}$ and $Post_{OT_x}$ (parallelism is crucial inside each $Post$ set but not between $Post_{OT}$ and $Post_{OT_x}$ that are independent), applying the assignments of the involved rules. We can prove that such a bisimulation is an equivalence relation.

**Theorem 1 (FH-Bisimulation is an equivalence).** *Suppose $\mathcal{R}$ is an FH-bisimulation. Then $\mathcal{R}$ is an equivalence, that is, $\mathcal{R}$ is reflexive, symmetric and transitive.*

The proof of this theorem can be found in Appendix A.1. The only non-trivial part of the proof is the proof of transitivity. It relies on the following elements. First, the transitive composition of two relations with predicate is defined; this is not exactly standard as it requires to define the right predicate

[3]In this article, we denote $\beta_{jx}$ a double indexed set, instead of the classical $\beta_{j,x}$. Indeed the standard notation would be too heavy in our case.

for the transitive composition and producing a single predicate to relate any two states. Then the fact that one open transition is simulated by a family of open transitions leads to a doubly indexed family of simulating open transition; this needs particular care, also because of the use of renaming (*Post*) when proving that the predicates satisfy the definition (property on $Pred_{s,t} \wedge Pred_{OT}$ in the definition).

*Finite versus infinite open automata, and decidability*

As mentioned in Definition 13, we adopt here a semantic view on open automata. More precisely, in [14], we define *semantic open automata* (infinite as in Definition 6), and *structural open automata* (finite) that can be generated as the semantics of pNets (see Definition 9), and used in the implementation. Then we define an alternative version of our bisimulation, called structural-FH-Bisimulation, based on structural open automata, and prove that the *semantic* and *structural* FH-Bisimulations coincide. In the sequel, all mentions of finite automata, and algorithms for bisimulations, implicitly refer to their *structural* versions.

If we assume that everything is finite (states and transitions in the open automata), then it is easy to prove that it is decidable whether a relation is a FH-bisimulation, provided the logic of the predicates is decidable (proof can be found in [12]). Formally:

**Theorem 2 (Decidability of FH-bisimulation).** *Let $A_1$ and $A_2$ be finite open automata and $\mathcal{R}$ a relation over their states $\mathcal{S}_1$ and $\mathcal{S}_2$ constrained by a set of predicates. Assume that the predicate inclusion is decidable over the action algebra $\mathbb{A}$. Then it is decidable whether the relation $\mathcal{R}$ is an FH-bisimulation.*

## 4. Semantics of Open pNets

This section defines the semantics of an open pNet as a translation into an open automaton. In this translation, the states of the open automata are obtained as products of the states of the pLTSs at the leaves of the composition. The predicates on the transitions are obtained both from the predicates on the transitions of the pLTSs, and from the synchronisation vectors involved in the transition.

The definition of bisimulation for open automata allows us to derive a bisimulation relation for open pNets. As pNets are composition structures, it then makes sense to prove composition lemmas: we prove that the composition of strongly bisimilar pNets are themselves bisimilar.

### *4.1. Deriving an open automaton from an open pNet*

To derive an open automaton from a pNet, we first describe the set of states of the automaton. Then we show the construction rule for transitions of the automaton, this relies on the derivation of predicates unifying synchronisation vectors and the actions of the pNets involved in a given synchronisation.

States of open pNets are tuples of states. We denote them as $\triangleleft \ldots \triangleright$ for distinguishing tuple states from other tuples.

**Definition 8 (States of open pNets).** A state of an open pNet is a tuple (not necessarily finite) of the states of its leaves.

For any pNet P, let $\text{Leaves}(P) = \langle\!\langle S_i, s_{i0}, V, \rightarrow_i \rangle\!\rangle^{i \in L}$ be the set of pLTS at its leaves, then $States(P) = \{\triangleleft s_i^{i \in L} \triangleright \mid \forall i \in L. s_i \in S_i\}$. A pLTS being its own single leave: $States(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \{\triangleleft s \triangleright \mid s \in S\}$.

The initial state is defined as: $InitState(P) = \triangleleft {s_{i0}}^{i \in L} \triangleright$.

To be precise, the state of each pLTS is entirely characterized by both the state of the automaton, and the value of its variables $V$. Consequently, the state of a pNet is not only characterized the tuple of pLTS states but also contains the value of its variables $vars(P)$.

Omit, IMHO.

*Predicates.* We define a predicate $Pred_{sv}$ relating a synchronisation vector (of the form $(\alpha_i')^{i \in I}, (\beta_j')^{j \in J} \rightarrow \alpha'[e_b]$), the actions of the involved sub-pNets and the resulting actions. This predicate verifies: satisfies

$$Pred_{sv}\Big(((\alpha_i')^{i \in I}, (\beta_j')^{j \in J} \rightarrow \alpha'[e_b]), \alpha_i^{i \in I}, \beta_j^{j \in J}, \alpha\Big) \Leftrightarrow$$
$$\forall i \in I. \alpha_i = \alpha_i' \wedge \forall j \in J. \beta_j = \beta_j' \wedge \alpha = \alpha' \wedge e_b$$

equivalently

Somehow, this predicate entails a verification of satisfiability in the sense that if the predicate $Pred_{sv}$ is not satisfiable, then the transition associated with the synchronisation will not occur in the considered state, or will occur with a *False* precondition which is equivalent. If the action families do not match or if there is no valuation of variables such that the above formula can be ensured then the predicate is undefined.

The definition of this predicate is not constructive but it is easy to build the predicate constructively by brute-force unification of the sub-pNets actions with the corresponding vector actions, possibly followed by a simplification step.

what is the complexity of this "brute-force" approach?

**Example 1 (An open-transition).** At the upper level, the *SimpleSystem* pNet of Figure 2 has 2 holes and *SimpleProtocol* as a sub-pNet, itself containing 3 pLTSs. One of its possible open transitions (synchronizing the hole $P$ with the *Sender* within the *SimpleProtocol*) is:

$$OT_1 = \frac{\{\texttt{P}\mapsto\texttt{p-send(m)}\}, [\texttt{m=m'}], (\texttt{s\_msg} \leftarrow \texttt{m})}{\triangleleft s_0, m_0, r_0 \triangleright \xrightarrow{\texttt{in(m')}} \triangleleft s_1, m_0, r_0 \triangleright}$$

The global states here are triples, the product of states of the 3 pLTSs (holes have no state). The assignment performed by the open transition uses the variable `m` from the action of hole `P` to set the value of the sender variable named `s_msg`.

We build the semantics of open pNets as an open automaton over the states given by Definition 8. The open transitions first project the global state into states of the leaves, then apply pLTS transitions on these states, and compose them with the sort of the holes. The semantics instantiates fresh variables using the predicate fresh($x$), additionally, for an action $\alpha$, fresh($\alpha$) means all variables in $\alpha$ are fresh.

**Definition 9 (Semantics of open pNets).** The semantics of a pNet $P$ is an open automaton $A = \langle\!\langle \mathit{Holes}(P), \mathit{States}(P), \mathit{InitState}(P), \mathit{vars}(P), \mathcal{T} \rangle\!\rangle$ where $\mathcal{T}$ is the smallest set of open transitions such that $\mathcal{T} = \{OT \mid P \models OT\}$ and $P \models OT$ is defined by the following rules:

- The rule for a pLTS checks that the guard is verified and transforms assignments into post-conditions:

$$\frac{s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j \in J} \rangle} s' \in \rightarrow}{\langle\!\langle S, s_0, \rightarrow \rangle\!\rangle \models \lhd s \rhd \xrightarrow{\alpha}{\scriptstyle \emptyset, e_b, \{x_j \leftarrow e_j\}^{j \in J}} \lhd s' \rhd} \quad \textbf{Tr1}$$

- The second rule deals with pNet nodes: for each possible synchronisation vector (of index $k$) applicable to the rule subject, the premisses include one *open transition* for each sub-pNet involved, one possible *action* for each hole involved, and the predicate relating these with the resulting action of the vector. The sub-pNets involved are split between two sets, $I_2$ for sub-pNets that are pLTSs (with open transitions obtained by rule **Tr1**), and $I_1$ for the sub-pNets that are not pLTSs (with open transitions obtained by rule **Tr2**), $J$ is the set of holes involved in the transition[4][5].

[4]Formally, if $SV_k = (\alpha')_m^{m \in M} \rightarrow \alpha'[e_b]$ is a synchronisation vector of P then $J = M \cap \mathrm{Holes}(P)$, $I_2 = M \cap \mathrm{Leaves}(P)$, $I_1 = M \setminus J \setminus I_2$

[5]We could replace $I_1$ and $I_2$ by their formal definition in **Tr2** but the rule would be more difficult to read.

$$\frac{\begin{array}{c}\text{Leaves}(\langle\!\langle P_m^{m\in I}, \overline{Sort}, SV_k^{k\in K}\rangle\!\rangle) = pLTS_l^{l\in L} \quad k\in K \\ SV_k = (\alpha'_m)^{m\in I_1\uplus I_2\uplus J} \rightarrow \alpha'[e_b] \\ \forall m\in I_1.\, P_m \models \dfrac{\beta_j^{j\in J_m}, Pred_m, Post_m}{\lhd s_i^{i\in L_m}\rhd \xrightarrow{\alpha_m} \lhd (s'_i)^{i\in L_m}\rhd} \\ \forall m\in I_2.\, P_m \models \dfrac{\emptyset, Pred_m, Post_m}{\lhd s_m\rhd \xrightarrow{\alpha_m} \lhd s'_m\rhd} \qquad J' = \biguplus_{m\in I_1} J_m \uplus J \\ Pred = \bigwedge_{m\in I_1\uplus I_2} Pred_m \wedge Pred_{sv}(SV_k, \alpha_m^{m\in I_1\uplus I_2}, \beta_j^{j\in J}, \alpha) \\ \forall i\in L\setminus\left(\biguplus_{m\in I_1} L_m \uplus I_2\right).\, s'_i = s_i \qquad \text{fresh}(\alpha'_m, \alpha', \beta_j^{j\in J}, \alpha)\end{array}}{\langle\!\langle P_m^{m\in I}, \overline{Sort}, SV_k^{k\in K}\rangle\!\rangle \models \dfrac{\beta_j^{j\in J'}, Pred, \biguplus_{m\in I_1\uplus I_2} Post_m}{\lhd s_i^{i\in L}\rhd \xrightarrow{\alpha} \lhd (s'_i)^{i\in L}\rhd}} \textbf{Tr2}$$

I'd place this in a figure to improve typesetting and the use of space on page 18.

A key to understand this rule is that the open transitions are expressed in terms of the leaves and holes of the whole pNet structure, i.e. a flatten view of the pNet. For example, $L$ is the index set of the Leaves, $L_m$ the index set of the leaves of one sub-pNet indexed $m$, so all $L_m$ are disjoint subsets of $L$. Thus the states in the open transitions, at each level, are tuples including states of all the leaves of the pNet, not only those involved in the chosen synchronisation vector.

Note that the construction is symbolic, and each open transition deduced expresses a whole family of behaviours, for any possible value of the variables.

In [12], we have shown a detailed example of the construction of a complex open transition, building a deduction tree using rules **Tr1** and **Tr2**. We have also shown in [12] that an open pNet with finite synchronisation sets, finitely many leaves and holes, and each pLTS at leaves having a finite number of states and (symbolic) transitions, ~~has~~ a finite automaton. The algorithm for building such an automaton can be found in [15].

is translated into / indexes

*Example*

Figure 3 shows the open automaton computed from the *SimpleProtocolSpec* pNet given in Figure 1. For later references, we name $SS_i$ the transitions of this (strong) specification automaton while transitions of the *SimpleProtocolImpl* pNet are labelled $SI_i$. In the figures we annotate each open automaton with the set of its variables.

Figure 4 shows the open automaton of *SimpleProtocolImpl* from Figure 2. In this drawing, we have short labels for states, representing $\lhd s_0, m_0, r_0\rhd$ by 000. Note that open transitions are denoted $\texttt{SI}_i$ and tau open transition by $\texttt{SI}_\tau$. The resulting behaviour is quite simple: we have a main loop including receiving a message from $P$ and transmitting the same message to $Q$, with some intermediate $\tau$ actions from the internal communications between the protocol processes.

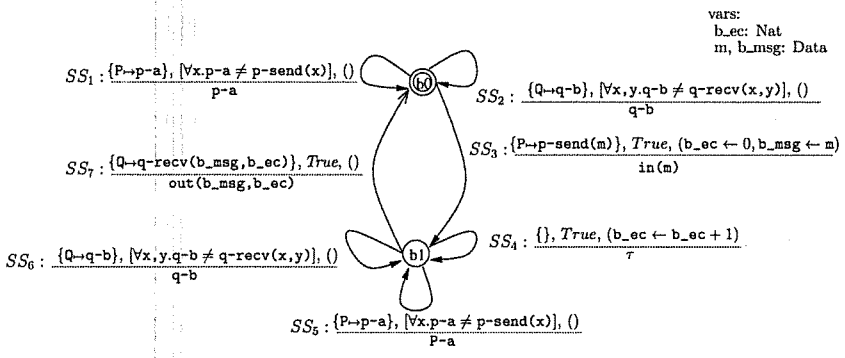



Figure 3: Open automaton for *SimpleProtocolSpec*

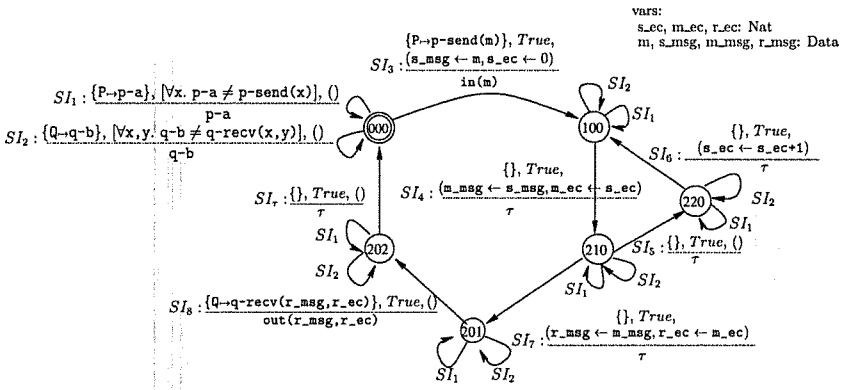



Figure 4: Open automaton for *SimpleProtocolImpl*

In most of the transitions, you can observe that data is propagated between the successive pLTS variables (holding the message, and the error counter value). On the right of the figure, there is a loop of $\tau$ actions ($\mathtt{SI}_4$, $\mathtt{SI}_5$ and $\mathtt{SI}_6$) showing the handling of errors and the incrementation of the error counter.

### *4.2. pNet Composition Properties: composition of open transitions*

The semantics of open pNets allows us to prove two crucial properties relating pNet composition with pNet semantics: open transition of a composed pNet can be decomposed into open transitions of its composing sub-pNets, and conversely, from the open transitions of sub-pNets, an open transition of the composed pNet can be built.

We start with a decomposition property: from one open transition of $P[Q]_{j_0}$, we exhibit corresponding behaviours of $P$ and $Q$, and determine the relation between their predicates.

**Lemma 1 (Open transition decomposition).** *Consider two pNets $P$ and $Q$ that are not pLTSs*[6]*. Let* $\text{Leaves}(Q) = p_l^{l \in L_Q}$ *and suppose:*

$$P[Q]_{j_0} \models \frac{\beta_j^{j \in J}, Pred, Post}{\lhd s_i^{i \in L} \rhd \xrightarrow{\alpha} \lhd {s'}_i^{i \in L} \rhd}$$

*with* $J \cap \text{Holes}(Q) \neq \emptyset$ *or* $\exists i \in L_Q . s_i \neq s'_i$*, i.e. $Q$ takes part in the reduction. Then there exist $\alpha_Q$, $Pred'$, $Pred''$, $Post'$, $Post''$ s.t.:*

$$P \models \frac{\beta_j^{j \in (J \setminus \text{Holes}(Q)) \cup \{j_0\}}, Pred', Post'}{\lhd s_i^{i \in L \setminus L_Q} \rhd \xrightarrow{\alpha} \lhd {s'}_i^{i \in L \setminus L_Q} \rhd}$$

$$\text{and} \qquad Q \models \frac{\beta_j^{j \in J \cap \text{Holes}(Q)}, Pred'', Post''}{\lhd s_i^{i \in L_Q} \rhd \xrightarrow{\alpha_Q} \lhd {s'}_i^{i \in L_Q} \rhd}$$

*and* $Pred \iff Pred' \wedge Pred'' \wedge \alpha_Q = \beta_{j_0}$*,* $Post = Post' \uplus Post''$ *where $Post''$ is the restriction of $Post$ over variables of $Q$.*

Lemma 2 is combining an open transition of $P$ with an open transition of $Q$, and building a corresponding transition of $P[Q]_{j_0}$ by assembling their elements.

**Lemma 2 (Open transition composition).** *Suppose* $j_0 \in J$ *and:*

$$P \models \frac{\beta_j^{j \in J}, Pred, Post}{\lhd s_i^{i \in L} \rhd \xrightarrow{\alpha} \lhd {s'}_i^{i \in L} \rhd} \qquad \text{and} \qquad Q \models \frac{\beta_j^{j \in J_Q}, Pred', Post'}{\lhd s_i^{i \in L_Q} \rhd \xrightarrow{\alpha_Q} \lhd {s'}_i^{i \in L_Q} \rhd}$$

*Then, we have:*

$$P[Q]_{j_0} \models \frac{\beta_j^{(j \in J \setminus \{j_0\}) \uplus J_Q}, Pred \wedge Pred' \wedge \alpha_Q = \beta_{j_0}, Post \uplus Post'}{\lhd s_i^{i \in L \uplus L_Q} \rhd \xrightarrow{\alpha} \lhd {s'}_i^{i \in L \uplus L_Q} \rhd}$$

Note that this does not mean that any two pNets can be composed and produce an open transition. Indeed, the predicate $Pred \wedge Pred' \wedge \alpha_Q = \beta_{j_0}$ is often not satisfiable, in particular if the action $\alpha_Q$ cannot be matched with $\beta_{j_0}$. Note also that $\beta_{j_0}$ is only used as an intermediate term inside formulas in the composed open transition: it does not appear as global action, and will not appear as an action of a hole.

*4.3. Bisimulation for open pNets – a composable bisimulation theory*

As our symbolic operational semantics provides an open automaton, we can apply the notion of strong (symbolic) bisimulation on automata to open pNets.

[6] A similar lemma can be proven for a pLTS $Q$

**Definition 10 (FH-bisimulation for open pNets).** Two pNets are FH-bisimilar if there exists a relation between their associated automata that is an FH-bisimulation and their initial states are in the relation (i.e. the predicate associated with the initial states is verifiable).

We can now prove that pNet composition preserves FH-bisimulation. More precisely, one can define two preservation properties, namely 1) when one hole of a pNet is filled by two bisimilar other (open) pNets; and 2) when the same hole in two bisimilar pNets are filled by the same pNet, in other words, composing a pNet with two bisimilar contexts. The general case will be obtained by transitivity of the bisimulation relation (Theorem 1).

**Theorem 3 (Congruence).** *Consider an open pNet:* $P = \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$. *Let* $j_0 \in J$ *be a hole. Let* $Q$ *and* $Q'$ *be two FH-bisimilar pNets such that*[7] $\mathrm{Sort}(Q) = \mathrm{Sort}(Q') = Sort_{j_0}$. *Then* $P[Q]_{j_0}$ *and* $P[Q']_{j_0}$ *are FH-bisimilar.*

**Theorem 4 (Context equivalence).** *Consider two open pNets* $P = \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$ *and* $P' = \langle\!\langle P'^{i\in I}_i, Sort_j^{j\in J}, \overline{SV'}\rangle\!\rangle$ *that are FH-bisimilar (they thus have the same holes). Let* $j_0 \in J$ *be a hole, and* $Q$ *be a pNet such that* $\mathrm{Sort}(Q) = Sort_{j_0}$. *Then* $P[Q]_{j_0}$ *and* $P'[Q]_{j_0}$ *are FH-bisimilar.*

Finally, the previous theorems can be composed to state a general theorem about composability and FH-bisimilarity.

**Theorem 5 (Composability).** *Consider two FH-bisimilar pNets with an arbitrary number of holes, when replacing, inside those two original pNets, a subset of the holes by FH-bisimilar pNets, we obtain two FH-bisimilar pNets.*

This theorem is quite powerful, as it somehow implies that the theory of open pNets is convenient to study properties of process composition. Open pNets can indeed be used to study process operators and process algebras, as shown in [12] where compositional properties are extremely useful. In the case of interaction protocols [16], composition of bisimulation can justify abstractions used in some parts of the application.

## 5. Weak bisimulation

Weak symbolic bisimulation was introduced to relate transition systems that have indistinguishable behaviour, with respect to some definition of *internal actions* that are considered local to some subsystem, and consequently cannot be observed, nor used for synchronisation with their context. The notion of non-observable actions varies in different contexts, e.g. *tau* in CCS, and *i* in Lotos, we could define classically a set of *internal/non-observable actions* depending

[7] Note that $\mathrm{Sort}(Q) = \mathrm{Sort}(Q')$ is ensured by strong bisimilarity.

on a specific action algebra. In this paper, to simplify the notations, we will simply use $\tau$ as the single non-observable action; the generalisation of our results to a set of non-observable actions is trivial. Naturally, a non-observable action cannot be synchronised with actions of other systems in its environment. We show here that under such assumption of non-observability of $\tau$ actions, see Definition 11, we can define a weak bisimulation relation that is compositional, in the sense of open pNet composition. In this section we will first define a notion of weak open transition similar to open transition. In fact a weak open transition is made of several open transitions labelled as non-observable transitions, plus potentially one observable open transition. This allows us to define weak open automata, and a weak bisimulation relation based on these weak open automata. Finally, we apply this weak bisimulation to open pNets, obtain a weak bisimilarity relation for open pNets, and prove that this relation has compositional properties.

### *5.1. Preliminary definitions and notations*

We first specify in terms of open transition, what it means for an action to be non-observable. Namely, we constraint ourselves to system where the emission of a $\tau$ action by a sub-pNet cannot be observed by the surrounding pNets. In other words, a pNet cannot change its state, or emit a specific observable action when one of its holes emits a $\tau$ action.

More precisely, we state that $\tau$ is not observable if the automaton always allows any $\tau$ transition from holes, and additionally the global transition resulting from a $\tau$ action of a hole is a $\tau$ transition not changing the pNet's state. We define $\mathrm{Id}(V)$ as the identity function on the set of variables $V$.

**Definition 11 (Non-observability of $\tau$ actions for open automata).** An open automaton $A = \langle\!\langle J, S, s_0, V, T \rangle\!\rangle$ *cannot observe* $\tau$ *actions* if and only if for all $j$ in $J$ and $s$ in $S$ we have:

1. $$\frac{(j \mapsto \tau), \mathit{True}, \mathrm{Id}(V)}{s \xrightarrow{\tau} s} \in T$$

   and

2. for all $\beta_j$, $J$, $\alpha$, $s$, $s'$, *Pred*, *Post* such that

   $$\frac{\beta_j^{j \in J}, \mathit{Pred}, \mathit{Post}}{s \xrightarrow{\alpha} s'} \in T$$

   If there exists $j$ such that $\beta_j = \tau$ then we have:

   $$\alpha = \tau \wedge s = s' \wedge \mathit{Pred} = \mathit{True} \wedge \mathit{Post} = \mathrm{Id}(V) \wedge J = \{j\}$$

The first statement of the definition states that the open automaton must allow a hole to do a silent action at any time, and must not observe it, i.e. it cannot

change its internal state because a hole did a $\tau$ transition. The second statement ensures that there cannot be in the open automaton other transitions that would be able to observe a $\tau$ action from a hole: statement (2) states that all the open transitions where a hole does a $\tau$ action must be of the shape given in statement (1). The condition $J = \{j\}$ is a bit restrictive, it could safely be replaced by $\forall j \in J. \beta_j = \tau$, allowing the other holes to perform $\tau$ transitions too (because these $\tau$ actions cannot be observed).

By definition, one weak open transition contains several open transitions, where each open transition can require an observable action from a given hole, the same hole might have to emit several observable actions for a single weak open transition to occur. Consequently, for a weak open transition to trigger, a sequence of actions from a given hole may be required.

Thus, we let $\gamma$ range over sequences of action terms and use $\uplus$ as the concatenation operator that appends sequences of action terms: given two sequences of action terms $\gamma \uplus \gamma'$ concatenates the two sequences. The operation is lifted to indexed sets of sequences: at each index $i$, $\overline{\gamma_1} \uplus \overline{\gamma_2}$ concatenates the sequences of actions at index $i$ of $\overline{\gamma_1}$ and the one at index $i$ of $\overline{\gamma_2}$[8]. $[a]$ denotes a sequence with a single element.

As required actions are now sequences of observable actions, we need an operator to build them from set of actions that occur in open transitions, i.e. an operator that takes a set of actions performed by one hole and produces a sequence of observable actions.

Thus we define $(\overline{\beta})^{\nabla}$ as the mapping $\overline{\beta}$ with only observable actions of the holes in $I$, but where each element is either empty or a list of length 1:

$$(\beta_i^{i \in I})^{\nabla} = [\beta_i]^{i \in I'} \text{ where } I' = \{i | i \in I \wedge \beta_i \neq \tau\}$$

As an example the $(\overline{\beta})^{\nabla}$ built from the transition $OT_1$ in Example 1, page 17 is $\mathtt{P} \mapsto [\mathtt{p\text{-}send(m)}]$. Remark that in our simple example no $\tau$ transition involves any visible action from a hole, so we have no $\beta$ sequences of length longer than 1 in the weak automaton.

### *5.2. Weak open transition definition*

Because of the non-observability property (Definition 11), it is possible to add any number of $\tau$ transitions of the holes before or after any open transition freely. This property justifies the fact that we can abstract away $\tau$ transitions from holes in the definition of a weak open transition. We define weak open transitions similarly to open transitions except that holes can perform *sequences of observable actions* instead of single actions (observable or not). Compared to the definition of open transition, this small change has a significant impact as a single weak transition is the composition of several transitions of the holes.

[8] One of the two sequences is empty when $i \notin \mathrm{dom}(\overline{\gamma_1})$ or $i \notin \mathrm{dom}(\overline{\gamma_2})$ .

**Definition 12 (Weak open transition (WOT)).** A weak open transition over a set $J$ of holes with sorts $Sort_j^{j \in J}$ and a set of states $\mathcal{S}$ is a structure of the form:

$$\frac{\gamma_j^{j \in J'}, Pred, Post}{s \overset{\alpha}{\Rightarrow} s'}$$

Where $J' \subseteq J$, $s, s' \in \mathcal{S}$ and $\gamma_j$ is a list of transitions of the hole $j$, with each element of the list in $Sort_j$. $\alpha$ is an action label denoting the resulting action of this open transition. *Pred* and *Post* are defined similarly to Definition 5. We use $\mathcal{WT}$ to range over sets of weak open transitions.

A weak open automaton $\langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT} \rangle\!\rangle$ is similar to an open automaton except that $\mathcal{WT}$ is a set of weak open transitions over $J$ and $\mathcal{S}$.

A weak open transition labelled $\alpha$ can be seen as a sequence of open transitions that are all labelled $\tau$ except one that is labelled $\alpha$; however conditions on predicates, effects, and states must be verified for this sequence to be fired.

We are now able to build a weak open automaton from an open automaton. This is done in a way that resembles the process of $\tau$ saturation: we add $\tau$ open transitions before or after another ~~(observable or not)~~ open transition.

regardless of whether it is observable or not / be it observable or not

**Definition 13 (Building a weak open automaton).**
Let $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$ be an open automaton. The weak open automaton *derived* from $A$ is an open automaton $\langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT} \rangle\!\rangle$ where $\mathcal{WT}$ is derived from $\mathcal{T}$ by saturation, applying the following rules:

$$\frac{}{\dfrac{\emptyset, True, \mathrm{Id}(V)}{s \overset{\tau}{\Rightarrow} s} \in \mathcal{WT}} \quad \textbf{WT1}$$

and

$$\frac{\dfrac{\overline{\beta}, Pred, Post}{s \xrightarrow{\alpha} s'} \in \mathcal{T}}{\dfrac{(\overline{\beta})^{\nabla}, Pred, Post}{s \overset{\alpha}{\Rightarrow} s'} \in \mathcal{WT}} \textbf{WT2}$$

and

$$\frac{\dfrac{\overline{\gamma_1}, Pred_1, Post_1}{s \overset{\tau}{\Rightarrow} s_1} \in \mathcal{WT} \qquad \dfrac{\overline{\gamma_2}, Pred_2, Post_2}{s_1 \overset{\alpha}{\Rightarrow} s_2} \in \mathcal{WT} \qquad \dfrac{\overline{\gamma_3}, Pred_3, Post_3}{s_2 \overset{\tau}{\Rightarrow} s'} \in \mathcal{WT} \qquad \overline{\gamma} = \overline{\gamma_1} \cup \overline{\gamma_2}\{\!\{Post_1\}\!\} \cup \overline{\gamma_3}\{\!\{Post_2 \odot Post_1\}\!\} \quad \alpha' = \alpha\{\!\{Post_1\}\!\} \quad Pred = Pred_1 \wedge Pred_2\{\!\{Post_1\}\!\} \wedge Pred_3\{\!\{Post_2 \odot Post_1\}\!\}}{\dfrac{\overline{\gamma}, Pred, Post_3 \odot Post_2 \odot Post_1}{s \overset{\alpha'}{\Rightarrow} s'} \in \mathcal{WT}} \textbf{WT3}$$

perform

Rule **WT1** states that it is always possible to ~~do~~ a non-observable transition, where the state is unchanged and the holes perform no action. Rule **WT2** states that each open transition can be considered as a weak open transition. The last rule is the most interesting: Rule **WT3** allows any number of $\tau$ transitions before or after any weak open transition. This rule carefully composes predicates, effects, and actions of the holes. Indeed in the rule, predicate $Pred_2$ manipulates variables of $s_1$ that result from the first weak open transition. Their values thus depend on the initial state but also on the effect (as a substitution function $Post_1$) of the first weak open transition. In the same manner, $Pred_3$ must be applied the substitution defined by the composition $Post_2 \odot Post_1$. Similarly, effects on variables must be applied to obtain the global effect of the composed weak open transition, ~~it must also be applied~~ to observable actions of the holes, and to the global action of the weak open transition.

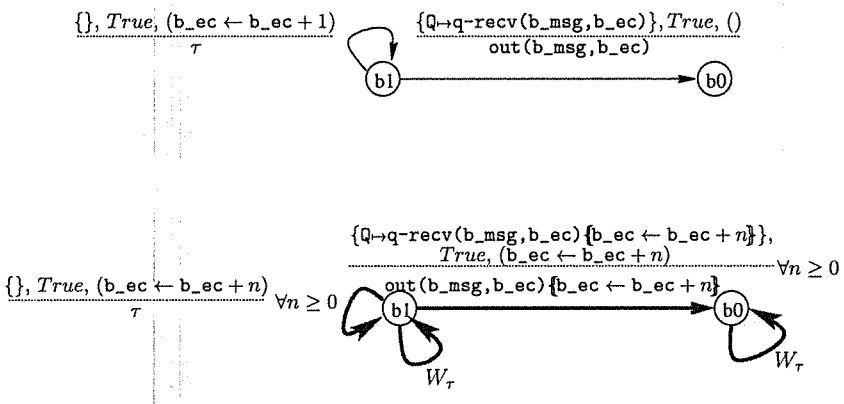



Figure 5: Construction of an example of weak open transition

**Example 2 (A weak open-transition).** Figure 5 shows the construction of one of the weak transitions of the open automaton of *SimpleProtocolSpec*. On the top we show the subset of the original open automaton (from Figure 3) considered here, and at the bottom the generated weak transition. For readability, we abbreviate the weak open transitions encoded by $\frac{\{\}, True, ()}{s \stackrel{\tau}{\Rightarrow} s'}$ as $W_\tau$. The weak open transition shown here is the transition delivering the result of the algorithm to hole $Q$ by applying rules: **WT1**,**WT2**, and **WT3**. First rule **WT1** adds a $WT_\tau$ loop on each state. Rule **WT2** transforms each 3 OTs into WOTs. Then consider application of Rule **WT3** on a sequence 3 WOTs. $\frac{\{\}, True, (\texttt{b\_ec} \leftarrow \texttt{b\_ec} + 1)}{\texttt{b1} \stackrel{\tau}{\Rightarrow} \texttt{b1}}$; $\frac{\{\}, True, (\texttt{b\_ec} \leftarrow \texttt{b\_ec} + 1)}{\texttt{b1} \stackrel{\tau}{\Rightarrow} \texttt{b1}}$; $\frac{\{\}, True, ()}{\texttt{b1} \stackrel{\tau}{\Rightarrow} \texttt{b1}}$. The result will be: $\frac{\{\}, True, (\texttt{b\_ec} \leftarrow \texttt{b\_ec} + 2)}{\texttt{b1} \stackrel{\tau}{\Rightarrow} \texttt{b1}}$. We can iterate this construction an

arbitrary number of times, getting for any natural number $n$ a weak open transition: $\dfrac{\emptyset, True, (\texttt{ec} \leftarrow \texttt{ec} + n)}{\texttt{b1} \stackrel{\tau}{\Rightarrow} \texttt{b1}} \forall n \geq 0$. Finally, applying again **WT3**, and using the central open transition having `out(b_msg,b_ec)` as $\alpha$, we get the resulting weak open transition between b1 and b0 (as shown in Figure 5). Applying the substitutions finally yields the weak transitions family $WS_7$ in Figure 6.

**Example 3 (Weak open automata).** Figures 6 and 7 respectively show the weak automata of *SimpleProtocolSpec* and *SimpleProtocolImpl*. We encode weak open transitions by $WS$ on the specification model and by $WI$ on the implementation model.

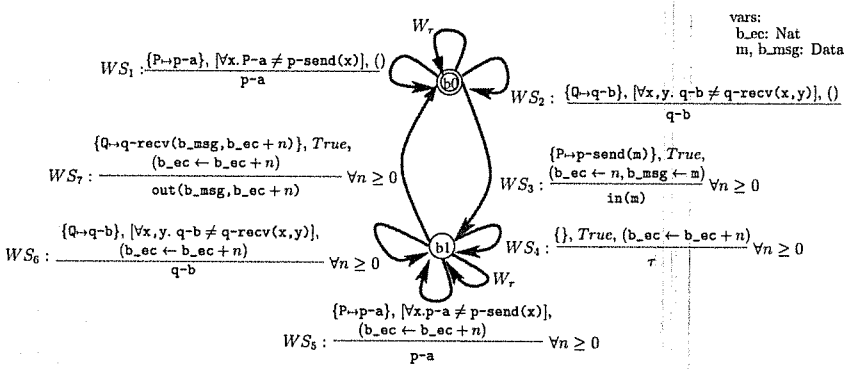



Figure 6: Weak Open Automaton of *SimpleProtocolSpec*

For readability, we only give names to the weak open transitions of *SimpleProtocolImpl* in Figure 7; we detail some of these transitions below and the full list is included in Appendix C. Let us point out that the weak OT loops ($WI_1$,$WI_2$ and $W_\tau$) on state 000 are also present in all other states, we did not repeat them. Additionally, many WOTs are similar, and numbered accordingly as 3, 3a, 3b, 3c and 8, 8a, 8b, 8c respectively: they only differ by their respective source or target states; the "variant" WOTs appear in blue in Figure 7.

Now let us give some details about the construction of the weak automaton of the *SimpleProtocolImpl* pNet, obtained by application of the weak rules as explained above. We concentrate on weak open transitions $WI_3$ and $WI_4$. Let us denote as $post_n$ the effect (as a substitution function) of the strong open transitions $SI_n$ from Figure 4:

$post_3 = (\texttt{s\_msg} \leftarrow \texttt{m}, \texttt{s\_ec} \leftarrow 0)$
$post_4 = (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec})$
$post_5 = ()$
$post_6 = (\texttt{s\_ec} \leftarrow \texttt{s\_ec+1})$

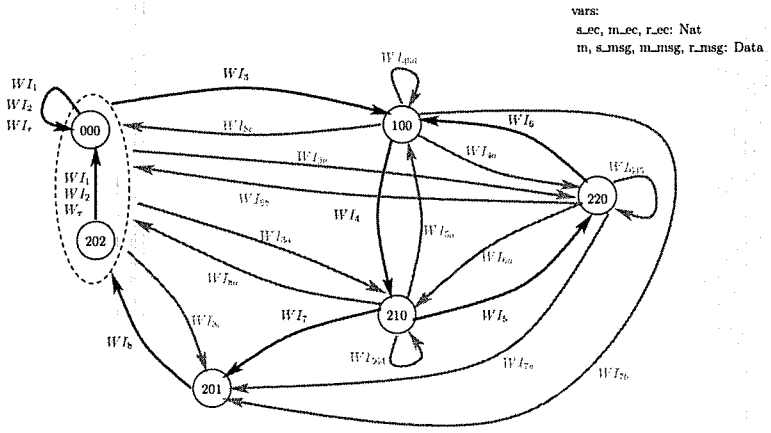



Figure 7: Weak Open Automaton of *SimpleProtocolImpl*

Then the effect of one single $100 \xrightarrow{OT_4} 210 \xrightarrow{OT_5} 220 \xrightarrow{OT_6} 100$ loop is[9]:

$$post_{456} = post_6 \odot post_5 \odot post_4 = (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1)$$

So if we denote $post_{456*}$ any iteration of this loop, we get $post_{456*} = (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)$ for any $n \geq 0$, and the *Post* of the weak OT $WI_3$ is:
$Post_3 = post_{456*} \odot post_3 = (\texttt{s\_msg} \leftarrow m, \texttt{s\_ec} \leftarrow n), \forall n \geq 0$ and *Post* of $WI_{3a}$ is:
$post_4 \odot post_{456*} \odot post_3 = (\texttt{m\_msg} \leftarrow m, \texttt{m\_ec} \leftarrow n), \forall n \geq 0.$

We can now show some of the weak OTs of Figure 7 (the full table is included in Appendix C). As we have seen above, the effect of rule $WT_3$ when a silent action have an effect on the variable *ec* will generate an infinite family of WOTs, depending on the number of iterations through the loops. We denote these families using a "meta-variable" $n$, ranging over `Nat`.

$$WI_1 = \frac{\{\texttt{P} \mapsto \texttt{p-a}\}, [\forall \texttt{x.p-a} \neq \texttt{p-send(x)}], ()}{s \xrightarrow{\texttt{p-a}} s} \text{ (for any } s \in S)$$

$$\forall n \geq 0.\ WI_3(n) = \frac{\{\texttt{P} \mapsto \texttt{p-send(m)}\},\ True,\ (\texttt{s\_msg} \leftarrow \texttt{m}, \texttt{s\_ec} \leftarrow n)}{000 \xrightarrow{\texttt{in(m)}} 100}$$

$$\forall n \geq 0.\ WI_4(n) = \frac{\{\},\ True,\ (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec} + n, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \xRightarrow{\tau} 210}$$

$$\forall n \geq 0.\ WI_{456}(n) = \frac{\{\},\ True,\ (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \xRightarrow{\tau} 100}$$

[9] when showing the result of *Posts* composition, we will omit the identity substitution functions introduced by the $\odot$ definition in page 6

The *Post* of the weak OT $WI_{6a}$ is:

$$\begin{aligned} Post_{6a} &= post_4 \odot post_{456*} \odot post_6 \\ &= (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec}) \odot (\texttt{s\_ec} \leftarrow \texttt{s\_ec}+n) \odot (\texttt{s\_ec} \leftarrow \texttt{s\_ec}+1) \\ &= (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec}+1+n, \texttt{s\_ec} \leftarrow \texttt{s\_ec}+1+n) \end{aligned}$$

So we get:

$$\forall n \geq 0.\, WI_{6a}(n) = \frac{\{\}, True, (\texttt{m\_ec} \leftarrow \texttt{s\_ec}+1+n, \texttt{s\_ec} \leftarrow \texttt{s\_ec}+1+n)}{220 \stackrel{\tau}{\Rightarrow} 210}$$

*5.3. Composition properties: composition of weak open transitions*

We now have two different semantics for open pNets: a strong semantics, defined as an open automaton, and as a weak semantics, defined as a weak open automaton. Like the open automaton, the weak open automaton features valuable composition properties. We can exhibit a composition property and a decomposition property that relate open pNet composition with their semantics, defined as weak open automata. These are however technically more complex than the ones for open automata because each hole performs a set of actions, and thus a composed transition is the composition of one transition of the top-level pNet and a sequence of transitions of the sub-pNet that fills its hole. They can be found as Lemmas 6, Lemma 7, and Lemma 8 in Appendix B.2.

what does "they" refer to?

*5.4. Weak FH-bisimulation*

One option is that

For defining a bisimulation relation between weak open automata, two options are possible. ~~Either~~ we define a simulation similar to the strong simulation but based on open automata; this would look like the FH-simulation but would need to be adapted to weak open transitions. ~~Or~~ we define directly and classically a weak FH-simulation as a relation between two ~~open automata~~, relating the open transition of the first one with the transition of the weak open automaton derived from the second one.

could

Alternatively,

The definition below specifies how a set of weak open transitions can simulate an open transition, and under which condition; this is used to relate, by weak FH-bisimulation, two open automata by reasoning on the weak open automata that can be derived from the strong ones. ~~This is defined formally as follows.~~

**Definition 14 (Weak FH-bisimulation).**
Let $A_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1 \rangle\!\rangle$ and $A_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2 \rangle\!\rangle$ be open automata with disjoint sets of variables. Let $\langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{WT}_1 \rangle\!\rangle$ and $\langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{WT}_2 \rangle\!\rangle$ be the weak open automata derived from $A_1$ and $A_2$ respectively. Let $\mathcal{R}$ a relation over $\mathcal{S}_1$ and $\mathcal{S}_2$, as in Definition 7.

Then $\mathcal{R}$ is a weak FH-bisimulation iff for any states $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$ such that $(s, t|Pred_{s,t}) \in \mathcal{R}$, we have the following:

- For any open transition $OT$ in $\mathcal{T}_1$:

$$\frac{\beta_j^{j \in J'}, Pred_{OT}, Post_{OT}}{s \xrightarrow{\alpha} s'}$$

there exists an indexed set of weak open transitions $WOT_x^{x \in X} \subseteq \mathcal{WT}_2$:

$$\frac{\gamma_{jx}^{j \in J_x}, Pred_{OT_x}, Post_{OT_x}}{t \stackrel{\alpha_x}{\Longrightarrow} t_x}$$

such that $\forall x. \{j \in J' | \beta_j \neq \tau\} = J_x, (s', t_x | Pred_{s',t_x}) \in \mathcal{R}$; and

$Pred_{s,t} \wedge Pred_{OT} \implies$

$$\bigvee_{x \in X} \left( \forall j \in J_x . (\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{OT_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t_x} \{\!\{ Post_{OT} \uplus Post_{OT_x} \}\!\} \right)$$

- and symmetrically any open transition from $t$ in $\mathcal{T}_2$ can be covered by a set of weak transitions from $s$ in $\mathcal{WT}_1$.

Two pNets are weak FH-bisimilar if there exists a relation between their associated automata that is a weak FH-bisimulation and their initial states are in the relation, i.e. the predicate associated to the relation between the initial states is *True*.

Compared to strong bisimulation, except the obvious use of weak open transitions to simulate an open transition, the condition on predicate is slightly changed concerning actions of the holes. Indeed only the visible actions of the holes must be compared and they form a list of actions, but of length at most one.

Our first important result is that weak FH-bisimilarity is an equivalence in the same way as strong FH-bisimilarity.

**Theorem 6 (Weak FH-Bisimulation is an equivalence).** *Suppose $\mathcal{R}$ is a weak FH-bisimulation. Then $\mathcal{R}$ is an equivalence, that is, $\mathcal{R}$ is reflexive, symmetric and transitive.*

The proof is detailed in Appendix B.1, it follows a similar pattern as the proof that strong FH-bisimulation is an equivalence, but technical details are different, and in practice we rely on a variant of the definition of weak FH-bisimilarity; this equivalent version simulates a *weak* open transition with a set of weak open transition. The careful use of the best definition of weak FH-bisimilarity makes the proof similar to the strong FH-bisimulation case.

*Proving bisimulation in practice*

In practice, we are dealing with finite representations of the (infinite) open automata. In [14], we defined a slightly modified definition of the "coverage" proof obligation, in the case of strong FH-Bisimulation. This modification is required to manage in a finite way all possible instantiations of an OT. In the case of weak FH-Bisimulation, the proof obligation from Definition 14 becomes:

$$\forall fv_{OT}. \left\{ Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x \in X} \left[ \exists fv_{OT_x}. \left( \forall j \in J_x . (\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{OT_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t_x} \{\!\{ Post_{OT} \uplus Post_{OT_x} \}\!\} \right) \right] \right\}$$

where $fv_{OT}$ denotes the set of free variables of all expressions in $OT$.

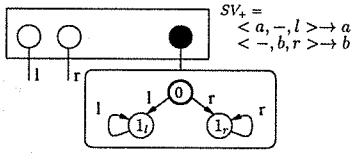
*5.5. Weak bisimulation for open pNets*

Before defining a weak open automaton for the semantics of open pNets, it is necessary to state under which condition a pNet is unable to observe silent actions of its holes. In the setting of pNets this can simply be expressed as a condition on the synchronisation vectors. Precisely, the set of synchronisation vectors must contain vectors that let silent actions go through the pNet, i.e. synchronisation vectors where one hole does a $\tau$ transition, and the global visible action is a $\tau$. Additionally, no other synchronisation vector must be able to react on a silent action from a hole, i.e. if a synchronisation vector observes a $\tau$ from a hole it cannot synchronise it with another action nor emit an action that is not $\tau$. This is formalised as follows:

**Definition 15 (Non-observability of silent actions for pNets).** A pNet $\langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$ *cannot observe silent actions* if it verifies: $\forall i \in I \uplus J.\, (i{\mapsto}\tau) \rightarrow \tau[True] \in \overline{SV}$ and

$$\forall \left( (\alpha_i)^{i\in I'} \rightarrow \alpha'[e_b] \in \overline{SV} \right), \forall i \in I' \cap J.\, \alpha_i = \tau \implies \alpha' = \tau \wedge I' = \{i\}$$

**Example 4 (CCS choice (counter-example)).** Here is the encoding of a choice operator.




The left hole is indexed $l$ the right hole $r$. The third subnet, contains an LTS encoding the control part. We obtain the specific behaviour with the synchronisation vector. 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.

This pNet does not satisfy Definition 15. Indeed, if $a$ or $b$ is $\tau$ then the $+$ operator can indeed observe the $\tau$ transition. On the other side, the parallel operator defined similarly *satisfies* Definition 15.

With this definition, it is easy to check that the open automaton that gives the semantics of such an open pNet cannot observe silent actions in the sense of Definition 11.

**Property 1 (Non-observability of silent actions).** *The semantics of a pNet, as provided in Definition 9, that cannot observe silent actions is an open automaton that cannot observe silent actions.*

Under this condition, it is safe to define the weak open automaton that provides a weak semantics to a given pNet. This is simply obtained by applying Definition 13 to generate a weak open automaton from the open automaton that is the strong semantics of the open pNet, as provided by Definition 9.

**Definition 16 (Semantics of pNets as a weak open automaton).** Let $A$ be the open automaton expressing the semantics of an open pNet $P$; let $\langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT} \rangle\!\rangle$ be the weak open automaton derived from $A$; we call this weak open automaton the weak semantics of the pNet $P$. Then, we denote $P \models \mathit{WOT}$ whenever $\mathit{WOT} \in \mathcal{WT}$.

From the definition of the weak open automata of pNets, we can now study the properties of weak bisimulation concerning open pNets.

*5.6. Properties of weak bisimulation for open pNets*

When silent actions cannot be observed, weak bisimulation is a congruence for open pNets: if $P$ and $Q$ are weakly bisimilar to $P'$ and $Q'$ then the composition of $P$ and $Q$ is weakly bisimilar to the composition of $P'$ and $Q'$, where composition is the hole replacement operator: $P[Q]_j$ and $P'[Q']_j$ are weak FH-bisimilar. This can be shown by proving the two following theorems. The detailed proof of these theorem can be found in Appendix B.2. The proof strongly relies on the fact that weak FH-bisimulation is an equivalence, but also on the composition properties for open automata.

**Theorem 7 (Congruence for weak bisimulation).** *Consider an open pNet $P$ that cannot observe silent actions, of the form $P = \langle\!\langle P_i^{i \in I}, \mathit{Sort}_j^{j \in J}, \overline{SV} \rangle\!\rangle$. Let $j_0 \in J$ be a hole. Let $Q$ and $Q'$ be two weak FH-bisimilar pNets such that*[10] $\mathrm{Sort}(Q) = \mathrm{Sort}(Q') \subseteq \mathit{Sort}_{j_0}$. *Then $P[Q]_{j_0}$ and $P[Q']_{j_0}$ are weak FH-bisimilar.*

**Theorem 8 (Context equivalence for weak bisimulation).** *Consider two open pNets $P = \langle\!\langle P_i^{i \in I}, \mathit{Sort}_j^{j \in J}, \overline{SV} \rangle\!\rangle$ and $P' = \langle\!\langle P'^{i \in I}_i, \mathit{Sort}_j^{j \in J}, \overline{SV'} \rangle\!\rangle$ that are weak FH-bisimilar (recall they must have the same holes to be bisimilar) and that cannot observe silent actions. Let $j_0 \in J$ be a hole, and $Q$ be a pNet such that* $\mathrm{Sort}(Q) \subseteq \mathit{Sort}_{j_0}$. *Then $P[Q]_{j_0}$ and $P'[Q]_{j_0}$ are weak FH-bisimilar.*

Finally, the previous theorems can be composed to state a general theorem about composability and weak FH-bisimilarity.

**Theorem 9 (Composability of weak bisimulation).** *Consider two weak FH-bisimilar pNets with an arbitrary number of holes, such that the two pNets cannot observe silent actions. When replacing, inside those two original pNets, a subset of the holes by weak FH-bisimilar pNets, we obtain two weak FH-bisimilar pNets.*

**Example 5 (CCS Choice).** Consider the + operator of CCS, shown in Example 4. It is well-known that weak bisimulation is not a congruence in CCS, and this is reflected here because we have shown that the + operator can observe the $\tau$ transitions. Thus, even if we can define a weak bisimulation for CCS with + it does not verify the necessary requirements for being a congruence.

I'm unsure about the need for / usefulness of this example.

[10] Note that $\mathrm{Sort}(Q) = \mathrm{Sort}(Q')$ is ensured by weak bisimilarity.

*Running example*

In Section 5 we have shown the full saturated weak automaton for both *SimpleProtocolSpec* and *SimpleProtocolImpl*. We will show here how we can check if some given relation between these two automata is a weak FH-Bisimulation.

Preliminary remarks:

- Both pNets trivially verify the "non-observability" condition: the vectors having $\tau$ as an action of a sub-net are of the form "$< -, \tau, - > \rightarrow \tau$".
- We must take care of variable name conflicts: in our example, the variables of the 2 systems already have different names, but the action parameters occurring in the transitions (`m`, `msg`, `ec`) are the same, that is not correct. In the tools, this is managed by the static semantic layer; in the example, we rename the only conflicting variables $m$ into $m1$ for *SimpleProtocolSpec*, and $m2$ for *SimpleProtocolImpl*.

Now consider the relation $\mathcal{R}$ defined by the following triples:

| *SimpleProtocolSpec* states | *SimpleProtocolImpl* states | Predicate |
|---|---|---|
| b0 | 000 | True |
| b0 | 202 | True |
| b1 | 100 | `b_msg = s_msg` $\wedge$ `b_ec = s_ec` |
| b1 | 210 | `b_msg = m_msg` $\wedge$ `b_ec = m_ec` |
| b1 | 220 | `b_msg = s_msg` $\wedge$ `b_ec = s_ec` |
| b1 | 201 | `b_msg = r_msg` $\wedge$ `b_ec = r_ec` |

Checking that $\mathcal{R}$ is a weak FH-Bisimulation means checking, for each of these triples, that each (strong) OT of one the states corresponds to a set of WOTs of the other, using the conditions from Definition 14. We give here one example: consider the second triple from the table, and transition $SS_3$ from state b0. Its easy to guess that it will correspond to $WI_3(0)$ of state 202.

$$SS_3 = \frac{\{\texttt{P}\mapsto\texttt{p-send(m1)}\}, True, (\texttt{b\_msg} \leftarrow \texttt{m1}, \texttt{b\_ec} \leftarrow 0)}{\texttt{b0} \xrightarrow{\texttt{in(m1)}} \texttt{b1}}$$

$$WI_3(0) = \frac{\{\texttt{P}\mapsto\texttt{p-send(m2)}\}, True, (\texttt{s\_msg} \leftarrow \texttt{m2}, \texttt{s\_ec} \leftarrow 0)}{000 \xrightarrow{\texttt{in(m2)}} 100}$$

Let us check formally the conditions:

- Their sets of active (non-silent) holes is the same: $J' = J_x = \{\texttt{P}\}$.
- Triple $(\texttt{b1}, 100, \texttt{b\_msg = s\_msg} \wedge \texttt{b\_ec = s\_ec})$ is in $\mathcal{R}$.
- The verification condition
  $\forall fv_{OT}. \{Pred \wedge Pred_{OT}$
  $\Longrightarrow \bigvee_{x \in X} [\exists fv_{OT_x}. (\forall j \in J_x. (\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{OT_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})]\}$

Gives us:

$\forall$m1. $\{True \wedge True \implies \exists$m2.
$([\texttt{p-send(m1)}] = [\texttt{p-send(m2)}] \wedge True \wedge \texttt{in(m1)} = \texttt{in(m2)} \wedge$
$(\texttt{b\_msg=s\_msg} \wedge \texttt{b\_ec=s\_ec})\{\!\{(\texttt{b\_msg} \leftarrow \texttt{m1}, \texttt{b\_ec} \leftarrow 0) \uplus (\texttt{s\_msg} \leftarrow$
$\texttt{m2}, \texttt{s\_ec} \leftarrow 0)\}\!\})\}$

That is reduced to:

$\forall \texttt{m1}. \exists \texttt{m2}. (\texttt{p-send(m1)} = \texttt{p-send(m2)} \wedge \texttt{in(m1)} = \texttt{in(m2)} \wedge \texttt{m1} = \texttt{m2} \wedge 0 = 0)$

That is a tautology.

## 6. Related Works

To the best of our knowledge, there are not many research works on Weak Bisimulation Equivalences between such complicate system models (open, symbolic, data-aware, with loops and assignments). We give a brief overview of other related publications, focussing first on Open and Compositional approaches, then on Symbolic Bisimulation for data-sensitive systems.

*Open and Compositional systems*

In [17, 18], the authors investigate several methodologies for the compositional verification of software systems, with the aim to verify reconfigurable component systems. To improve scaling and compositionality, the authors decouple the verification problem that is to be resolved by a SMT (satisfiability modulo theory) solver into independent sub-problems on independent sets of variables. These works clearly highlight the interest of incremental and compositional verification in a very general setting. In our own work on open pNets, adding more structure to the composition model, we show how to enforce a compositional proof system that is more powerful than independent sets of variables. Our theory has also been encoded into an SMT solver and it would be interesting to investigate how the examples of evolving systems studied by the authors could be encoded into pNet and verified by our framework. However, the models of Johnson et al. are quite different from ours, in particular they are much less structured, and translating them is clearly outside the scope of this article. In previous work [19], we also have shown how (closed) pNet models could be used to encode and verify finite instances of reconfigurable component systems.

Methodologies for reasoning about abstract semantics of open systems can be found in [20, 21, 22], authors introduce behavioural equivalences for open systems from various symbolic approaches. Working in the setting of process calculi, some close relations exist with the work of the authors of [20, 21], where both approaches are based on some kinds of labelled transition systems. The distinguishing feature of their approach is the transitions systems are labelled with logical formulae that provides an abstract characterization of the structure that a hole must possess and of the actions it can perform in order to allow a

transition to fire. Logical formulae are suitable formats that capture the general class of components that can act as the placeholders of the system during its evolution. In our approach we purposely leave the algebra of action terms undefined but the only operation we allow on action of holes is the comparison with other actions. Defining properly the interaction between a logical formulae in the action and the logics of the pNet composition seems very difficult.

Among the approaches for modelling open systems, one can cite [23] that uses transition conditions depending on an external environment, and introduce bisimulation relations based on this approach. The approach of [23] is highly based on logics and their bisimulation theory richer in this aspect, while our theory is highly structural and focuses on relation between structure and equivalence. Also, we see composition as a structural operation putting systems together, and do not focus on the modelisation of an unknown outside world. Overall we believe that the two approaches are complementary but checking the compatibility of the two different bisimulation theories is not trivial.

There is also a clear relation with the seminal works on rule formats for Structured Operational Semantics, e.g. DeSimone format, GSOS, and conditional rules with or without negative premisses [1, 24, 25, 8]. The Open pNets model provides a way to define operators similar to these rules formats, but with quite different aim and approach. A formal comparison would be interesting, though not trivial. What we can say easily is that: the pNet format syntactically encompasses ~~both~~ DeSimone, GSOS, and conditional premisses rules. Then our compositionality result is more powerful than their classical results, but this is not a surprise, as we rely on a (sufficient) syntactic hypothesis on a particular system, rather than the general rules defining an operator. Last, we intentionally do not accept negative premisses, that would be more to put into practice in our implementation. ~~an~~ This extension could be studied in future work.

*Symbolic and data-sensitive systems* We were substantially inspired by the work of Lin et al.

As mentioned in the *Introduction*, ~~the work that brought us a lot of inspirations are those of Lin et al~~. [3, 4, 5]. They developed the theory of symbolic transition graphs (STG), and the associated symbolic (early and late, strong and weak) bisimulations, they also study STGs with assignments as a model for message-passing processes. Our work extends these in several ways. First our models are compositional, and our bisimulations come with effective conditions for being preserved by pNet composition (i.e. congruent), even for the weak version. This result is more general than the bisimulation congruences for value-passing CCS in [3]. Then our settings for management of data types are much less restrictive, thanks to our use of satisfiability engines, while Lin's algorithms were limited to data-independent systems.

In a similar way, [26] presents a notion of "data-aware" bisimulations on data graphs, in which computation of such bisimulations is studied based on XPath logical language extended with tests for data equality.

Research related to the keyword "Symbolic Bisimulation" refer to two very different domains, namely BDD-like techniques for modelling and computing

It would be appropriate to mention other hierarchical models based on automata — statecharts? The model used by Andersen, Larsen and colleagues and for which they did compositional backward reachability?

finite-state bisimulations, that are not related to our topic; and symbolic semantics for data-dependant or high-order systems, that are very close in spirit to our approach. In this last area, we can mention Calder's work [27], that defines a symbolic semantic for full Lotos, with a symbolic bisimulation over it; Borgstrom et al., Liu et al, Delaune et al. and Buscemi et al. providing symbolic semantic and equivalence for different variants of pi calculus respectively [28, 29, 30, 31]; and more recently Feng et al. provide a symbolic bisimulation for quantum processes [32]. All the above works, did not give a complete approach for verification, and the models on which these works based are definitely different from ours.

Boreale & De Nicola?

Make this more concrete

## 7. Conclusion and Discussion

pNets (Parameterised Networks of Automata) is a formalism adapted to the representation of the behaviour of parallel or distributed systems. One strength of pNets is their parameterised nature, making them ~~adapted~~data the ~~representation~~ of systems of arbitrary size, and making the modelling of parameterised system possible. Parameters are also crucial to reason ~~on~~ interaction protocols that can address one entity inside an indexed set of processes. pNets have been successfully used to represent behavioural specification of parallel and distributed components and verify their correctness [10, 11]. VCE is the ~~specification~~ and verification platform that uses pNets as an intermediate representation.

suitable for

about

Open pNets are pNets ~~with holes~~; they are adapted to represent processes parameterised by the behaviour of other processes, like composition operators or interaction protocols that synchronise the actions of processes that can be plugged afterwards. Open pNets are hierarchical composition of automata with holes and parameters. We defined here a semantics for open pNets and a complete bisimulation theory for them. The semantics of open pNets relies on the definition of open automata that are automata with holes and parameters, but no hierarchy. Open automata are ~~somehow~~ labelled transition systems with parameters and holes, a notion that is useful to define semantics, but makes less sense when modelling a system, compared to pNets. To be precise, it is on open automata that we define our bisimulation relations.

specified further?

In the context of hierarchical system specifications these are "flattened" system descriptions, aren't they?

This article defines a strong and a weak bisimulation relation that are adapted to parameterised systems and hierarchical composition. Our bisimulation principle handles pNet parameters in the sense that two states might be or not in relation depending on the value of parameters. Our strong bisimulation is compositional ~~by nature~~ in the sense that bisimulation is maintained when composing processes. We also identified a simple and realistic condition on the semantics of non-observable actions that allows weak bisimulation to be also compositional. Overall we believe that this article paved the way for a solid theoretical foundation for compositional verification of parallel and distributed systems.

pNets support the refinement checking at the automata level through a simulation approach, with symbolic evaluation of the guards and transitions. The



Again, I'm not so sure that those conditions are novel. IMHO, it is good to see that they work in your setting, but I'd recommend a comparison with the literature on rule formats for weak bisimilarity.

definition of simulation on open automata should be stronger than a strict simulation since it matches a transition with a family of transitions. Such a relation should be able to check the refinement between two open automata with the same level of abstraction but specified differently, for example, by duplicating states, removing transitions, reinforcing guards, modifying variables. Additionally, composition of pNets gives the possibility to either add new holes to a system or fill holes. A useful simulation relation should thus support the comparison of automata that do not have the same number of holes. Designing such a simulation relation is a non-trivial extension of this work that we are investigating.

We are currently extending this work, looking at further properties of FH-bisimulation, but also the relations with existing equivalences on both closed and open systems. In particular, our model being significantly different from those considered in [3], it would be interesting to compare our "FH" family of bisimulations with the hierarchy of symbolic bisimulations from these authors. We also plan to apply open pNets to the study of complex composition operators in a symbolic way, for example in the area of parallel skeletons, or distributed algorithms. We have developed tool support for computing the symbolic semantics in term of open automata [15], and have developed algorithms to check strong FH-bisimulation [14]. More recently we published preliminary work for the case of weak FH-Bisimulation [1]. The challenges here, in the context of our symbolic systems, ~~is~~ not so much algorithmic complexity, as was the case with classical weak bisimulation on finite models, but decidability and termination. The naive approach using an explicit construction of the weak transition, may in itself introduce non-termination, so we prefer a direct implementation of the weak bisimulation definition, without constructing the weak automata, but searching *on demand* to construct the required weak transitions. Beside, we explore in [33] more pragmatic approaches using weak bisimulation preserving (pattern-based) reduction rules.

## References


[1] De Simone, R.: Higher-level synchronising devices in MEIJE-SCCS. Theoretical Computer Science **37** (1985) 245–267

[2] Larsen, K.G.: A context dependent equivalence between processes. Theoretical Computer Science **49** (1987) 184–215

[3] Ingólfsdóttir, A., Lin, H.: A symbolic approach to value-passing processes. In Bergstra, J.A., Ponse, A., Smolka, S.A., eds.: Handbook of Process Algebra. North-Holland/Elsevier (2001) 427–478

[4] Hennessy, M., Lin, H.: Symbolic bisimulations. Theoretical Computer Science **138**(2) (1995) 353–389

[5] Lin, H.: Symbolic transition graph with assignment. In Montanari, U., Sassone, V., eds.: Concur'96. Volume 1119 of LNCS., Springer, Heidelberg (1996) 50–65

[6] Hennessy, M., Rathke, J.: Bisimulations for a calculus of broadcasting systems. Theoretical Computer Science **200**(1-2) (1998) 225–260

[7] Groote, J.F.: Transition system specifications with negative premises. Theoretical Computer Science **118**(2) (1993) 263–299

[8] van Glabbeek, R.: The meaning of negative premises in transition system specifications ii. The Journal of Logic and Algebraic Programming **60-61** (2004) 229–258 Structural Operational Semantics.

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

[10] Ameur-Boulifa, R., Henrio, L., Kulankhina, O., Madelaine, E., Savu, A.: Behavioural semantics for asynchronous components. Journal of Logical and Algebraic Methods in Programming **89** (2017) 1 – 40

[11] Henrio, L., Kulankhina, O., Madelaine, E.: Integrated environment for verifying and running distributed components. In: in proc. of the 19th Int. Conf. on Fundamental Approaches to Software Engineering (FASE'16), Springer (2016)

[12] Henrio, L., Madelaine, E., Zhang, M.: A Theory for the Composition of Concurrent Processes. In Albert, E., Lanese, I., eds.: 36th International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE). Volume LNCS-9688 of Formal Techniques for Distributed Objects, Components, and Systems., Heraklion, Greece (June 2016) 175–194

[13] Arnold, A.: Synchronised behaviours of processes and rational relations. Acta Informatica **17** (1982) 21–29

[14] Hou, Z., Madelaine, E.: Symbolic Bisimulation for Open and Parameterized Systems. In: PEPM 2020 - ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, New-Orleans, United States (January 2020)

[15] Qin, X., Bliudze, S., Madelaine, E., Zhang, M.: Using SMT engine to generate symbolic automata. In: 18th International Workshop on Automated Verification of Critical Systems (AVOCS 2018), Electronic Communications of the EASST (2018)

[16] Boulifa, R.A., Halalai, R., Henrio, L., Madelaine, E.: Verifying safety of fault-tolerant distributed components. In: International Symposium on Formal Aspects of Component Software (FACS 2011). Lecture Notes in Computer Science, Oslo, Springer (September 2011)

[17] Johnson, K., Calinescu, R., Kikuchi, S.: An incremental verification framework for component-based software systems. In: Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering. CBSE '13, New York, NY, USA, ACM (2013) 33–42

[18] Johnson, K., Calinescu, R.: Efficient re-resolution of smt specifications for evolving software architectures. In: Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures. QoSA '14, New York, NY, USA, ACM (2014) 93–102

[19] Gaspar, N., Henrio, L., Madelaine, E.: Formally reasoning on a reconfigurable component-based system — a case study for the industrial world. In: The 10th International Symposium on Formal Aspects of Component Software, Nanchang, China (October 2013)

[20] Baldan, P., Bracciali, A., Bruni, R.: Bisimulation by unification. In Kirchner, H., Ringeissen, C., eds.: Algebraic Methodology and Software Technology, 9th International Conference, AMAST 2002, France. Volume 2422 of Lecture Notes in Computer Science., Springer (2002) 254–270

[21] Baldan, P., Bracciali, A., Bruni, R.: A semantic framework for open processes. Theor. Comput. Sci. **389**(3) (2007) 446–483

[22] Dubut, J.: Bisimilarity of diagrams. In Fahrenberg, U., Jipsen, P., Winter, M., eds.: Relational and Algebraic Methods in Computer Science - 18th International Conference, RAMiCS 2020, Palaiseau, France, April 8-11, 2020, Proceedings [postponed]. Volume 12062 of Lecture Notes in Computer Science., Springer (2020) 65–81

[23] Beohar, H., König, B., Küpper, S., Silva, A.: Conditional transition systems with upgrades. Science of Computer Programming **186** (2020) 102320

[24] Bloom, B., Istrail, S.: Bisimulation can't be traced: preliminary report. In: Department of Computer Science, Cornell University. (1988) 229–239

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

[26] Abriola, S., Barceló, P., Figueira, D., Figueira, S.: Bisimulations on data graphs. J. Artif. Intell. Res. **61** (2018) 171–213

[27] Calder, M., Shankland, C.: A symbolic semantics and bisimulation for full lotos. In: International Conference on Formal Techniques for Networked and Distributed Systems, Springer (2001) 185–200

[28] Borgström, J., Briais, S., Nestmann, U.: Symbolic bisimulation in the spi calculus. In: International Conference on Concurrency Theory, Springer (2004) 161–176

[29] Delaune, S., Kremer, S., Ryan, M.: Symbolic bisimulation for the applied pi calculus. In: International Conference on Foundations of Software Technology and Theoretical Computer Science, Springer (2007) 133–145

[30] Liu, J., Lin, H.: A complete symbolic bisimulation for full applied pi calculus. In: International Conference on Current Trends in Theory and Practice of Computer Science, Springer (2010) 552–563

[31] Buscemi, M.G., Montanari, U.: Open bisimulation for the concurrent constraint pi-calculus. In: European Symposium on Programming, Springer (2008) 254–268

[32] Feng, Y., Deng, Y., Ying, M.: Symbolic bisimulation for quantum processes. ACM Transactions on Computational Logic (TOCL) **15**(2) (2014) 14

[33] Wang, B., Madelaine, E., Zhang, M.: Symbolic Weak Equivalences: Extension, Algorithms, and Minimization - Extended version. Research Report RR-9389, Inria, Université Cote d'Azur, CNRS, I3S, Sophia Antipolis, France ; East China Normal University (Shanghai) (January 2021)