Refinements for Open Automata

Rabéa Ameur-Boulifa, Quentin Corradi, Ludovic Henrio, Eric Madelaine

Refinements for Open Automata

Rabéa Ameur-Boulifa[1][0000−0002−2471−8012], Quentin Corradi[2][0000−0003−4218−3987], Ludovic Henrio[2][0000−0001−7137−3523], and Eric Madelaine[3][0000−0002−5552−5993]

[1] LTCI, Télécom Paris, Institut Polytechnique de Paris, France
first.last@telecom-paris.fr
[2] Université Lyon, EnsL, UCBL, CNRS, Inria, LIP, France
first.last@ens-lyon.fr
[3] INRIA Sophia Antipolis Méditérannée, UCA, France
first.last@inria.fr

Abstract. Establishing equivalence and refinement relations between programs is an important mean for verifying their correctness. By establishing that the behaviours of a modified program simulate those of the source one, simulation relations formalise the desired relationship between a specification and an implementation, two equivalent implementations, or a program and its optimised implementation. This article discusses a notion of simulation between *open automata*, which are symbolic behavioural models for communicating systems. Open automata may have *holes* modelling elements of their context, and can be composed by instantiation of the holes. This allows for a compositional approach for verification of their behaviour.
We define a simulation between open automata that may or may not have the same holes, and show under which conditions these refinements are preserved by composition of open automata.

Keywords: Labelled transition systems · Simulation · Composition.

1 Introduction

Compositional design is a highly convenient approach for specifying and verifying large systems. Automata are often used as the basic formalism for this approach, but most automata definitions allow only the specifications of finite closed systems. These systems can be verified efficiently, but programming often consists in writing systems that should be interfaced with others, and with potentially unbound behaviours. We investigate in our works the reasoning on open symbolic systems, with a strong focus on compositionality of properties. More precisely, we say that a system is open if it contains a "hole" to be filled by another system. Open systems are typically composition operators [17] or componentised systems where some of the components are yet to be provided [6]. This form of composition is more complex to handle than top-level interaction usually found in process algebra, as the behaviour of each entity in the system is parameterised both by classical symbolic variables and by process variables.

Symbolic systems and their bisimulation raises additional challenges [15,17]. Reasoning on a symbolic automaton allows one to represent an infinite system in a finite manner, but then the state of the system is not only characterised by an automaton state but also by the value of the different variables representing the system. In parameterised systems, it is necessary to guard state transitions depending on the system state and on the input values. This is why in previous works and in this article, it we extend the classical form of bisimulation relation: in a symbolic setting a bisimulation relation relates classically states of two systems but it is additionally parameterised by a formula that must be verified by the state variables. This has been introduced in details in previous works [6] and will be recalled briefly in Section 2.2. We have shown in previous works that open symbolic systems are particularly convenient to model process algebra operators and open component systems with infinite behaviour [6,17].

The refinement concept plays an important role in software engineering. In addition to helping to cope with the complexity of requirements and design, refinement provides a foundation for ensuring system correctness. The correctness of a system can be established by proving, that a system refines its specification with the idea that some properties of the specification are preserved in the refined system. Refinement entails that one system can be considered as a more precise version of another one that is considered to be the specification. The refined model features all the specified behaviours with more concrete details. From a formal point of view, refinement is a mathematical relations between a specification and its implementation, with trace inclusion or simulation being frequently used relations [22,20].

In this article, we design a simulation theory for open symbolic systems. We build a very generic theory that should allow us to reason on simulation-based verification for most concurrent systems, as our base theory merely relies on automata parameterised by both variables and processes. As we shall see, our composition of automata is also very generic to account for any interaction mechanism found in concurrent systems. While our contribution is theoretical, it establishes the foundations for to the verification of any compositionally designed system, like component systems, algorithmic skeletons.

Open automata (that we abbreviate OA) were defined as a way to provide a semantics for open parameterised hierarchical labelled transition systems (abbreviated LTS). They were proposed as a theoretical foundation for parametrised automata used in verification tools and called *pNets*. An OA [17] is similar to a classical automaton but with variables and holes. Variables make automata symbolic and allow them to encode infinite-state systems. Holes enable the composition of automata: an automaton with a hole is an operator that takes another automaton as parameter and reacts to the actions it emits; the composed automaton is an automaton where the behaviour of one "process parameter" of the main automaton has been provided. Due to their generic nature, the notion of OA model is quite abstract but we already illustrated previously how to derive OAs for process algebra operators [17] or for component systems [6,5].

In previous works [6,24] a bisimulation relation was defined for OA and open parameterised hierarchical LTSs. It exhibited good properties concerning bisimulation, but refinement relations were not studied. In this article we go further to define a theory of simulation for OA. The simulation relation we introduce in the paper is based on the notion of simulation, in a similar way to that defined in classical automata theory [21,8]. It possesses the common behaviour-preserving property: all the behaviour of the abstract specification must be followed by its (complex) implementation but additional behaviours may exist. However we also ensure that a whole scenario, made of several steps, of the specification can also be simulated by the refined system, which is slightly richer than the traditional simulation relation and allows us to obtain a compositionality result.

Our contribution in this paper is the definition of a simulation relation for OA that has the following characteristics:

- Classical simulation characterisation but also an additional criteria ensuring that simulation does not introduce deadlocks when following a trace from the simulated automaton.
- Good properties relatively to composition: we prove that composition preserves the simulation relation.
- Ability to take into account both composition and transitivity: this is a challenge because composition changes the set of holes of the OA and simulation takes into account the actions of the holes.

The simulation relation is introduced in two steps. First we define a simulation that relates two automata with the same holes, which allows us to focus on the automaton aspect. Second we introduce a relation that relates two automata with different sets of holes, which allows us to take into account the open nature of OA, and to deal with composition. Properties of the simulation are stated and proven on the second, more general version of the relation, thus also being valid for the first simpler simulation relation.

This paper is organized as follows. Section 2 recalls the definition of OA and defines their composition. We then define a simulation relation for OA, first only considering two automata with the same set of holes in Section 3 and generalize it to automata with a different set of holes in Section 4. Section 5 is dedicated to formalize and prove basic properties of the simulation defined, including the proof that simulation is a preorder and has nice composability properties. In Section 6 we review related works, and Section 7 concludes the paper.

2 Open Automata and their Composition

This section presents our notations and the principles of automata. Except for minor changes in the notations, compared to previous works [6] the only new contribution is the definition of a composition operator for OA.

2.1 Preliminaries and notations

Countable families of values (equivalent to maps) will be noted $x_i^{i\in I}$, $\{i \mapsto x_i \mid i \in I\}$, or $\{i \leftarrow x_i \mid i \in I\}$, depending on what is more convenient (e.g. $i \leftarrow x_i$ is used

for maps that are used as substitution). Statements like $\exists c_j^{j\in J}$ defines both J and the mapping $j \mapsto c_j$. The disjoint union on sets is noted $\uplus$. Disjoint union is also used on maps. There are several ways of ensuring a union is disjoint, we will indifferently either suppose sets are disjoint or rename conflicting objects (useful for variables). In a formula, a quantifier followed by a finite set will be used as a shorthand for the quantification on every variable in the set: $\forall\{a_1,\ldots,a_n\}, \exists\{b_1,\ldots,b_m\}, P$ means $\forall a_1,\ldots,\forall a_n, \exists b_1,\ldots,\exists b_m, P$.

Our expression algebra E is the disjoint union of terms, actions, and formulas $E = \mathcal{T} \uplus \mathcal{A} \uplus \mathcal{F}$. $\mathcal{T}$ and $\mathcal{A}$ are term algebras. The set of formulas $\mathcal{F}$ contain at least first order formulas and equality[4] over $\mathcal{T}$ and $\mathcal{A}$. For $e \in E$, $vars(e)$ is the set of variables in e that are not bound by a binder. An expression is closed if $vars(e) = \varnothing$. The set $\mathcal{P}$ denotes values which is a subset of closed terms. $\mathcal{F}_V$ is the set of formulas f that only uses variables in V, i.e., the formulas such that $vars(f) \subseteq V$. The parallel substitution of variables in e by a map $\psi : V \to \mathcal{T}$ is denoted $e\{\!\{\psi\}\!\}$.

We suppose given a satisfiability relation on closed formulas, denoted $\models f$. We will use two variants of the satisfiability relation:

- The satisfiability of a formula $f \in \mathcal{F}$ under some valuation $\sigma : V \to \mathcal{P}$ is defined as follows: $\sigma \models f \iff \models \exists vars(f\{\!\{\sigma\}\!\}), f\{\!\{\sigma\}\!\}$
- The satisfiability of a formula $f \in \mathcal{F}$ with some variable set V as context is defined as follows: $V \models f \iff \models \forall V, \exists(vars(f) \setminus V), f$

2.2 Open Automata (OA)

OA are labelled transition systems with variables that can be used to compose other automata: they are made of transitions that are dependent on the actions of "holes", a composition operation consists in filling a hole with another automaton to obtain a more complex automaton. The variables makes the OA symbolic, and the holes allow for a partial definition of the behaviour.

Definition 1 (Open transition, Open automaton). *An* open automaton *is a tuple $\langle S, s_0, V, \sigma_0, J, T\rangle$ with S a set of states, $s_0 \in S$ the initial state, V the finite set of variable names, $\sigma_0 : V \to \mathcal{P}$ the initial valuation of variables, J the set of hole names and T the set of open transitions.*

An open transition *is a structure* $s \xrightarrow{\alpha} s'$ *with* $\beta_j^{j\in J'}, g, \psi$ *made of several composing entities, equivalent to a tuple. In an open transition $s, s' \in S$ are the source and target states, $\alpha \in \mathcal{A}$ is the resulting action that can be observed from the outside, $J' \subseteq J$ are the holes involved in the transition, $g \in \mathcal{F}$ is the guard that may constraint the transition, and $\psi : V \to \mathcal{T}$ are the variable assignments that have an effect on the state of the automaton. Each $\beta_j \in \mathcal{A}$ is an action of the holes j, To be well-formed, an open transition should use only variables of the automaton*

[4] Equality does not need to be only syntactic.

and variables appearing in the involved actions, formally:

$$vars(g) \subseteq vars(\alpha) \cup \bigcup_{j \in J'} vars(\beta_j) \cup V$$
$$\forall v \in V.\, vars(\psi(v)) \subseteq vars(\alpha) \cup \bigcup_{j \in J'} vars(\beta_j) \cup V$$

A pair consisting of a state and a valuation is called a *configuration* (of the automaton). We use two operators to access pieces of information of the OA.

Definition 2 (Out-transition, Transition variables). *Let $\langle S, s_0, V, \sigma_0, J, T\rangle$ be an automaton and let r be a state in S. $\mathrm{OT}_T(r) \subset T$ are the transition outgoing from state r[5]. The local variables of a transition $vars(t)$ are all variables appearing in transition t except the variables of the automaton. Outgoing transitions and variables are formally defined as follows.*

$$\mathrm{OT}_T(r) = \left\{ \begin{array}{c} \beta_j^{j\in J'}, g, \psi \\ s \xrightarrow{\alpha} s' \end{array} \in T \;\middle|\; s = r \right\}$$

$$vars\left(\begin{array}{c} \beta_j^{j\in J'}, g, \psi \\ s \xrightarrow{\alpha} s' \end{array} \right) = \left(vars(\alpha) \cup \bigcup_{j \in J'} vars(\beta_j) \right) \setminus V$$

Example 1 (`prod-cons`). As a running example, we consider a classical producer-consumer pair interacting through FIFO buffer, named `prod-cons`. Fig. 1 reflects the overall structure of the system involving a producer process, a consumer process and an orchestrator that coordinates their activities.

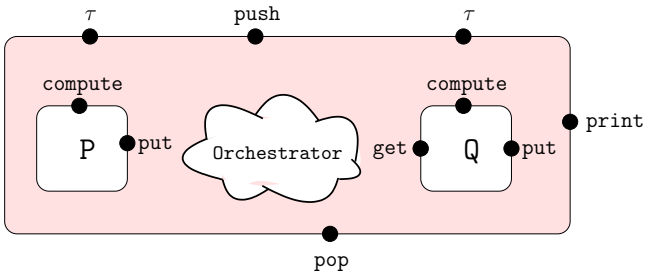


Fig. 1. Structure of the example. Each box corresponds to a process whose ports are the actions it can perform. The actions observable by the environment are `push`, which indicates the enqueuing of an element, `pop` which indicates the dequeuing, and `print` which indicates the production of results.

The OA modelling the behaviour of such a system using an unbounded circular/ring buffer is depicted in Fig. 2. The automaton has a single state with

[5] When the set T is clear from the context, it will be omitted and we will use $\mathrm{OT}(r)$

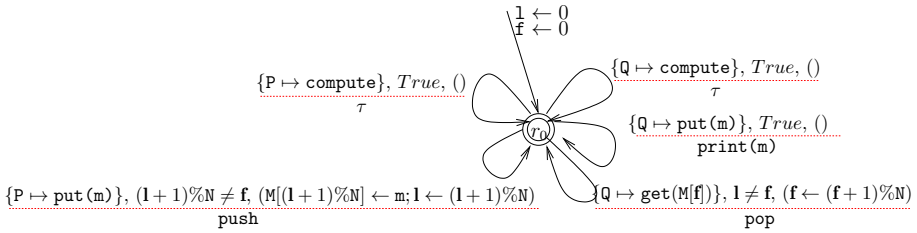


Fig. 2. OA for the `prod-cons` system using FIFO circular buffer.

two holes: `P` and `Q` that are the two interacting processes. `l` (as last) indicates the next available position for enqueuing an element and `f` (as first) is the position that contains the next element to be dequeued. The buffer reacts to a push from `P` and enqueues it. Similarly, whenever `Q` pops an element, it dequeues it. Additionally, whenever `Q` produces an item, it is exposed as an external observable `print` action. When any process do its internal computation, it is exposed externally as unobservable action τ.

The example uses several kinds of data. Variable `m` holds a message (we can leave the message type abstract here). We additionally use arrays of messages with a syntax of the form `M[l]` for array accesses; `M` is an array of `N` elements, from 0 to `N` − 1. Finally we use addition and modulo operation (%) on integers. □

Open automata composition. OA are partially specified automata, the partiality arises from the holes. A hole can be seen as a port in which we can plug an OA. The plugging operation is called composition. The composition of OA was already implicitly defined by the means of composition on pNets in previous work [17]. We provide here a (new) direct definition of composition for OA.

Definition 3 (Composition of OA). *Let $A_c = \langle S_c, s_{0c}, V_c, \sigma_{0c}, J_c, T_c\rangle$ be an OA and k one of its holes, $k \in J_c$. Let $A_p = \langle S_p, s_{0p}, V_p, \sigma_{0p}, J_p, T_p\rangle$ be another OA, the composition $A_c[A_p/k]$ that fills the hole k of the context OA A_c with the parameter OA A_p is defined as follows:*

$$A_c[A_p/k] ::= \langle S_c \times S_p, (s_{0c}, s_{0p}), V_c \uplus V_p, \sigma_{0c} \uplus \sigma_{0p}, J_p \uplus J_c \setminus \{k\}, T\rangle$$

with

$$T = \left\{ \frac{\beta_j^{j\in J'_p \uplus J'_c}, g_c \wedge g_p \wedge \alpha_p = \beta_k, \psi_c \uplus \psi_p}{(s_c, s_p) \xrightarrow{\alpha_c} (s'_c, s'_p)} \;\middle|\; \frac{\beta_j^{j\in J'_c \uplus \{k\}}, g_c, \psi_c}{s_c \xrightarrow{\alpha_c} s'_c} \in T_c, \frac{\beta_j^{j\in J'_p}, g_p, \psi_p}{s_p \xrightarrow{\alpha_p} s'_p} \in T_p \right\}$$
$$\cup \left\{ \frac{\beta_j^{j\in J'_c}, g_c, \psi_c}{(s_c, s_p) \xrightarrow{\alpha_c} (s'_c, s_p)} \;\middle|\; \frac{\beta_j^{j\in J'_c}, g_c, \psi_c}{s_c \xrightarrow{\alpha_c} s'_c} \in T_c, k \notin J'_c, s_p \in S_p \right\}$$

The first OA decides when the second can evolve by involving its hole in a transition: the action emitted when A_p makes a transition is synchronised with

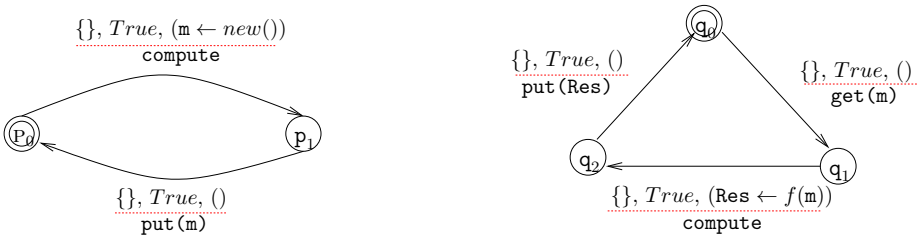


Fig. 3. (Left) A `producer`. It produces one item at a time and pushes it. (Right) A `consumer`. It pops an item, does some work and pushes the result.

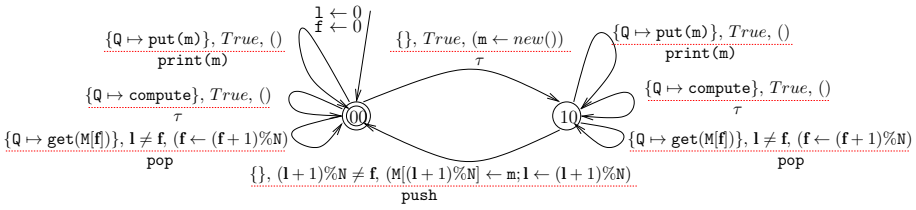


Fig. 4. OA for filling the hole `P` in `prod-cons`: `prod-cons`[`P`/`producer`].

the action of the hole k in transitions of A_c. The condition $\alpha_p = \beta_k$ ensures that the action emitted by the automaton A_p filling the hole is the one expected in the hole k of the open automaton A_c.

Example 2. Fig. 3 shows a producer automaton and a consumer automaton that can be used to fill the holes `P` and `Q` of `prod-cons` defined in Example 1.

The OA on Fig. 4 is the composition of the system in Fig. 2 and the producer in Fig. 3 (left). The composition consists of two states (the product of the states of both automata). The transitions from one state to another come from the synchronisation of the transitions of the encompassing automaton with those of the producer filling the hole `P`, this is why there is no more action from hole `P` in the composed automaton. Only elements related to the hole `P` are changed and in particular, transitions involving `Q` remain unchanged. □

2.3 Relations between Open Automata

Establishing semantic equivalences and simulation relations between different OA requires to compare their states. For this purpose, we suppose that the variables of the two OA are disjoint (a renaming of variables may have to be applied before comparing OA states).

Definition 4 (Relation on open automata configurations). *Suppose V_1 and V_2 are disjoint. A relation on configurations of two OA $\langle S_1, s_{01}, V_1, \sigma_{01}, J_1, T_1\rangle$ and $\langle S_2, s_{02}, V_2, \sigma_{02}, J_2, T_2\rangle$ is a function $\mathcal{R} : S_1 \times S_2 \to \mathcal{F}_{V_1 \uplus V_2}$.*

The idea is that two states are related depending on the satisfiability of the expression relying their variables, i.e., if the variables of the OA verify a certain formula. In other words, to each pair of states is attached a boolean formula that may refer to the variables of each of the two OA, stating whether the two states are related or not. Additionally, we say that the relation $\mathcal{R}$ *relates the initial states* of the automata if: $\sigma_{01} \uplus \sigma_{02} \models \mathcal{R}(s_{01}, s_{02})$. We illustrate such a relation over automata with bisimulation relation below.

2.4 A Bisimulation for Open Automata

Bisimulation between OA was defined in [6]. We show below the principles of this bisimulation. We first recall the usual definition of bisimulation. Bisimulation can be defined as follows for standard transition systems:

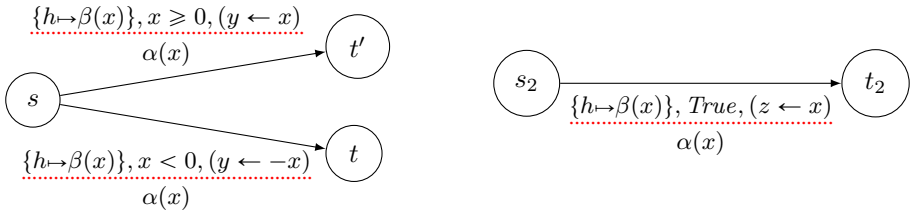
Definition 5 (Classical Bisimulation). *A bisimulation is a relation $\mathcal{R}$ such that if $s\ \mathcal{R}\ t$ then:*

$$\forall l\ s',\ s \xrightarrow{l} s' \implies \exists t'.\ s'\ \mathcal{R}\ t' \wedge t \xrightarrow{l} t'$$
$$\textit{and conversely}$$
$$\forall l\ t',\ t \xrightarrow{l} t' \implies \exists s'.\ s'\ \mathcal{R}\ t' \wedge s \xrightarrow{l} s'$$

$$\textit{i.e.}\quad \begin{array}{ccc} s & \mathcal{R} & t \\ l\downarrow & & \downarrow l \\ s' & \mathcal{R} & t' \end{array}$$

s and t are bisimilar, written $s \sim t$ iff there is a bisimulation relation $\mathcal{R}$ such that $s\ \mathcal{R}\ t$. If only the first one of the two implications above is verified, we say that s simulates t and denote it $s \leqslant t$.

A bisimulation relation relates pair of states and ensures that any behaviour of one automaton can be performed by the other one while staying in relation. We informally explain here the symbolic nature of the bisimulation for OA and the related complexity of its definition. The notion of symbolic bisimulation, as it was introduced in [15], is aimed at computing bisimulation of value-passing systems, i.e. systems made of processes exchanging data with their environment and between processes, where data are values from a possibly infinite domain. The presence of holes in fact raises no strong difficulty but the variables must be handled carefully. Consider the two following simple OA:



We should be able to consider these two OA as bisimilar. Both can input any $\beta(x)$ input on their hole and stores the value of x, emitting $\alpha(x)$ along the transition. The difference is the way x is stored. We can then define a configuration relation $\mathcal{R}$ such that $\mathcal{R}(s, s_2)$ is true and $\mathcal{R}(t, t_2)$ holds when $z \geqslant 0$ and

$y = z$, while $\mathcal{R}(t', t_2)$ holds when $z < 0$ and $y = -z$. This illustrates relation on configurations, but also shows that bisimulation on OA is more complex than in the classical case. Indeed, we need two transitions on the left OA to simulate a single one on the right OA. We should check that these two transitions cover all the cases accepted by the right hand side OA, and of course that destination states are in relation. Formally, FH-bisimulation is defined as follows [6]:

Definition 6 (Strong FH-bisimulation).
Suppose $\langle S_1, s_{01}, V_1, \sigma_{01}, J_1, T_1\rangle$ *and* $\langle S_2, s_{02}, V_2, \sigma_{02}, J_2, T_2\rangle$ *are OA with identical holes of the same sort, with disjoint sets of variables* ($V_1 \cap V_2 = \varnothing$).

Then $\mathcal{R}$*, a relation on configurations of OA, is an FH-bisimulation if and only if for any states* $s \in S_1$ *and* $t \in S_2$*, we have the following:*

- *For any open transition OT in* T_1*:* $\dfrac{\beta_j^{j\in J'}, g_{OT}, \psi_{OT}}{s \xrightarrow{\alpha} s'}$ *there exists an indexed set of open transitions* $OT_x^{x\in X} \subseteq T_2$*:* $\dfrac{\beta_{jx}^{j\in J_x}, g_{OT_x}, \psi_{OT_x}}{t \xrightarrow{\alpha_x} t_x}$ *such that the following holds*

$$\mathcal{R}(s,t) \wedge g_{OT} \implies \bigvee_{x\in X} \left(\forall j.\beta_j = \beta_{jx} \wedge g_{OT_x} \wedge \alpha = \alpha_x \wedge \mathcal{R}\left(s', t_x\right) \left\{\!\!\left\{\psi_{OT} \uplus \psi_{OT_x}\right\}\!\!\right\}\right)$$

- *and symmetrically any open transition from* t *in* T_2 *can be covered by a set of transitions from* s *in* T_1*.*

Two automata are bisimilar if there exists a strong FH-bisimulation $\mathcal{R}$ *that relates their initial states.*

Note that this definition matches an open transition t_1 to a family of covering open transitions $t_{2x}^{x\in X}$. Intuitively, this means that for every pair of related states (s_1, s_2) of the two automata, and for every transition of the first automaton from s_1, there is a set of matching transitions of the second automaton from s_2 such that the produced action match, the actions of the same holes and the successors are related after variable update. Technically, the following sections do not rely on the definition of strong bisimulation on OA, but they follow the same principles and in particular the same way to faithfully simulate an open transition by a set of other open transitions.

2.5 Reachability

We finally define a new predicate abstracting state reachability for OA, it allows us to reason on reachable states in an automaton. It can be seen as an abstraction of the reachable states under the form of a predicate that must stay verified along the execution of the OA.

Definition 7 (Reachability). *For any OA* $A = \langle S, s_0, V, \sigma_0, J, T\rangle$*, a reachability predicate* $\checkmark_A : S \to \mathcal{F}_V$ *is any predicate on states that is valid on initial state, and preserved across transitions:*

$$\sigma_0 \models \checkmark_A(s_0) \quad \wedge \quad \forall t = \overset{\beta_j^{j\in J'}, g, \psi}{\underset{s \xrightarrow{\alpha} s'}{\cdots\cdots\cdots}} \in T, vars(t) \models \big(\checkmark_A(s) \wedge g \implies \checkmark_A(s')\{\!\{\psi\}\!\}\big)$$

Reachability takes into account all paths, and can over-approximate the reachable configurations. From an automation point of view, finding the most precise reachability predicate for a given automaton is not decidable because of the symbolic nature of OA, but only an over-approximation is necessary.

3 Simulation for Automata with the Same Holes

Similarly to FH-bisimulation [6] we are interested in finding simulation relations between configurations of two OA that contain variables and holes. When dealing with open systems it is common to define simulation in terms of a simulation relation. We rely on a classical notion of simulation and perform the same extension as in [6], i.e., we start from a simulation relation and add holes and symbolic. The idea is to consider two configurations related by a relation; if one state can do a transition, then the other can also make this transition. Like for bisimulation, a simulation relation characterises when two states are related, and this characterisation is expressed as a predicate on the variables of the two automata. Simulation defines conditions on a relation $\mathcal{R}$ such that $\mathcal{R}(s_1, s_2)$ is a predicate (possibly involving variables of the automata) that is true when the state s_1 of A_1 simulates the state s_2 of A_2.

However here we want to build a simulation relation that also guarantees that no deadlock is introduced when refining the automaton. This property is quite frequent in simulation relation, and referred to as *lack of new deadlocks* [20] or *complete simulation* [23]. The notion of deadlock should however be specialised to our OA. Indeed, it is not very useful to check the existence of a transition, instead it makes more sense to use the guards to check if a transition can be taken. We thus define a deadlock reduction criterion based on how the outgoing transitions are guarded. As such, a simulation does not introduce deadlocks if in the conditions where no transition is possible in the refined automaton, no transition were already possible in the more general one. More formally, for any pair of states s_1 and s_2 we introduce a criterion of the form:

$$\forall (s_1, s_2) \in S_1 \times S_2,$$
$$V_1 \uplus V_2 \models \Big(\mathcal{R}(s_1, s_2) \wedge \neg\Big(\bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1)\Big) \implies \neg\Big(\bigvee_{t_2 \in \mathrm{OT}(s_2)} guard(t_2)\Big)\Big)$$

Which can be rewritten as:

$$\forall (s_1, s_2) \in S_1 \times S_2, V_1 \uplus V_2 \models \Big(\mathcal{R}(s_1, s_2) \implies \Big(\bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1)\Big) \vee \neg\Big(\bigvee_{t_2 \in \mathrm{OT}(s_2)} guard(t_2)\Big)\Big)$$

Both statements being equivalent, as each of them may reveal more intuitive than the other in different situations, we use them interchangeably. We can now state the definition of simulation between OA that have the same set of holes.

Definition 8 (Hole-equal simulation). *Consider two OA with identical set of holes:* $A_1 = \langle S_1, s_{01}, V_1, \sigma_{01}, J, T_1\rangle$ *and* $A_2 = \langle S_2, s_{02}, V_2, \sigma_{02}, J, T_2\rangle$*, the relation on configurations* $\mathcal{R} : S_1 \times S_2 \to \mathcal{F}_{V_1 \uplus V_2}$ *is a hole-equal simulation from* A_1 *to* A_2 *if the following conditions hold :*

(1) $\sigma_{01} \uplus \sigma_{02} \models \mathcal{R}(s_{01}, s_{02})$

(2) $\forall (s_1, s_2) \in S_1 \times S_2$,

$$\forall t_1 = \frac{\beta_{1j}^{j\in J_1'}, g_1, \psi_1}{s_1 \xrightarrow{\alpha_1} s_1'} \in \mathrm{OT}(s_1). \quad \exists \left(t_{2x} = \frac{\beta_{2xj}^{j\in J_{2x}'}, g_{2x}, \psi_{2x}}{s_2 \xrightarrow{\alpha_{2x}} s_{2x}'} \right)^{x\in X} \in \mathrm{OT}(s_2).$$

$$\big(\forall x \in X, J_{2x}' = J_1'\big) \wedge$$

$$V_1 \uplus V_2 \uplus vars(t_1) \models \mathcal{R}(s_1, s_2) \wedge g_1 \implies \bigvee_{x\in X} \begin{pmatrix} \alpha_{2x} = \alpha_1 \wedge \bigwedge_{j\in J_1'} \beta_{2xj} = \beta_{1j} \wedge \\ g_{2x} \wedge \mathcal{R}\big(s_1', s_{2x}'\big) \{\!\!\{ \psi_{2x} \uplus \psi_1 \}\!\!\} \end{pmatrix}$$

(3) Deadlock reduction:

$$\forall (s_1, s_2) \in S_1 \times S_2, V_1 \uplus V_2 \models \Big(\mathcal{R}(s_1, s_2) \implies \Big(\bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1) \Big) \vee \neg \Big(\bigvee_{t_2 \in \mathrm{OT}(s_2)} guard(t_2) \Big) \Big)$$

If there is a hole-equal simulation from A_1 *to* A_2*, then we say that* A_2 *simulates* A_1*; we denote it* $A_2 \leqslant A_1$*.*

The first and second conditions coincide with the natural way to prove inductively that an automaton simulates another by starting with the initial state. The third condition ensures that simulation prevents the introduction of deadlocks. Similarly to bisimulation, the second condition states that, for any transition of the simulating automaton A_1, it corresponds to a transition of the automaton A_2 that does the same thing and ends up in a similar state. However a family is needed in A_2 because of the symbolic nature of transitions, and because depending on the values of the variables, t_1 may correspond to different transitions in A_2. Our definition captures a simple simulation for OA with the same holes that is more expressive than a strict simulation since it matches a transition with a family of transitions. For example, with such a relation we are able to check the simulation between two OA that differ by duplicated states, removed duplicated transitions, reinforced guards, different variables, etc. We will show in Section 5 that this simulation relation has good properties in terms of transitivity, compositionality, and reflexivity.

Example 3. To illustrate the simulation of OA, we consider a variation on the `prod-cons` example. Namely, we suppose that the two processes `P` and `Q` communicate through a one-place buffer. Fig. 5 shows the OA modelling this simpler version of the system, that we refer to as `simprod-cons`. We can easily

check that this automaton simulates the one of Fig. 2. Indeed, one can see that $\mathcal{R} = \{(r_0, s_0) \mapsto \texttt{l} = \texttt{f}, (r_0, s_1) \mapsto \texttt{f} = \texttt{l} + 1\%\texttt{N}\}$ is a simulation relation. It follows that `simprod-cons` $\leqslant$ `prod-cons`. □

The simulation relation defined above is insufficient in the setting of composition which is the main advantage of the OA-based approach. Indeed, it should be possible to refine an automaton by filling its hole, providing a concrete view of a part of the application that was not specified originally. More generally, it should be possible to relate automata that do not have the same holes because composition is a crucial part of system specification. However, filling holes can result in a system with more or less holes than the original system because the plugged subsystem can contain itself many holes. Next section defines a more powerful simulation relation able to reason on automata with different sets of holes.

4 A Simulation Relation that Takes Holes into Account

This section extends the preceding relation to automata where the set of holes is not the same. This is particularly useful to state whether the automaton after composition is a simulation of the original automaton or not. Indeed, when composing the set of holes changes. Being able to compare automata with only some of their holes in common seems useful in general.

One major challenge in the extension of simulation to different sets of holes is to maintain a form of transitivity while being able to take into account the actions of some of the holes. A naive definition of simulation would ensure that only the holes that are identical in the two OA are taken into account in the simulation. Unfortunately, considering all the common holes does not ensure transitivity of the simulation for the following reason. If A_1 simulates A_2 and A_2 simulates A_3, and one hole j appears in A_3 and in A_1 but not in A_2 then we have no guarantee on the way A_1 and A_3 take the actions of this hole into account, thus a simulation between and A_1 and A_3 would require conditions involving actions of the hole j which cannot be ensured. The way we solve this issue is to remember in the simulation relation which holes have been compared. This makes the relation parameterised by a subset of the set of holes that belong to the two automata that we want to take into account. This way, in the example

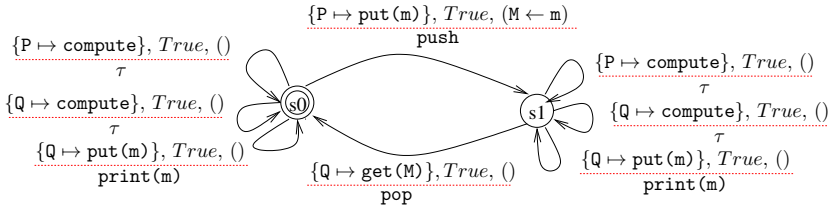


Fig. 5. The `simprod-cons` OA: the system using one-place buffer.

above, we would have no guarantee on actions the hole j by transitivity but can state a simulation relation with guarantees on the actions of the other holes.

In the following definition we add a parameter H which is the set of holes tracked by the simulation relation and adapt the definition by ignoring actions of the holes that are not in H.

Consequently, there is no guarantee related to the actions of the holes outside H. We provide compositionality properties when plugging an automaton inside a hole in H but cannot state anything when plugging an automaton outside H. The principle is that any property concerning holes that are not in H should be proven specifically for the considered automaton or the considered composition of automata.

Definition 9 (Hole-tracking simulation). *For two OA* $A_1 = \langle S_1, s_{01}, V_1, \sigma_{01}, J_1, T_1\rangle$ *and* $A_2 = \langle S_2, s_{02}, V_2, \sigma_{02}, J_2, T_2\rangle$*,* A_1 *is a simulation of* A_2 *tracking holes* H*, noted* $A_1 \leqslant_H A_2$*, with* $H \subseteq J_1 \cap J_2$*, if there is a relation on configurations* $\mathcal{R} : (S_1 \times S_2) \to \mathcal{F}_{V_1 \uplus V_2}$ *such that*[6]*:*

(1) $\sigma_{01} \uplus \sigma_{02} \models \mathcal{R}(s_{01}, s_{02})$

(2) $\forall (s_1, s_2) \in S_1 \times S_2$,

$$\forall \frac{\beta_{1j}^{j\in J_1'}, g_1, \psi_1}{s_1 \xrightarrow{\alpha_1} s_1'} \in \mathrm{OT}(s_1), \exists \left(\frac{\beta_{2xj}^{j\in J_{2x}'}, g_{2x}, \psi_{2x}}{s_2 \xrightarrow{\alpha_{2x}} s_{2x}'} \in \mathrm{OT}(s_2) \right)^{x\in X},$$
$$\big(\forall x \in X, J_{2x}' \cap H = J_1' \cap H\big) \wedge$$
$$V_1 \uplus V_2 \uplus \mathit{vars}(t_1) \models$$
$$\left(\mathcal{R}(s_1, s_2) \wedge g_1 \implies \bigvee_{x\in X} \left(\begin{array}{c} \alpha_1 = \alpha_{2x} \wedge \bigwedge\limits_{j\in J_1'\cap H} \beta_{1j} = \beta_{2xj} \wedge \\ g_{2x} \wedge \mathcal{R}\big(s_1', s_{2x}'\big)\{\!\{\psi_1 \uplus \psi_{2x}\}\!\} \end{array} \right) \right)$$

(3) Deadlock reduction:

$$\forall (s_1, s_2) \in S_1 \times S_2, V_1 \uplus V_2 \models \left(\mathcal{R}(s_1, s_2) \implies \Big(\bigvee_{t_1 \in \mathrm{OT}(s_1)} \mathit{guard}(t_1) \Big) \vee \neg \Big(\bigvee_{t_2 \in \mathrm{OT}(s_2)} \mathit{guard}(t_2) \Big) \right)$$

Note that every action of the holes outside H is unconstrained according to the simulation relation.

Property 1 (Relating simulations). Hole-equal simulation is a particular case of hole-tracking simulation when $J_1 = J_2 = H$.

In particular, if an OA has no hole, the two definitions are equivalent and result in a "symbolic simulation", if additionally there is no variable in the OA, this corresponds to classical simulation.

Example 4. Consider the automata of Examples 1 and 3. As we saw above, `simprod-cons` $\leqslant$ `prod-cons`, therefore `prod-cons` $\leqslant_{\{\mathtt{P},\mathtt{Q}\}}$ `simprod-cons`.

[6] Note that the definition below is identical to the hole equal simulation except $\cap H$ is added in a few places.

Property 2 (Tracked holes). By construction, if an automaton is the simulation of another one, it is also a simulation by tracking less holes.

$$A_1 \leqslant_H A_2 \wedge H' \subseteq H \implies A_1 \leqslant_{H'} A_2$$

Now that we have a simulation relation that takes both variable parameters and process parameters into account, we would like to ensure that it has properties one would expect for a simulation relation.

5 Properties of our Simulation Relations

Before reasoning on the properties of simulation, we need to introduce one additional notion that characterises when the composition of two automata does not introduce new blocked transitions.

5.1 Non-blocking Composition

Unfortunately, the deadlock reduction property in the definition of simulation is not compositional: the composition operator can itself introduce a deadlock. In other words, when filling the hole of two related automata with a third one, even if there is a deadlock reduction between the two original automata, there might not be a deadlock reduction in the composed ones. The same problem may arise when two related automata are composed in the same hole of a third one.

This creates a conflict between deadlock reduction and the properties involving composition. We call *non-blocking composition* a composition that can safely be used to compose OA that are involved in a deadlock reducing relation.

Definition 10 (Non-blocking composition). *Consider two OA:* $A_1 = \langle S_1, s_{01}, V_1, \sigma_{01}, J_1, T_1\rangle$ *and* $A_2 = \langle S_2, s_{02}, V_2, \sigma_{02}, J_2, T_2\rangle$. *Let* A *be the OA resulting from the composition* $A = A_1[A_2/k] = \langle S, s_0, V, \sigma_0, J, T\rangle$. *The composition* $A_1[A_2/k]$ *is non-blocking if* A *has a reachability predicate such that, for each reachable configuration, if there is a possible transition in* A_1 *then there is a possible transition in* A*:*

$$\forall s = (s_1, s_2) \in S, V \uplus \biguplus_{t \in \mathrm{OT}(s_1)} vars(t) \models \left(\checkmark_A(s) \wedge \bigvee_{t \in \mathrm{OT}(s_1)} guard(t) \implies \bigvee_{t \in \mathrm{OT}(s)} guard(t) \right)$$

Like in the definition of simulation (Definition 8) we use guards to ensure that the transition can occur. In general, one would not want to only consider non-blocking composition as it may reveal a bit restrictive, but it is the best necessary condition that we could identify for compositionality of simulation. It will be used to prove composition theorems given below. In absence of non-blocking composition, simulation may also be checked specifically for a given composed automaton.

5.2 Properties

We now state the properties of our simulation, their formal proofs can be found in the extended version of this paper [16]. We express these properties in terms of hole-tracking simulation because, thanks to Property 1 all the properties of hole-tracking simulation are also valid for hole-equal simulation. The first crucial theorem of simulation is that it is a preorder on the set of OA. This latter enables stepwise refinement.

Theorem 1 (Simulation is a preorder). *Hole-tracking simulation is reflexive and transitive: it is a preorder on the set of OA.*

Proof sketch. The relation $\leqslant_H$ is reflexive, $A \leqslant_H A$. This is shown by considering the relation $\mathcal{R}$ such that $\mathcal{R}(s_1, s_2) \triangleq s_1 = s_2 \wedge \bigwedge_{v \in vars(s_1)} v = v$ we can prove the conditions for Definition 9. In [16], we give proof of transitivity. It is done classically by identifying the relation between A_1 and A_3 that is a simulation. What is less classical is the definition of this relation because it is a boolean formula. For each couple of states s_1 and s_3 of A_1 and A_3 we build a formula that defines the simulation. To do this, we take the disjunction of formulas relating s_1 and s_3, and passing by all states s_2 of A_2. More precisely, we define a relation of the following form:

$$\mathcal{R}_{13}(s_1, s_3) = \bigvee_{s_2 \in S_2} (\mathcal{R}_{12}(s_1, s_2) \wedge \mathcal{R}_{23}(s_2, s_3))$$

We then prove that this relation is a simulation, according to Definition 9. □

The next theorem states that if two automata are in simulation relation and the same automaton is placed in the same hole of the two automata, then the simulation is preserved. This is the first step toward proving that simulation is compositional in the sense that it is sufficient to prove simulation for the composed automata separately to obtain a simulation relation.

Theorem 2 (Context refinement). *Let A_1, A_2 and A_3 be three OA with $A_1 \leqslant_H A_2$. Let J_3 be the set of holes of A_3 and suppose that $k \in H$. Suppose additionally that $A_1[A_3/k]$ is non-blocking. We have:*

$$A_1[A_3/k] \leqslant_{J_3 \uplus H \setminus \{k\}} A_2[A_3/k]$$

Proof sketch. The proof relies on a simulation relation that we consider is the one that makes A_1 and A_2 similar, complemented with identity of configurations for A_3. Then, by construction, all transitions of the composed automaton $A_1[A_3/k]$ are specified by open transitions of A_1. For the transitions that do not involve hole k, the transition of $A_1[A_3/k]$ is the same and simulation between A_1 and A_2 allows us to conclude directly. If the hole k is involved the considered relation implies that valuations in A_3 are equal (i.e., the value for each variable are the same in both valuations), after a transition we should obtain "equal" valuations because post-conditions are deterministic. The requirement "$A_1[A_3/k]$ is non-blocking" ensures the deadlock reduction property holds. More precisely, if $A_1[A_3/k]$ is stuck, then A_1 is stuck, and thus $A_1[A_2/k]$ is also stuck. □

Example 5. Consider again the `prod-cons` and `simprod-cons` automata given in the examples above. Since `prod-cons` $\leqslant_{\{\texttt{P},\texttt{Q}\}}$ `simprod-cons`, then according to Theorem 2, `prod-cons[producer/P]` $\leqslant_{\{\texttt{Q}\}}$ `simprod-cons[producer/P]`. The automaton of `prod-cons[producer/P]` is shown in Fig. 4. The automaton resulting from the composition of `simprod-cons` and `producer` is bigger and not shown here. □

Theorem 3 (Congruence). *Let A_1, A_2 and A_3 be three OA with $A_2 \leqslant_H A_3$. Let J_1 be the set of holes of A_1 and suppose that $k \in J_1$. Suppose additionally that the composition $A_1[A_2/k]$ is non-blocking. We have:*

$$A_1[A_2/k] \leqslant_{J_1 \uplus H \setminus \{k\}} A_1[A_3/k]$$

Consequently, as the simulation is transitive we can compose the previous theorems and state the following:

Theorem 4 (Composability). *Let A_1, A_2, A_3 and A_4 be four OA with $A_1 \leqslant_H A_2$ and $A_3 \leqslant_{H'} A_4$. Suppose that $k \in H$. We have:*

$$A_1[A_3/k] \leqslant_{H \uplus H' \setminus \{k\}} A_2[A_4/k]$$

Example 6. As an example of the use of this theorem, if we design a refined version of the producer process of Example 2 called `Refproducer`. According to Theorem 4, we have `prod-cons[producer/P]` $\leqslant_{\{\texttt{Q}\}}$ `simprod-cons[Refproducer/P]`.

Note that the substitution operation can be extended to a multiple substitution that fills several holes at the same time, and the theorems can be adapted accordingly.

6 Related Work

The origins of refinement are in the approach of programming that aims to provide solid foundations for building correct programs [12]. Many work contributed to the development of elaborated notions of refinement in various area (e.g. [7,1,10,8]). In the context of process algebra, refinement between processes can be defined in terms of simulations relation (e.g. ([19,22]). However, the concept of simulations presented so far has focused on the refinement of systems that are inherently closed, i.e., systems which are bounded and without environment,

The simulation ensures the preservation of safety properties as deadlock-freeness and, more generally, all linear temporal logic properties [1,20]. The difference between the existing refinement principles have been studied in [13], for example the authors explain in what sense failure semantics is different from (bi)simulation in the compared systems and properties ensured. In this paper we particularly focus on the compositionality of simulation-based refinement.

There are not a lot of works that study refinement for open systems. Defining refinement of open systems as trace inclusion is addressed as a notion of subtyping in type theory (e.g. [14,9]). The definition of refinement is based on a

connection between session types and communicating automata theories – a notion of session automata based on Communicating Finite-State Machines, that are used for modelling processes communicating through FIFO channels. The refinement of open systems is also defined in terms of alternating simulation [4,3]. Alternating simulation is originating from the game theory [2], it allows the study of relation between individual components by viewing them as alternating transition systems. In particular, a refinement of game-based automata expresses that the refined component can offer more services (input actions) and fewer service demands (output actions). However, the composition of such automata may lead to illegal states, where one automaton issues an output that is not acceptable as input in the other one. The theory of alternating simulation provides an optimistic approach to compute compatibility between automata based on the fact that each automaton expects the other to provide legal inputs, i.e., two components can be composed if there is an environment where they can work together. Our approach has some commonalities with the above mentioned simulation [3]: both are process-oriented approaches even if they are not based on the same notion of simulation, and both include in the model how to compose and interact with processes that are accepted as parameters. Nevertheless, they differ in that our approach focuses on the compositional properties of the simulation, and not on the fact that entities can be composed.

Previous works on OA focused on equivalence relations compatible with composition. In [18], a computable bisimulation is introduced, while in [6] a weak version of the bisimulation is introduced. In this paper we tackle the refinement relation in the form of simulation, as is the case for the corresponding relations on labelled transition systems [8]. Unlike the standard simulation we deal with symbolic and open models. In [25], the authors exploit transition systems to reason about the systems that are partially specified by using variables, making the state space potentially infinite.

Some work target component-based refinement with the concern of preserving deadlock freedom (e.g. [11,20]). These works are not concerned with the theory of open symbolic systems, and therefore do not focus on the same modularity as we do, in particular we provide preservation of refinement by composition.

7 Conclusion

In this article we investigated the notion of refinement for a symbolic and open model: open automata. OA are convenient for compositional software verification. Indeed, OA model parallel systems that are parameterised both by the use of variables and by the possibility to compose automata. The formalism supports compositional specification through the simulation paradigm. In this paper, we introduce a refinement relation between open automata. It relies on a simulation relation between the two automata; it specifies that the refined process must follow the behaviour of the simulated one. We finally showed that simulation is a preorder that is preserved by composition, both when filling a hole and when placing automata in comparable contexts.

References

1. Abrial, J.R.: The B-book - Assigning programs to meanings. Cambridge University Press (1996)
2. de Alfaro, L.: Game models for open systems. In: Dershowitz, N. (ed.) Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday. Lecture Notes in Computer Science, vol. 2772, pp. 269–289. Springer (2003). https://doi.org/10.1007/978-3-540-39910-0_12
3. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Tjoa, A.M., Gruhn, V. (eds.) Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering 2001, Vienna, Austria, September 10-14, 2001. pp. 109–120. ACM (2001). https://doi.org/10.1145/503209.503226
4. Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.Y.: Alternating refinement relations. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR'98 Concurrency Theory. pp. 163–178. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
5. 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**, 1–40 (2017). https://doi.org/https://doi.org/10.1016/j.jlamp.2017.02.003, `https://www.sciencedirect.com/science/article/pii/S2352220817300287`
6. Ameur-Boulifa, R., Henrio, L., Madelaine, E.: Compositional equivalences based on Open pNets. Journal of Logical and Algebraic Methods in Programming **131**, 100842 (2023). https://doi.org/https://doi.org/10.1016/j.jlamp.2022.100842, `https://www.sciencedirect.com/science/article/pii/S2352220822000955`
7. Back, R., Sere, K.: Stepwise refinement of parallel algorithms. Science of Computer Programming **13**(2), 133–180 (1990). https://doi.org/https://doi.org/10.1016/0167-6423(90)90069-P, `https://www.sciencedirect.com/science/article/pii/016764239090069P`
8. Bellegarde, F., Julliand, J., Kouchnarenko, O.: Ready-simulation is not ready to express a modular refinement relation. In: Maibaum, T. (ed.) Fundamental Approaches to Software Engineering. pp. 266–283. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
9. Bravetti, M., Zavattaro, G.: Asynchronous session subtyping as communicating automata refinement. Softw. Syst. Model. **20**(2), 311–333 (2021). https://doi.org/10.1007/s10270-020-00838-x
10. Butler, M.J., Grundy, J., Långbacka, T., Ruksenas, R., von Wright, J.: The refinement calculator: Proof support for program refinement. In: Groves, L., Reeves, S. (eds.) Proc. Conf. Formal Methods Pacific'97, Springer Series in Discrete Mathematics and Theoretical Computer Science (01/01/97). pp. 40–61 (1997), `https://eprints.soton.ac.uk/250550/`
11. Dihego, J., Sampaio, A., Oliveira, M.: A refinement checking based strategy for component-based systems evolution. Journal of Systems and Software **167**, 110598 (2020). https://doi.org/https://doi.org/10.1016/j.jss.2020.110598
12. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall (1976)
13. Eshuis, R., Fokkinga, M.M.: Comparing refinements for failure and bisimulation semantics. Fundam. Inf. **52**(4), 297–321 (Apr 2002)
14. Gay, S.J., Hole, M.: Subtyping for session types in the pi calculus. Acta Informatica **42**(2-3), 191–225 (2005). https://doi.org/10.1007/s00236-005-0177-z

15. Hennessy, M., Lin, H.: Symbolic bisimulations. Theoretical Computer Science **138**(2), 353–389 (1995). https://doi.org/https://doi.org/10.1016/0304-3975(94)00172-F, https://www.sciencedirect.com/science/article/pii/030439759400172F, meeting on the mathematical foundation of programing semantics
16. Henrio, L., Madelaine, E., Ameur-Boulifa, R., Corradi, Q.: Refinements for Open Automata (Extended Version). Tech. Rep. RR-9517, Inria - Research Centre Grenoble – Rhône-Alpes (Sep 2023), https://inria.hal.science/hal-04193421
17. Henrio, L., Madelaine, E., Zhang, M.: A theory for the composition of concurrent processes. In: Albert, E., Lanese, I. (eds.) Formal Techniques for Distributed Objects, Components, and Systems. pp. 175–194. Springer International Publishing, Cham (2016)
18. Hou, Z., Madelaine, E.: Symbolic bisimulation for open and parameterized systems. In: Proceedings of the 2020 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation. pp. 14–26. PEPM 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372884.3373161
19. Jifeng, H.: Process simulation and refinement. Form. Asp. Comput. **1**(1), 229–241 (mar 1989). https://doi.org/10.1007/BF01887207, https://doi.org/10.1007/BF01887207
20. Kouchnarenko, O., Lanoix, A.: How to verify and exploit a refinement of component-based systems. In: Virbitskaite, I., Voronkov, A. (eds.) Perspectives of Systems Informatics. pp. 297–309. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
21. Milner, R.: Communication and Concurrency. Prentice-Hall, Inc., USA (1989)
22. Milner, R.: A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92. Springer (1980). https://doi.org/10.1007/3-540-10235-3
23. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press (2012), https://hal.inria.fr/hal-00907026
24. 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) (Jan 2021), https://hal.inria.fr/hal-03126313
25. Zhang, L., Meng, Q., Lo, K.: Compositional abstraction refinement for component-based systems. Journal of Applied Mathematics **2014**, 1–12 (2014). https://doi.org/10.1155/2014/703098