# Refinements for open pNets

Rabéa$^{1[1111-2222-3333-4444]}$, Quentin$^{1,2[1111-2222-3333-4444]}$,
Ludo$^{2[0000-1111-2222-3333]}$, and Eric$^{3[2222--3333-4444-5555]}$

[1] blabla `lncs@fff`
`http://www.springer.com/gp/computer-science/lncs`
[2] blibli
`{abc,def}@fff`

**Abstract.** The abstract should briefly summarize the contents of the paper in 150–250 words.

**Keywords:** Labelled transition systems · Refinement · Composition.

## 1 Introduction

**[TODO:1.5 pages]**

## 2 Related Work

**[TODO:1 page]**

Previous work on open automata focused on equivalence relations compatible with composition. In an article by Hou, Zechen and Madelaine [5], a computable bisimulation is introduced and proved equivalent to the previous bisimulation already introduced. In a more recent work by Ameur-Boulifa, Henrio and Madelaine [1], a weak version of the bisimulation on open automata is introduced. These works differ from ours because the relation introduced in this report is a refinement relation in the form of a simulation and not a bisimulation. Also we do not have results as strong as computability neither a weak version able to tackle silent actions.

Some related work on other models than open automata introduce refinement relations. In a chapter by Bellegarde, Julliand and Kouchnarenko [2], a simulation relation on transition systems is introduced. This simulation encompass action refinement, is able to deal with silent actions and is compatible with parallel composition. Here the refinement relation does not consider action refinement as valid but it should be done in future work. Also they check how LTL properties are preserved or combined using ther refinement which we do not do. However their model is less expressive: the transition system model is less expressive than open automata and the parallel composition is less expressive than composition on open automata. In a later report by Kouchnarenko and Lanoix [6], the refinement relation they introduce is on LTS (labelled transition systems). Their relation additionally prevents deadlock and livelocks. The compositon is also extended to synchronised composition which is more expressive. In our work we also deal with deadlocks but not with livelocks since the latter arise only with silent actions. This work is closer than the previous one to what we do here, still open automata are more expressive than LTS and composition is more general than sychronised composition.

A refinement relation on models nearer to open automata is introduced in an article by Zhang, Meng and Lo [7]. In their article they work with transition systems with variable which makes the state space potentially infinite. This aspect is also present in open automata. They show how invariants, a notion near to our reachability predicates, are composed. By relation on these invariants they introduce several refinement relations. We could have done something similar for non-locking composition reachability predicates which are introduced in Section **??**.

On the deadlock prevention aspect, an article by Dihego, Sampaio and Oliveira [3] present a refinement relation on process algebra (translated to LTS). This refinement relation is a special case of inheritance and prevents the introduction of deadlocks. Their refinement and inheritance are quite the opposite of our refinement in terms of new behaviours. They have channels, interfaces, inputs and outputs, which in the open automata model can be compared to action labels, holes and action data for both inputs and outputs. They have a rich composition as open automata but the introduction of deadlock is already prevented by a well chosen set of composing operations. Also their composition is slightly different than the one on open automata because they can cause loops by linking two channels of the same process, where in open automata the composition makes an oriented tree. In their model there is an explicit deadlock and a succesful termination where in open automata there are no explicit termination. We define a deadlock as a configuration without possible transition and assume what is a deadlock when comparing the open automata. To define their refinement and inheritance relation they use trace and failure semantics, which are weaker than (bi)simulations [4] and could break with open automata compositon.

## 3 Background: Open Automata and their Composition

**[TODO:3.5 pages]**

This section presents the notations we used and the principles of automata. Except for minor changes in the notations, the only new contribution of this section is the definition of a composition operator for open automata.

Families of values, or equivalently maps will be noted $\{i \mapsto x_i \mid i \in I\}$, $\{i \leftarrow x_i \mid i \in I\}$ or $x_i^{i\in I}$. The disjoint union on set is noted $\uplus$[3]. Disjoint union is also used on maps. 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: $\forall\{a_1,\dots,a_n\},\exists\{b_1,\dots,b_m\},P$ means $\forall a_1,\dots,\forall a_n,\exists b_1,\dots,\exists b_m,P$.

[3] $\uplus$ notation either supposes that the sets are disjoint or rename conflicting objects depending on the context

**Definition 1 (Expression algebra, Action algebra, Formulas, Terms).** *An 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 formulas $\mathcal{F}$ contain at least first order formulas and equality*[4] *over $\mathcal{T}$ and $\mathcal{A}$.*

$vars(e)$ is the set of variables in $e \in E$ that are not bound by any binder. An expression is closed if $vars(e) = \emptyset$.

**[TODO:on a besoin de quoi dans la suite?]**

**Definition 2 (Values, Satisfiability, (Parallel) substitution).** *We assume that the following are given:*

- *The values $\mathcal{P}$, which are interpretations of closed terms.*
- *The satisfiability relation on closed formulas, $\vdash f$ where $f \in \mathcal{F}_\emptyset$.*
- *The substitution in $e \in E$ of $x \in vars(e)$ by $t \in \mathcal{T}$, $e[t/x]$.*
- *The parallel substitution in $e \in E$ of variables in $V$ by $\psi : V \to \mathcal{T}$, $e\{\{\psi\}\}$.*

For the parallel substitution, the set $V$ is not required to be a subset of $vars(e)$. In the case it isn't, the variables in $V \setminus vars(e)$ are not substituted. The substitutions might give a ill-formed expression; for instance let the terms be integers and pairs with (pointwise) addition, $(a+b)\{\{a \mapsto 7, b \mapsto (4,5)\}\}$ is a ill-formed term. This can be guarded with the check $e[t/x] \in E$ and $e\{\{\psi\}\} \in E$ and it will implicitly be the case, for instance when there is quantification on $t$ and $\psi$, to simplify notations.

The interpretation of terms is supposed to be decidable. The satisfiability of formulas might not be decidable nor complete nor consistent, but we will consider in the following that they are; this can be achieved by restricting the term algebra to a decidable subset. For instance a formula with quantifiers on variables might not be provable even if it is true for all values of these variables. In practice the formulas will be given to a SMT solver and we cannot always make sure they have all the previous properties but, at least any property found satisfiable by the SMT is ensured to be true. $\vdash$ can hence be interpreted as an indicator of what is given to the SMT; it separates the external logic and the logic on $\mathcal{F}$.

### 3.1 Open Automata

## 4 A First Refinement Relation

**[TODO:4 pages]**

## 5 A Refinement Relation that Takes Holes into Account

**[TODO:3.5 pages]**

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

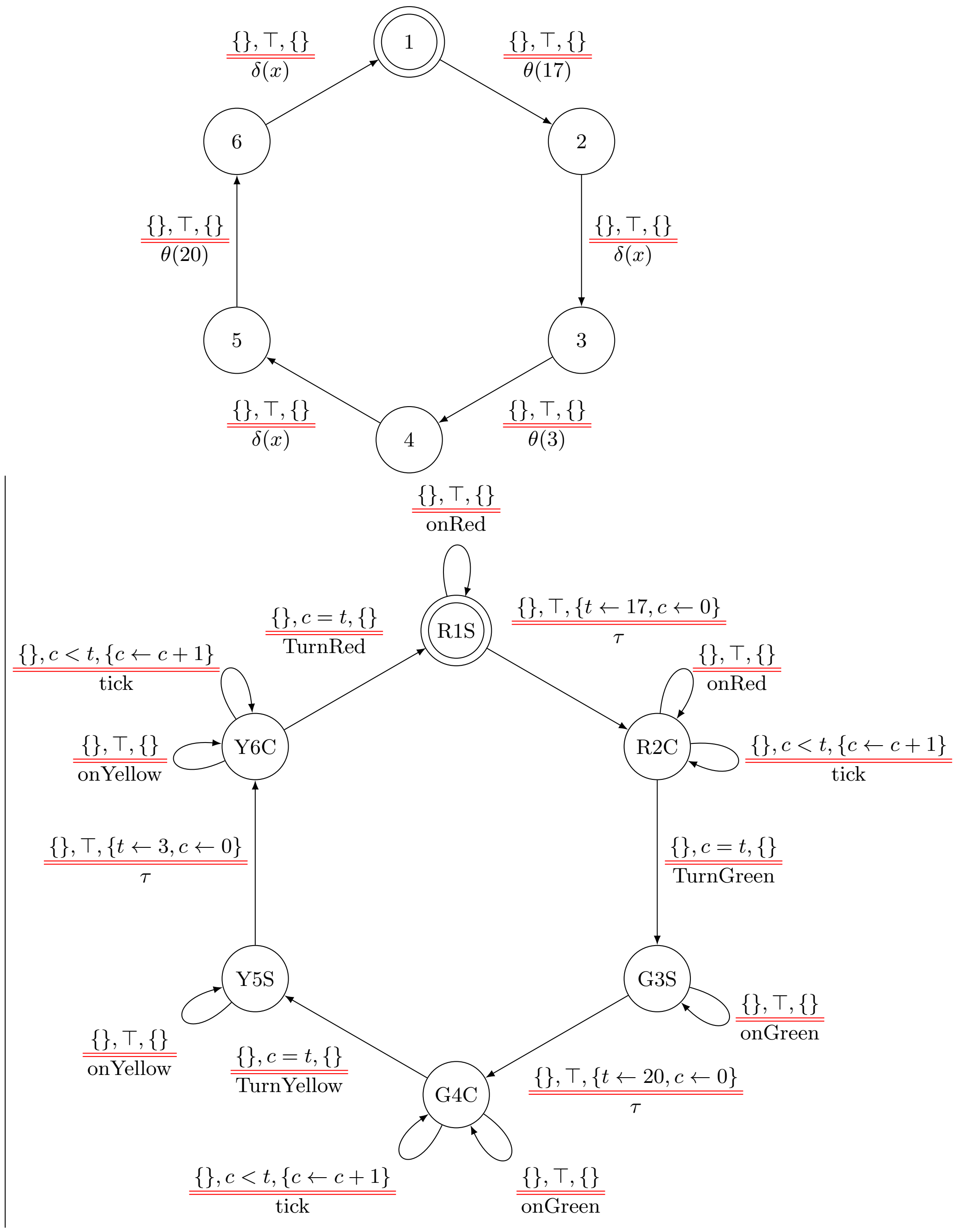
1
{}, ⊤, {} θ(17)
2
{}, ⊤, {} δ(x)
3
{}, ⊤, {} θ(3)
4
{}, ⊤, {} δ(x)
5
{}, ⊤, {} θ(20)
6
{}, ⊤, {} δ(x)
R1S
{}, ⊤, {} onRed
{}, ⊤, {t ← 17, c ← 0} τ
R2C
{}, ⊤, {} onRed
{}, c < t, {c ← c + 1} tick
{}, c = t, {} TurnGreen
G3S
{}, ⊤, {} onGreen
{}, ⊤, {t ← 20, c ← 0} τ
G4C
{}, ⊤, {} onGreen
{}, c < t, {c ← c + 1} tick
{}, c = t, {} TurnYellow
Y5S
{}, ⊤, {} onYellow
{}, ⊤, {t ← 3, c ← 0} τ
Y6C
{}, ⊤, {} onYellow
{}, c < t, {c ← c + 1} tick
{}, c = t, {} TurnRed

**Fig. 1.** TL

## 6 Properties

**[TODO:1.5 pages]**

## 7 Conclusion

**[TODO:0.5 pages]**

## References

1. Ameur-Boulifa, R., Henrio, L., Madelaine, E.: Compositional equivalences based on open pnets. CoRR **abs/2007.10770** (2020), `https://arxiv.org/abs/2007.10770`
2. 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)
3. 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, `https://www.sciencedirect.com/science/article/pii/S0164121220300765`
4. Eshuis, R., Fokkinga, M.M.: Comparing refinements for failure and bisimulation semantics. Fundam. Inf. **52**(4), 297–321 (Apr 2002)
5. 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, `https://doi.org/10.1145/3372884.3373161`
6. 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)
7. 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, `https://doi.org/10.1155/2014/703098`