The Omniscient Garbage Collector: a Resource Analysis Framework

Aurélien Deharbe, Frédéric Peschanski

The Omniscient Garbage Collector: a Resource Analysis Framework (technical report)

AURELIEN DEHARBE, Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6
FREDERIC PESCHANSKI, Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6

In this technical report, we develop a framework for the analysis of resource usage in concurrent systems. We propose an oracle – the omniscient garbage collector (OGC) – that decides precisely the minimal resource consumption of a process: its resource index. The underlying theory is developed as a nominal automata framework, and applied to the specific problem of tracking resource usage. The framework is put into practice to track resource usage and consumption in pi-calculus processes. Two complementary abstractions are proposed: the first is based on the labelled transitions, while the second relies on a variant calculus – slice-pi – that enrich reductions with observations about the resource events. The two abstractions are tested experimentally on classical pi-calculus examples using a prototype analysis tool. In all the examples the resource index can be computed very quickly – although in theory it is an NP-complete problem.

1. INTRODUCTION

The analysis of resource usage in computational systems is undoubtedly a fundamental research topic, especially in the realm of resource-contrained embedded systems. In this paper, we study basic qualitative and quantitative questions about resource usage for systems involving concurrent activities sharing dynamic resources.

If we abstract from its internal structure, a resource becomes a *pure name* [Gordon 2000], i.e. an object with a globally unique and testable identity. This is the specialty of *nominal calculi* in general, and the π-calculus [Sangiorgi and Walker 2001] in particular. Despite their lack of structure, the pure names display the primordial life-cycle of resources: (1) dynamic allocation, (2) arbitrary usage orderings, and (3) non-trivial *garbage collection* semantics, the latter point being central in our study.

Our starting point is the *resource graph* : a resource-focused view of the state-space of a process. From a qualitative perspective, our principal means of abstraction from the low-level details of the graphs (e.g. their branching structure) is the *resource profile* : a trace-semantics of resource usage. Traces of resource profiles are *words* of formal languages – that we name ν-languages – defined over infinite alphabets of *fresh* names. To recognize resource profiles, we introduce the ν-automata, which are variants of *register automata*. Using these devices, we show that the ν-languages corresponding to resource profiles are *quasi-reguar* [Kaminski and Francez 1994]. On the positive side, this means that many results about quasi-regular languages can naturally be lifted to resource profiles. On the more negative side, a basic question such as the equivalence-testing of resource profiles can easily be shown difficult.

To reason about the quantitative notion of *resource consumption*, we refine the ν-language characterization by considering finite restrictions of their alphabets. The corresponding *bounded resource profiles* provide a natural measure of resource consumption of process behaviors. An interesting indicator is the *resource bound* which confines the number of resources required for the correct execution of a given system. Ultimately, the least of such bounds – namely the *resource index* – represents a profound semantic characteristic of the behavior under study. We provide the *Omniscient Garbage Collector* (OGC) : a static analysis approach for resource bound and resource index closely related to the *maximal independent sets* problem and thus *graph coloring* [Jensen and Toft 2011]. We show, in particular, that computing the resource index is a NP-complete problem, but tight resource bounds can be computed with simple and efficient polynomial algorithms.

Beyond the theory, we aim at the development of practical tools for the analysis of resource usage in concurrent systems. Using a prototype, we propose a couple of experiments of resource consumption in the realm of the π-calculus. To illustrate the versatiliy of the approach, we propose two different resource abstractions for π-processes: one based on the labelled transitions for open systems, and another one for closed systems. Pure reductions are opaque and to circumvent this, we introduce *slice-π*, a rather standard π-calculus extended with an alternative restriction operator allowing the observation of names flowing between processes. In all the experiments we observe the same phenomenon – which reinforces a strong belief – that the apparent intractability of some of the proposed algorithms, especially the computation of the resource index, is largely compensated by the small size of the objects on which they apply. Indeed, most of the examples we explored (especially some classical π-calculus benchmarks) yield very small conflict graphs in comparison to the state space of the analyzed processes: in the order of at most a few dozen nodes for systems with more than 100 000 states.

The outline of the paper is as follows. In Section 2 we introduce the basic features and properties of resource graphs. The resource profiles are presented in Section 3. The related ν-automata theory is developed in Section 4. In Section 5 we discuss the quantitative aspects most notably the resource bounds and indices as well as the OGC framework for resource analysis. Our experimental study with the π-calculus is described in Section 6. A panorama of related work is given in Section 7.

This paper is based on a previous publication [Deharbe and Peschanski 2014] that focuses on the algorithmic aspects. Thus, the latter can be seen as a companion for the present paper whose purpose is to dig much deeper into the underlying automata-based theory. Most proofs (except for the shorter ones) are detailed in a dedicated Appendix (cf. page 21).

2. RESOURCE GRAPHS

As a starting point we propose a simple yet accurate characterization of resource usage in concurrent processes. The *resource graphs* correspond to the transition systems of processes in which we only observe the events related to resource usage.

Definition 2.1 (*Resource graph*). Let $\mathcal{R}$ be a countably infinite set of *resource variables* ranging over $X, Y, Z, \ldots$ A *resource graph* G is a directed graph $\langle R, V, E, \alpha, \gamma, \delta\rangle$ with:

— $R \subseteq \mathcal{R}$ a finite set of resource variables, also denoted by $\mathsf{vars}(G)$.
— V a finite set of vertices, and $E \subseteq V \times V$ a finite set of edges, such that there is a unique root $v_\perp \in V$ s.t. $\forall v \in V, (v, v_\perp) \notin E$ and a unique tail $v_\top \in V$ s.t. $\forall v \in V, (v_\top, v) \notin E$.
— $\alpha : V \to 2^R$ to record resource allocations,
— $\gamma : V \to 2^R$ to record resource uses,
— $\delta : V \to 2^R$ to record deletions.

An example of a resource graph is depicted on Fig. 1. It has 8 resource variables $A \ldots H$ and is sufficiently non-trivial so that it exhibits most of the "corner cases" of the model.

Most of the properties of resource graphs we will consider can be characterized as properties about finite paths, falling in two categories: *complete paths* and *lasso expansions*.

Definition 2.2 (*Complete path and lasso*). A *complete path* of a resource graph G with edge set E is of the form $\rho = \langle v_1, \ldots, v_n\rangle$ of pairwise distinct vertices such that $v_i \to v_{i+1} \in E$ for any $i,\ 1 \leq i \leq n-1$, and $v_1 = v_\perp,\ v_n = v_\top$. A *lasso* $\widehat{\rho} = \langle v_1, \ldots, v_{e-1} \mid v_e, \ldots, v_n\rangle$ is a sequence of pairwise distinct vertices such that $v_i \to v_{i+1} \in E$ for any $i,\ 1 \leq i \leq n-1$, $v_1 = v_\perp$ and $\exists e, 1 < e \leq n, v_n \to v_e \in E$. The vertex v_e is called the entry of the lasso and v_n its exit. These are respectively denoted by $v_e = \mathsf{entry}(\widehat{\rho})$ and $v_n = \mathsf{exit}(\widehat{\rho})$. The *finite expansion* of depth k of the lasso is denoted by $\widehat{\rho}^k = \langle v_1, \ldots, v_{e-1} \mid v_e, \ldots, v_n\rangle^k$, which corresponds to the finite path:

$$\langle v_1, \ldots, v_{e-1}, \underbrace{v_e, \ldots, v_n, \ldots, v_e, \ldots, v_n}_{k \text{ times}}, v_\top\rangle \text{ of length } e + k * (n - e + 1).$$

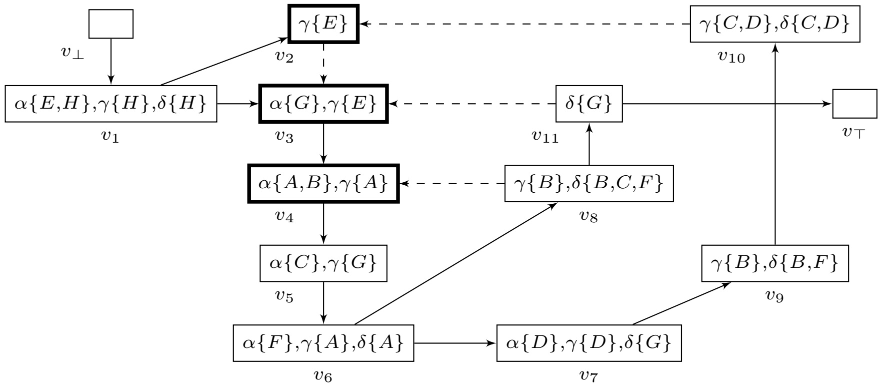

$v_\bot$
$\gamma\{E\}$
v_2
$\gamma\{C,D\},\delta\{C,D\}$
v_{10}
$\alpha\{E,H\},\gamma\{H\},\delta\{H\}$
v_1
$\alpha\{G\},\gamma\{E\}$
v_3
$\delta\{G\}$
v_{11}
$v_\top$
$\alpha\{A,B\},\gamma\{A\}$
v_4
$\gamma\{B\},\delta\{B,C,F\}$
v_8
$\alpha\{C\},\gamma\{G\}$
v_5
$\gamma\{B\},\delta\{B,F\}$
v_9
$\alpha\{F\},\gamma\{A\},\delta\{A\}$
v_6
$\alpha\{D\},\gamma\{D\},\delta\{G\}$
v_7

Fig. 1. Example of a resource graph

We denote by $\Psi(G)$ (resp. $\widehat{\Psi}(G)$) the finite sets of all complete paths (resp. lassos) of G.

There are only two complete paths in the graph of Fig. 1: $\langle v_\bot, v_1, v_k, \ldots, v_6, v_8, v_{11}, v_\top \rangle$ with $v_k = v_2$ or $v_k = v_3$. There are six lassos, an example being $\langle v_\bot, v_1, v_3 \mid v_4, \ldots, v_6, v_8 \rangle$ with entry v_4. Lasso entries are the emphasized vertices, while dashed edges represent the loop closing connections from the exit to the entry of the lasso. In [Deharbe and Peschanski 2014], we describe a polynomial algorithm to compute the complete paths and lassos of a resource graph using a decomposition in nested strongly-connected components.

We impose only minimal constraints on the nature and usage of resource events that are carried by resource graph vertices, although some usage patterns must be enforced.

Definition 2.3 (*Correct resource usage*). A resource graph G has correct resource usage iff for each resource $X \in \mathsf{vars}(G)$ it has at most one vertex v such that $X \in \alpha(v)$, and for each finite path $\rho = \langle v_1, \ldots, v_n \rangle$ of G there is at most one vertex v in ρ such that $X \in \delta(v)$. Moreover, $\forall j,\ 1 \leq j \leq n$ s.t. $X \in \gamma(v_j)$, $\exists i,\ 1 \leq i \leq j$ s.t. $X \in \alpha(v_i)$ and:

— if ρ is a complete path then: $\exists k,\ j \leq k \leq n$ s.t. $X \in \delta(v_k)$.
— if ρ is a lasso with entry v_e then:
  — (dynamic) $\exists k,\ j \leq k \leq n$ s.t. $X \in \delta(v_k)$ if $i \geq e$ or $\forall l,\ e \leq l \leq n,\ X \notin \gamma(v_l)$,
  — (static) $\forall k,\ 1 \leq k \leq n, X \notin \delta(v_k)$ otherwise.

Most of the constraints are obvious: a given resource is allocated only once globally, and deleted at most once in each path. For a resource used in a given path, a basic principle is that it must be preceded by an allocation and followed by a deletion. However, some subtlety arise because of the cyclic nature of the lassos. Suppose a resource X allocated at some vertex v_i and used at v_j ($j \geq i$) in a lasso with entry v_e. There are two cases to consider depending on whether X should "survive" the cycle or not. In the *dynamic* case X must be deleted at some vertex v_k with $k \geq j$. This corresponds to two possible situations: (1) the allocation is performed after the lasso entry (thus, within the cycle), or (2) the resource is not used within the cycle. Considering the lasso $\langle v_\bot, v_1, v_3 \mid v_4, \ldots, v_6, , v_8 \rangle$ in Fig. 1, then situation (1) applies to resources A and B and situation (2) applies to resource H which is allocated before the entry but not used within the cycle. Complementarily, if the allocation of X is performed before the entry of a lasso ρ *and* it is used within its cycle, then X must "survive" the cycle and is thus said a *static resource* for ρ. We denote by $\mathsf{static}(\rho)$ the set of variables that are static for the lasso ρ. For example, in the lasso $\langle v_\bot, v_1 \mid v_3, \ldots, v_{11} \rangle$ the resources E (allocated at v_1 and used at v_3) and G (allocated at v_3 and used at v_5) are static resources.

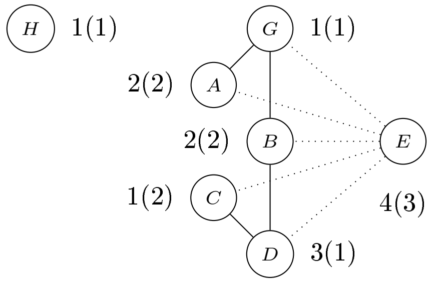


Fig. 2. The conflict relation $\sharp$ for the resource graph of Fig. 1.

This leads to a fundamental classification between *inactive* (i.e. unused), *static*, and otherwise *dynamic* resources.

Definition 2.4 (*Resource classification*). Let G be a resource graph with vertex set V.

$$\left[\begin{array}{l} \mathsf{inactive}(G) \overset{\text{def}}{=} \{X \mid \forall v \in V,\ X \notin \gamma(v)\} \\ \mathsf{static}(G) \overset{\text{def}}{=} \{X \mid \forall \rho \in \widehat{\Psi}(G),\ X \in \mathsf{static}(\rho)\} \\ \mathsf{dynamic}(G) \overset{\text{def}}{=} \mathsf{vars}(G) \setminus (\mathsf{static}(G) \cup \mathsf{inactive}(G)) \end{array}\right.$$

In our example the sets are $\mathsf{static}(G) = \{E, G\}$, $\mathsf{inactive}(G) = \{F\}$ and $\mathsf{dynamic}(G) = \{A, B, C, D, H\}$. A *central* concept is the notion of *conflict* in resource uses.

Definition 2.5 (*Conflict relation*). Let $X, Y \in \mathsf{vars}(G)$ be two distinct resources and ρ a path of G. A conflict between X and Y occurs in $\rho = \langle v_1, \ldots, v_n\rangle$ at position j, $1 \leq j \leq n$, denoted by $X \sharp_\rho^j Y$, if any of the following conditions holds:

— there exists two vertices v_i, v_k of ρ ($i \leq j \leq k$) such that $X \in \gamma(v_i) \cap \gamma(v_k)$ and $Y \in \gamma(v_j)$.
— whenever ρ is a lasso $\langle v_1, \ldots, v_{e-1} \mid v_e, \ldots, v_n\rangle$ in which X is static, $e \leq j \leq n$ and $Y \in \gamma(v_j)$.

If there is a position i in a path ρ such that $X \sharp_\rho^i Y$ or $Y \sharp_\rho^i X$ then X and Y are said in conflict, which is denoted $X \sharp Y$. □

The conflict graph obtained for our illustrative example is depicted on Fig. 2 (for the moment, we ignore the numeric annotations of the nodes). For example we have a conflict $B \sharp D$ generated by the sub-path $\langle v_7, v_9, v_{10}\rangle$ since $D \in \gamma(v_7)$, $B \in \gamma(v_9)$ and $D \in \gamma(v_{10})$. This corresponds to the first case of the definition. For the second case, a conflict such as $A \sharp G$ comes from the fact that $A \in \gamma(v_4)$ and $G \in \gamma(v_5)$ with v_5 occurring in a lasso after its entry v_4 and before its exit (in this case v_8).

Two events play an important role in the life-cycle of a resource: its *first use* and its *last use*. There is also one special case to consider: when a resource must be *fetched* for a later use, which as we will see allow a form of *non-local freshness*. These can be characterized on the finite paths of a resource graph.

Definition 2.6 (*First use, last use and resource fetch*). Let G be a resource graph, ρ either a complete path $\langle v_1, \ldots, v_n\rangle$ or a lasso $\langle v_1, \ldots, v_{e-1} \mid v_e, \ldots, v_n\rangle$, and v_j one of the vertices of ρ ($1 \leq j \leq n$). For a resource X of G:

— $X \in \mathsf{first}_\rho(v_j)$ iff $X \in \gamma(v_j)$ and $\forall i,\ 1 \leq i < j,\ X \notin \gamma(v_i)$ and if ρ is a lasso and $X \in \mathsf{static}(\rho)$ then $j < e$.
— $X \in \mathsf{last}_\rho(v_j)$ iff $X \in \gamma(v_j) \setminus \mathsf{static}(\rho)$ and $\forall k,\ j < k \leq n,\ X \notin \gamma(v_k)$

— $X \in \mathsf{fetch}_\rho(v_j)$ iff ρ is a lasso, $X \in \mathsf{static}(\rho)$, $j = e - 1$ and $\forall i,\ 1 \leq i < j,\ X \notin \gamma(v_i)$

For the example of Fig. 1, we consider the resource E in the complete path $\rho = \langle v_\perp, v_1, v_2, \ldots, v_6, v_8, v_{11}, v_\top\rangle$. Its first use is at vertex v_2 (i.e. $E \in \mathsf{first}_\rho(v_2)$) while its last use is at vertex v_3 (i.e. $E \in \mathsf{last}_\rho(v_2)$). Note that these are path-specific notions since in the other complete path (not going through v_2) the resource E is both first-and-last used at vertex v_3. Now we consider the lasso $\widehat{\rho} = \langle v_\perp, v_1, v_3 \mid v_4, v_5, v_6, v_8\rangle$ and resource G. Since G is static in $\widehat{\rho}$ it has no last use. Moreover, it has no first use also because what would be its first use is within the cycle. Hence at vertex v_4 (the entry of the lasso) the variable G must be fetched (i.e. pre-allocated), thus $G \in \mathsf{fetch}_{\widehat{\rho}}(v_4)$.

3. RESOURCE PROFILES

Resource graphs provide quite a low level view over concurrent process behaviors wrt. resource usage. All the properties we aim to study can be characterized at a more abstract level considering only *traces* of resource uses. The most fundamental aspect of these trace sets is that they involve dynamically bound resources, whose identity is not known in advance. This naturally leads to languages involving *pure names*, that is, sets of words with symbols freshly generated that we name *ν-languages.*

Definition 3.1 (*ν-language*). Let $\mathcal{V}$ be a (potentially) countably infinite set of pure names ranging over $\nu_1, \nu_2, \ldots$. A ν-language $\mathbb{L}$ over alphabet $\mathcal{V}$ is a set of words in $(2^{\mathcal{V}})^*$.

The definition makes ν-language quite similar to traditional formal language, although defined over potentially infinite alphabets. This way, when a new resource is needed for a given system to perform an action, a fresh identity for the resource is always available. Reasoning on languages with infinite alphabets is in general very difficult [Isper 1989] so we must define a proper language subset that fits our reasoning requirements.

Our objective is to precisely characterize the ν-language corresponding to the expected behavior of a resource graph. The first step is to explain how pure names are consumed by the processes.

Definition 3.2 (*Allocator, Allocation*). An *allocator* Γ is a partial one-to-one function from a finite set of variables to the alphabet of a ν-language. Let ρ be a finite path of a graph G and v one of its vertices an Γ an allocator. An *allocation* $\mathsf{alloc}_\rho(\Gamma, \alpha, v)$ of a ν-symbol α (a set of pure names) is such that α can be decomposed as $\mathsf{old}(\alpha) \cup \mathsf{trans}(\alpha) \cup \mathsf{new}(\alpha)$ with:

$$\left[\begin{array}{l}\mathsf{old}(\alpha) = \{\nu \in \mathsf{ran}(\Gamma) \mid \exists X \in \gamma(v) \setminus \mathsf{first}_\rho(v),\ \Gamma(X) = \nu\} \\ \mathsf{trans}(\alpha) \text{ is a set } T \subseteq \alpha \setminus \mathsf{old}(\alpha) \text{ such that } \mathsf{card}(T) = \mathsf{card}(\mathsf{first}_\rho(v) \cap \mathsf{last}_\rho(v)) \\ \mathsf{new}(\alpha) \text{ is a set } N \subseteq \alpha \setminus (\mathsf{old}(\alpha) \cup \mathsf{trans}(\alpha)) \text{ such that } \mathsf{card}(N) = \mathsf{card}(\mathsf{first}_\rho(v) \setminus \mathsf{last}_\rho(v))\end{array}\right.$$

The definition above explains how a ν-symbol α is consumed, which corresponds to resource uses at vertex v. The subset $\mathsf{old}(\alpha)$ corresponds to the pure names already bound in Γ (old bindings). The pure names in $\mathsf{new}(\alpha)$ corresponds to simultaneous allocations and uses at vertex v (new bindings). The remaining consumed names in $\mathsf{trans}(\alpha)$ are for a special case: when the names are allocated, used and released in an atomic way. In this case there is no need to record any binding.

The second ingredient is the binding of variables, i.e. the storage of the allocated pure names.

Definition 3.3 (*Binding*). Let Γ be an allocator, α a set of symbols and v a vertex of a finite path ρ. The *binding* of α to Γ is an allocator $\mathsf{bind}_\rho(\Gamma, \alpha, v) = \Gamma'_{\mathsf{old}} \cup \Gamma'_{\mathsf{new}} \cup \Gamma'_{\mathsf{fetch}}$ with:

$$\left[\begin{array}{l}\Gamma'_{\mathsf{old}} = \{X \mapsto \Gamma(X) \mid X \notin \mathsf{last}_\rho(v)\} \\ \Gamma'_{\mathsf{new}} = \{X \mapsto \nu \mid \nu \in \mathsf{new}(\alpha) \wedge X \in \mathsf{first}_\rho(v) \setminus \mathsf{last}_\rho(v)\} \\ \Gamma'_{\mathsf{fetch}} = \{X \mapsto \nu \mid \nu \notin \alpha \wedge X \in \mathsf{fetch}_\rho(v)\}\end{array}\right.$$

The definition separates the binding in three disjoint subsets: the old bindings that must be preserved from the previous steps (i.e. for allocated resources that are not release at vertex v and the new bindings that must be recorded for further uses. A special case if for the pure names that are allocated but not used at vertex v, which we call fetching a resource. Based on the previous two definitions we can properly define the trace sets of resource usage.

Definition 3.4 (*Resource profile*). The resource profile $\mathfrak{R}_G$ of a resource graph G is a ν-language such that $w = \alpha_1 \cdots \alpha_n \in \mathfrak{R}_G$ if and only if there exists a finite path $\rho = \langle v_0, \ldots, v_n \rangle$ (either a complete path or a finite expansion of a lasso) such that, for each position k, $0 < k \leq n$, we have $\mathsf{alloc}_\rho(\Gamma_{k-1}, \alpha_k, v_k)$ with: $\left[\begin{array}{l} \Gamma_0 = \emptyset \\ \forall j > 0,\ \Gamma_j = \mathsf{bind}_\rho(\Gamma_{j-1}, \alpha_j, v_j) \end{array} \right.$

Two properties summarize in *essence* the ν-languages characterized by the definition above: the *binding condition* that impersonates the *identity* of resources, and the *conflict freedom* that is the main requirement of resource profiles.

PROPOSITION 3.5 (BINDING CONDITION). *For a position k, $1 \leq k \leq n$ of a finite path $\rho = \langle v_1, \ldots, v_n \rangle$ (either a complete path or a finite expansion of a lasso) then $\Gamma_k(X) = \nu$ if and only if either there exists i, $1 \leq i \leq k$ such that $X \in \mathsf{first}_\rho(v_i) \cup \mathsf{fetch}_\rho(v_i)$ and $\Gamma_i(X) = \nu$, provided $\nexists j$, $i \leq j < k$ such that $X \in \mathsf{last}_\rho(v_j)$.*

PROPOSITION 3.6 (CONFLICT FREEDOM). *If $X \sharp_\rho^k Y$ then $\Gamma_k(X) \neq \Gamma_k(Y)$.*

PROOF. cf. Appendix A.1 page 21. □

4. AUTOMATA-THEORETIC FRAMEWORK

It is possible to work directly at the level of resource graphs to characterize many properties of the resource profiles. This is basically what it is done in [Deharbe and Peschanski 2014]. However, if this is interesting from an algorithmic point of view, we overlook many important theoretical issues. Moreover, the automata-theory developed in this section provides a much more solid ground for the statements of the properties and their proof.

4.1. ν-automata

Our first step is the definition of *recognizers* for an interesting subset of ν-languages.

Definition 4.1 (*ν-automaton*). A ν-automaton is a pair $\langle \mathcal{X}, A \rangle$, where $\mathcal{X}$ is a finite disjoint set of variables, and $A = \langle Q, q_{\mathsf{init}}, \Delta, F \rangle$ is a finite state automaton over the alphabet $\Sigma = 2^{\{\nu X, X, \overline{\nu} X \mid X \in \mathcal{X}\}}$ with Q a finite set of states, $q_{\mathsf{init}} \in Q$ the initial state, $\Delta \subseteq Q \times \Sigma \times Q$ the transition relation and $F \subseteq Q$ the set of accepting states.

A ν-automaton is similar to a finite-memory automaton (FMA [Kaminski and Francez 1994]) except that the transitions allow to explicitly *bind* or *unbind* a symbol. For example it is possible to fetch a fresh symbol without actually consuming it. Also, it is possible to release a name stored in memory. Moreover, the symbols are in fact sets of pure names, which amounts to consider simultaneous resource uses. Hence, ν-automata are resource-oriented variants of traditional FMA's.

As recognizers, the ν-automata correspond to a generalization of the resource profiles characterized by Def. 3.4.

Definition 4.2 (*ν-language of a ν-automaton*).
Let a ν-automaton be $\mathcal{A} = \langle \mathcal{X}, A \rangle$, with $A = \langle Q, q_{\mathsf{init}}, \Delta, F \rangle$. For $q \in Q$ an actual state of $\mathcal{A}$ is a *configuration* $q^c = (q, \Gamma)$ with Γ an allocator. We denote by Q^c the set of all configurations of automaton $\mathcal{A}$. The initial configuration is $q^c_{\mathsf{init}} \overset{\text{def}}{=} (q_{\mathsf{init}}, \emptyset)$ and $F^c = \{(q, \Gamma) \mid q \in F \wedge \Gamma \text{ is an allocator } \}$ is the set of accepting configurations. The transition relation Δ

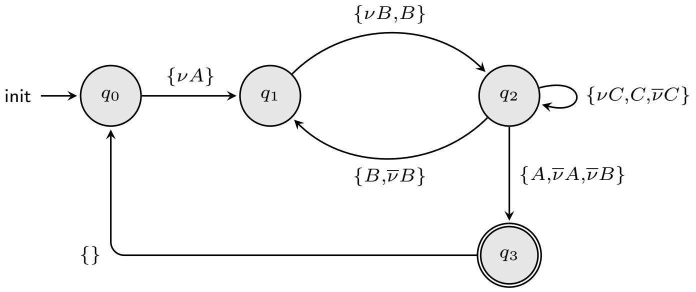

init
q_0
$\{\nu A\}$
q_1
$\{\nu B, B\}$
$\{B, \overline{\nu} B\}$
q_2
$\{\nu C, C, \overline{\nu} C\}$
$\{A, \overline{\nu} A, \overline{\nu} B\}$
q_3
$\{\}$

Fig. 3. An example of aν-automaton $\mathcal{A}$.

induces the relation $\Delta^c \subseteq Q^c \times \Sigma \times Q^c$ such that $((q, \Gamma), \alpha, (q', \Gamma')) \in \Delta^c$ if and only if there exists a label $\lambda \in 2^{\{\nu X, X, \overline{\nu} X \mid X \in \mathcal{X}\}}$ and an allocator Γ' such that $(q, \lambda, q') \in \Delta$ and:

— α can be decomposed as $\alpha_{\mathsf{old}} \cup \alpha_{\mathsf{trans}} \cup \alpha_{\mathsf{new}}$ with:

$$\left[\begin{array}{l}\alpha_{\mathsf{old}} = \{\nu \in \mathsf{ran}(\Gamma) \mid X \in \lambda \wedge \nu X \notin \lambda \wedge \Gamma(X) = \nu\} \\ \alpha_{\mathsf{trans}} = \{\nu \in \mathcal{V} \setminus \alpha_{\mathsf{old}} \mid X \in \lambda \wedge \nu X \in \lambda \wedge \overline{\nu} X \in \lambda\} \\ \alpha_{\mathsf{new}} = \{\nu \in \mathcal{V} \setminus (\alpha_{\mathsf{old}} \cup \alpha_{\mathsf{trans}}) \mid X \in \lambda \wedge \nu X \in \lambda \wedge \overline{\nu} X \notin \lambda\}\end{array}\right.$$

— $\Gamma' = \{X \mapsto \Gamma(X) \mid \overline{\nu} X \notin \lambda\}$
$\quad \cup \{X \mapsto \nu \mid \nu \in \alpha_{\mathsf{new}} \wedge \nu X \in \lambda \wedge \overline{\nu} X \notin \lambda\}$
$\quad \cup \{X \mapsto \nu \mid \nu \notin \alpha \wedge \nu X \in \lambda \wedge X \notin \lambda \wedge \overline{\nu} X \notin \lambda\}$

Let $w = \alpha_1 \alpha_2 \cdots \alpha_n$ be a ν-word over alphabet $2^{\mathcal{V}}$. A *run* of $\mathcal{A}$ on w consists of a sequence of configurations $c_0, c_1, \ldots, c_n$ such that $c_0 = q^c_{\mathsf{init}}$ and for all $i, 1 \leq i \leq n$, $(c_{i-1}, \alpha_i, c_i) \in \Delta^c$. For a word w, an *accepting run* of w is a run $c_0, c_1, \ldots, c_n$ such that $c_n \in F^c$. An automaton $\mathcal{A}$ *accepts* a word w if there exists an accepting run of w. The ν-language of automaton $\mathcal{A}$ is the of set of ν-words $\mathbb{L}(\mathcal{A}) = \{w \mid \mathcal{A} \text{ accepts } w\}$.

Example. Consider a ν-automaton $\mathcal{A} \overset{\text{def}}{=} \langle\{A, B, C\}, \langle\{q_0, q_1, q_2, q_3\}, q_0, \Delta, \{q_3\}\rangle\rangle$ with $\Delta = \{(q_0, \{\nu A\}, q_1), (q_1, \{\nu B, B\}, q_2), (q_2, \{B, \overline{\nu} B\}, q_1), (q_2, \{\nu C, C, \overline{\nu} C\}, q_2), (q_2, \{A, \overline{\nu} A, \overline{\nu} B\}, q_3), (q_3, \{\}, q_0)\}$. Its graphical representation is depicted on Fig. 3, and its behavior is as follows. From the initial state q_0 to q_1, an arbitrary (but finite) set of pure names is *fetched* for variable A, but only an empty-set of pure names is consumed along the transition. The states q_1 and q_2 form a kind a sub-automaton comparable to a FMA recognizing the repetition of a symbol. From q_1 to q_2 a symbol is is first read from the input and bound to variable B. Then an arbitrary number of symbols are read but not stored (this is a transient binding) in the self-loop of q_2. Then B is release either by reentering the loop starting from q_1 or before acceptance by final state q_3. The transition from q_2 to q_3 is interesting since variable the variable A fetched initially is now used. This means that the pure names consumed must be distinct from all the names previously used. Thus, the ν-automaton implements a form of *non-local freshness* unavailable in FMA. This is neither a *fresh register automaton* (FRA [Tzevelekos 2011]) which require *global freshness* while here, the pure names bound to A can be reused if after the transition from q_3 to q_0.

4.2. Resource profile recognizers

The investigation of the ν-automata in isolation – although an interesting goal – would go beyond the scope of the present paper. We are more interested in using ν-automata as recognizers of resource profiles.

Definition 4.3. Let $G = \langle R, V, E, \alpha, \gamma, \delta\rangle$ be a resource graph. Its induced ν-automaton $\mathcal{A}_G$ is defined as follows: $\mathcal{A}_G \overset{\text{def}}{=} \langle \mathsf{vars}(G), \langle Q, q_{v_\perp}, \Delta, \{q_{v_\top}\}\rangle\rangle$, with:

- $Q = \{q_{v,\rho} \mid \rho \in \Psi(G) \cup \widehat{\Psi}(G), v \in \rho \setminus \{v_\bot, v_\top\}\} \cup \{q_{v_\bot}, q_{v_\top}\}$
- $\Delta = \{(q_{v_i,\rho}, \mathsf{lbl}^{\rho}_{v_i \to v_j}, q_{v_j,\rho}) \mid \rho \in \Psi(G) \cup \widehat{\Psi}(G), v_i \to v_j \in \rho\}$
  $\cup \{(q_{v_n,\widehat{\rho}}, \mathsf{lbl}^{\rho}_{v_n \to v_e}, q_{v_e,\widehat{\rho}}), (q_{v_n,\widehat{\rho}}, \{\}, q_{v_\top}) \mid \widehat{\rho} = \langle \cdots \mid v_e, \ldots, v_n\rangle \in \widehat{\Psi}(G)\}$

$$\text{with: } \mathsf{lbl}^{\rho}_{v \to v'} = \begin{array}{l} \{\nu X \mid X \in \mathsf{first}_\rho(v') \vee (X \in \mathsf{fetch}_\rho(v') \wedge v \neq \mathsf{exit}(\rho)\} \\ \cup \{X \mid X \in \gamma(v')\} \cup \{\overline{\nu} X \mid X \in \mathsf{last}_\rho(v')\} \end{array}$$

The construction of the ν-automaton from a given resource graph relies on a decomposition of the latter in terms of its complete paths and lassos (cf. Def. 2.2). Note that there is no one-to-one correspondence between the vertices of the resource graph G and the states of the automaton. However, each vertex v of a given pat ρ is in one-to-one correspondence with a state named $q_{v,\rho}$ in $\mathcal{A}^G$. It is a simple fact that the ν-automaton corresponding to a resource graph can be of an exponential size, since we enumerate the paths of the graph. In practice, we show in [Deharbe and Peschanski 2014] that it is often possible to work directly at the graph level.

We must now show that the construction is sound, i.e. that the automaton we build from a resource graph G indeed recognizes its resource profile $\mathfrak{R}_G$.

LEMMA 4.4. *Let G a resource graph. Then $\mathbb{L}(\mathcal{A}_G) = \mathfrak{R}_G$.*

PROOF. cf. Appendix A.2 page 21. □

The ν-automaton of Fig. 3 can be easily shown *not* to correspond to any possible resource graph. It it thus interesting to characterize more precisely the sub-class of ν-automata recognizers of resource profiles. One interesting argument is that this sub-class is (strictly) contained in the *quasi-regular* languages.

THEOREM 4.5. *Let G be a resource graph. The $\mathfrak{R}_G$ is quasi-regular.*

PROOF. The proof is both non-trivial and tedious. The rough sketch is as follows. First, we provide an alternative construction of ν-automata from resource graphs, in which all the lassos are expanded exactly once. This allows to remove the *first use vs. fetch* subtlety. Then, we show that this new construction still characterizes the resource profiles. Moreover the ν-words accepted by these automata can be transformed by a simple homomorphism so that they can be recognized by finite-memory automata (FMA), demonstrating the membership result. The detailed sketch is explained in Appendix A.3 page 22. □

It is important to emphasize the obtained FMA are in an order of magnitude larger than the original ν-automata. For example the translation of the resource graph of Fig 1 (with 13 vertices) yield a ν-automaton with 32 states. In comparison, the translated FMA has 408 states, and the growth can be show exponential. Note that the translation to FMA does not work for arbitrary ν-automata.

5. RESOURCE ANALYSIS

We are now interested in quantifying the amount of resources the system under study requires to behave correctly. To address this question, we introduce an oracle – the *Omniscient Garbage Collector* (OGC) – that decides *a priori* the maximum amount of resources a system can consume. We rely on a simple although far-reaching principle : if the OGC under-estimates the consumption of the system, then a *resource conflict* would occur.

5.1. Bounded resource profiles

Although ν-language are defined over infinite alphabets, the individual ν-words may only consume a finite amount of resources.

Definition 5.1 (*Consumption*). Let $\mathbb{L}$ be a ν-language. The *consumption* of a ν-word $w = \alpha_1 \dots \alpha_n$ in $\mathbb{L}$ is $\xi(w) = \mathsf{card}\left(\bigcup_{i=1}^{n} \alpha_n\right)$. The *consumption* of the ν-language itself is $\xi(\mathbb{L}) = \mathsf{card}\left(\bigcup_{w=\alpha_1\dots\alpha_n \in \mathbb{L}} \bigcup_{i=1}^{n} \alpha_n\right)$

If a ν-word consumption is inherently finite (they have a finite length), this is not the case of ν-languages that are in most cases infinite sets of words. However, it is natural to introduce such a restricted class.

Definition 5.2 (*Bounded ν-language*). A k-bounded ν-language $\mathbb{L}_k$ is a ν-language defined over a finite alphabet $\mathcal{V}_k$ of cardinality k.

PROPOSITION 5.3. *A k-bounded ν-language has resource consumption at most k.*

PROOF. This is trivial by Def. 5.1. □

The notion of bounded ν-language can be naturally lifted to resource profiles.

Definition 5.4 (*Bounded resource profile*). A k-bounded resource profile $\mathfrak{R}_G^k$ of a resource graph G is a k-bounded ν-language satisfying Def. 3.4. The measure k is named the *resource bound* of the graph G.

This restricted definition implies a major requirement for the resource bound k: it must be large enough so that the constraints imposed by Def. 3.4 can be fulfilled.

An important property of bounded ν-language is that they are insensitive to bijective renamings of pure names.

Definition 5.5 (*Renaming*). Let $\mathcal{V}$ and $\mathcal{V}'$ be two disjoint sets of pure names. A *renaming* ζ is a mapping of $\mathcal{V} \rightarrow \mathcal{V}'$. The renaming by ζ of :

— a ν-symbol $\alpha \subseteq \mathcal{V}$ is $\zeta(\alpha) = \{\zeta(\nu) \mid \nu \in \alpha\}$,
— a ν-word $w = \alpha_1 \cdots \alpha_n \in (2^{\mathcal{V}})^*$ is $\zeta(\alpha_1) \cdots \zeta(\alpha_n)$,
— a ν-language $\mathbb{L}$ over alphabet $\mathcal{V}$ is $\zeta(\mathbb{L}) = \{\zeta(w) \mid w \in \mathbb{L}\}$.

PROPOSITION 5.6. *Let G be a resource graph, and $\mathcal{R}_G^k$, $\mathcal{R}_G^{\prime k}$ two k-bounded resource profiles defined over respective alphabets $\mathcal{V}_k$ and $\mathcal{V}_k'$. Then there is a bijective renaming ζ of $\mathcal{V}_k \rightarrow \mathcal{V}_k'$ such that $\mathcal{R}_G^k = \zeta(\mathcal{R}_G^{\prime k})$.*

PROOF. It is a simple fact that in Def. 3.4 only the equality of the pure names in $\mathcal{V}$ is exploited, which is the basic principle of the pure names. As such, if $\mathcal{R}_G^k$ is assumed to be a k-bounded resource profile, then renaming its pure names by another set of at least k pure names would not contradict Def. 3.4. □

From now on, we will thus consider *the* k-bounded resource profile $\mathfrak{R}_G^k$, implicitly considering the whole family of resource profiles that can be obtained by bijective renamings of their alphabets.

The central question remains: is there in general a bound k such that $\mathfrak{R}_G^k$ is defined? In this paper, we only study finite resource graph, which means that we only capture processes that consume only a finite amount of resources at any given point of execution. Moreover, this amount must be bound by the *memory* of the resource graph, i.e. its resource variables. Thus a worst-case bound does exists, and it is determined thanks to the following Lemma.

LEMMA 5.7. *Let G be a resource graph such that $\mathsf{inactive}(G) = \emptyset$. If $k = \mathsf{card}(\mathsf{vars}(G))$ then $\mathfrak{R}_G^k$ exists and is recognized by $\mathcal{A}_G^k$ which is automaton $\mathcal{A}_G$ (of Def. 4.3) restricted to alphabet $\mathcal{V}_k = \{v_X \mid X \in \mathsf{vars}(G)\}$.*

PROOF. The principle of the proof is to apply a renaming of the alphabet that associates a single pure name ν_X to each resource variable X of $\mathsf{vars}(G)$. The complete proof is in Appendix A.4 page 23. □

Corollary 5.8. $\mathsf{card}(\mathsf{vars}(G))$ *is a resource bound for resource graph G.*

This provides us with a starting point for our quantitative resource bound analysis. The bound $k = \mathsf{card}(\mathsf{vars}(G))$ is the *nominal resource bound* of G. Before going further in our quantitative study, we provide alternative – and arguable simpler – proofs for some results presented in the companion paper [Deharbe and Peschanski 2014].

Theorem 5.9. *Let G be a resource graph with resource bound k. Then $\mathfrak{R}_G^k$ is regular.*

Proof. This is a simple Corollary of Lemma 4.5, for the languages of FMA over finite alphabets are proved regular in [Kaminski and Francez 1994]. □

Corollary 5.10. *Bounded resource profile equivalence is PSPACE-hard.*

5.2. The Omniscient Garbage Collector

Obviously, a process cannot require more resources than the amount of available memory. However, it may be the case that *less* memory is enough so that it still behaves correctly. One thing leading to another, we may ask what is the miminal amount of needed memory. To adress these questions, we develop in this section a static analysis that can determinate lower bounds of resource consumption direcly from the resource graphs. The objective is to compute a measure k that guarantees the existence of a k-bounded resource profile for a given graph G. The greatest of such bounds is of course the number of resource variables of G, according to Lemma 5.7. To decrease the bound, the only possible way is thus to *unify* variables of the translated ν-automata.

Definition 5.11 (*Unifying variables in ν-automata*). Let $\mathcal{A} = \langle \mathcal{X}, A \rangle$ be a ν-automaton with $A = \langle Q, q_{\mathsf{init}}, \Delta, F \rangle$. The unification of variables $E \subseteq \mathcal{X}$ with $Z \notin \mathcal{X}$ is:

$$\mathsf{unify}_{E \lhd X}(\mathcal{A}) = \langle (\mathcal{X} \setminus E) \cup \{Z\}, \mathsf{unify}_{E \lhd X}(A) \rangle \text{ provided:}$$

$$\left[\begin{array}{l} \mathsf{unify}_{E \lhd X}(A) = \langle Q, q_{\mathsf{init}}, \mathsf{unify}_{E \lhd X}(\Delta), F \rangle \\ \mathsf{unify}_{E \lhd X}(\Delta) = \{(q, \{\mathsf{unify}_{E \lhd X}(l) \mid l \in \lambda\}, q') \mid (q, \lambda, q') \in \Delta\} \\ \mathsf{unify}_{E \lhd X}(X) = \begin{cases} Z \text{ if } X \in E \\ X \text{ otherwise} \end{cases} \\ \mathsf{unify}_{E \lhd X}(\nu X) = \nu \mathsf{unify}_{E \lhd X}(X) \\ \mathsf{unify}_{E \lhd X}(\overline{\nu} X) = \overline{\nu} \mathsf{unify}_{E \lhd X}(X) \end{array} \right.$$

Let $\Pi = \{E_i \mid i \in [1;n]\}$ be a partition of $\mathsf{vars}(G)$, and let $\mathcal{Z} = \{Z_1 \ldots Z_n\}$ a set of n variables distinct from those $\mathsf{vars}(G)$. Then $\mathsf{unify}_{\Pi \lhd \mathcal{Z}}(\mathcal{A}) = \mathsf{unify}_{E_{m_1} \lhd Z_{m_2}}(\ldots \mathsf{unify}_{E_{m_1} \lhd Z_{m_n}}(\mathcal{A}))$ for an arbitrary permutation $m_1, \ldots, m_n$ of $[1;n]$.

The unification process is relatively technical but intuitively quite simple: each reference to any of the variables of E in the automaton is replaced by the fresh variable Z. The unification is also lifted to partitions of the set of variables. The unification of each indepedent subset can be performed in an arbitrary order (trivially the outcome is the same since the sets are disjoint).

A fundamental requirement is that the unified variables must correspond to non-conflicting resources.

Lemma 5.12. *Let G be a resource graph and $\mathfrak{R}_G^k$ its bounded resource profile such that $k = \mathsf{card}(\mathsf{vars}(G))$, and $\mathcal{A}_G^k$ its recognizer. Moreover let $E \subseteq \mathsf{vars}(G)$ such that, $\forall X_i, X_j \in E,\ i \neq j \implies \neg(X_i \sharp X_j)$. Then $\mathfrak{R}_G^{k - \mathsf{card}(E)+1}$ is recognized by $\mathsf{unify}_{E \lhd Z}(\mathcal{A}_G^k)$ provided $Z \notin \mathsf{vars}(G)$.*

Proof. The proof is similar to that of Lemma 5.7, except that we do not consider a simple renaming of pure name, but a unification of variables. The proof details are in Appendix A.5 page 24. □

Similarly to the unification process, the reduction scheme can be lifted to the partition of the resource variables wrt. the conflict relation.

Lemma 5.13. *Let G be a resource graph and $\mathfrak{R}_G^k$ its bounded resource profile such that $k = \mathsf{card}(\mathsf{vars}(G))$, and $\mathcal{A}_G^k$ its recognizer. Moreover let $\Pi = \{E_1, \ldots, E_n\}$ a partition of $\mathsf{vars}(G)$ such that, $\forall E = \{X_1, \ldots, X_m\} \in \Pi, \forall i, j, i \neq j \implies \neg(X_i \sharp X_j)$. Then $\mathfrak{R}_G^{\mathsf{card}(\Pi)}$ is recognized by $\mathsf{unify}_{\Pi \triangleleft \mathcal{Z}}(\mathcal{A}_G^k)$ provided $\mathcal{Z} \cap \mathsf{vars}(G) = \emptyset$.*

Proof. cf. Appendix A.5 page 24. □

Of course, the reduction process must stop at some point, since a minimal amount of memory is required for a process to behave correctly. Hence, a resource graph G has a minimal resource bound which we name its *resource index.* The basic principle is to minimize the parameter $\mathsf{card}(\Pi)$ of Lemma 5.13.

Lemma 5.14 (Resource index). *Let G be a resource graph and Π the set of maximal independent subsets of $\mathsf{vars}(G)$ wrt. the conflict relation $\sharp$. Then $\mathsf{card}(\Pi)$ is the* resource index *of G.*

Proof. cf. Appendix A.6. □

From an algorithmic point of view, the computation of the maximal indepedents sets is based on *grap coloring* [Jensen and Toft 2011]. In Fig. 2, the numbered labels of the nodes correspond to colorings of the conflicts corresponding to the resource graph of Fig. 1. The numbers on the left (before the open parenthesis) correspond to *first-fit coloring* using the node ordering H, G, A, B, C, D, E. First, H can be colored by (location) 1 and so is G since it is not connected to H. Next, A and B must use color 2 since they are connected to G. The color 1 can be reused for C since it is not yet connected to a colored node. The node D is connected to C (color 1) and B (color 2) and thus must be colored 3. Finally, E is connected to nodes colored up-to 3 and thus has color 4. The independent sets we consider form the partition $\Pi_{\mathsf{firstfit}} = \{\{C, G, H\}, \{A, B\}, \{D\}, \{E\}\}$. The corresponding resource bound is 4.
To obtain the maximal independent sets, we require the *perfect coloring* of the relation graph. In Fig. 2 this corresponds to the numbers within parentheses. The strategy here is to use the color 2 for both B and C. This way D can reuse color 1 and thus E has color 3 instead of 4. We obtain the partition $\Pi_{\mathsf{perfect}} = \{\{D, G, H\}, \{A, B, C\}, \{E\}\}$. The resource index of the resource graph is thus 3 (which is also the *chromatic number* of the conflict graph). Unfortunately, the finding of a perfect coloring is notoriously a difficult problem.

Theorem 5.15. *Computing the resource index of a resource graph is NP-complete.*

Proof. cf. e.g. [Jensen and Toft 2011] for a detailed proof. □

This can be seen as a somewhat negative result, although we remark that the perfect coloring algorithm only applies to the conflict graph and not the complete resource graph. In most practical cases the former should be much smaller than the latter. Moreover, interesting properties of separability can often be exploited (cf. [Deharbe and Peschanski 2014]). Last but not least, less tight but still interesting resource bounds can be found in polynomial time. One such example is through the use of first-fit coloring.

Proposition 5.16. *Let G be a resource graph with conflict graph $\sharp$. A resource bound for G lower than $d_G + 1$ where $d_G \overset{def}{=} \mathsf{max}_{X \in \mathsf{vars}(G)}\{Y \mid X \sharp Y\}$ can be computed in linear time in the size of $\sharp$.*

$$\begin{array}{lll}
\multicolumn{3}{l}{\textbf{Process } P, Q} \\
::= & 0 & \text{(inert)} \\
\mid & \mathsf{new}(x)\ P & \text{(observable)} \\
\mid & \mathsf{local}(x)\ P & \text{(inobservable)} \\
\mid & \alpha.P & \text{(prefix)} \\
\mid & P \mid Q & \text{(parallel)} \\
\mid & D[\tilde{a}] & \text{(call)}
\end{array}
\qquad\Bigg|\qquad
\begin{array}{lll}
\multicolumn{3}{l}{\textbf{Definition}} \\
::= & D(\tilde{x}) \overset{\text{def}}{=} P & \\
\multicolumn{3}{l}{\textbf{Action } \alpha} \\
::= & \tau & \text{(silent)} \\
\mid & \overline{a}b & \text{(output)} \\
\mid & a(x) & \text{(input)}
\end{array}$$

Fig. 4. The syntax of the π-calculus (with slices).

$$\frac{}{\tau.P \xrightarrow{\tau} P}\ \text{(step)} \qquad \frac{}{\overline{a}b.P \xrightarrow{\overline{a}b} P}\ \text{(output)} \qquad \frac{x \notin \mathsf{fn}(P)}{a(x).P \xrightarrow{ay} P}\ \text{(input)}$$

$$\frac{P \xrightarrow{\overline{a}x} P' \quad a \neq x \quad \mathsf{res} \in \{\mathsf{new}, \mathsf{local}\}}{\mathsf{res}(x)\ P \xrightarrow{\overline{a}\nu x} P}\ \text{(open)} \qquad \frac{P \xrightarrow{\alpha} P' \quad x \notin \alpha \quad \mathsf{res} \in \{\mathsf{new}, \mathsf{local}\}}{\mathsf{res}(x)\ P \xrightarrow{\alpha} \mathsf{res}(x)\ P'}\ \text{(res)}$$

$$\frac{P \xrightarrow{\alpha} P' \quad \mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \text{(par)} \qquad \frac{P \xrightarrow{ab} P' \quad Q \xrightarrow{\overline{a}b} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \text{(sync)}$$

$$\frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q \equiv Q'}{P \xrightarrow{\alpha} Q}\ \text{(struct)}$$

Fig. 5. The early labelled transition semantics of the π-calculus.

Proof. Let d be the maximum out-degree of a graph. It is a classical result that a k-coloring bounded by $d+1$ can be computed in linear time by a first-fit coloring based on an arbitrary ordering of the graph vertices. □

6. APPLICATION: RESOURCE ANALYSIS OF π-CALCULUS PROCESSES

In this section we describe the experimental application of our framework for the analysis of resource consumption in π-calculus processes.

6.1. A π-calculus refresher

The syntax of the variant of the π-calculus we cover in the experiment[1] is given in Fig. 4. We also remind the structural congruence between two processes, which is the least relation on processes satisfying:

— $P \equiv Q$ by a renaming of bound variables
— $P \mid Q \equiv Q \mid P$, $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$ and $P \mid 0 \equiv P$
— $D[\widetilde{a}] \equiv P\{\widetilde{a}/\widetilde{x}\}$ if $D(\widetilde{x}) \overset{\text{def}}{=} P$
— for $\mathsf{res} \in \{\mathsf{new}, \mathsf{local}\}, \mathsf{res}(x)\ (P \mid Q) \equiv P \mid \mathsf{res}(x)\ Q$ provided $x \notin \mathsf{free}(P)$

The semantics of the language is recalled in Fig.5. Informally, the process 0 has no transition. The scope of a name x can be restricted by either $\mathsf{new}(x)$ of $\mathsf{local}(x)$ and for now, the two constructs are assumed synonymous (this will be different in reduction semantics). A prefixed process $\alpha.P$ denotes a transition with a label corresponding to the action α and continuing as process P. There are four kinds of labels depending on the action α:

[1] For the sake of concision, we omit the constructs of non-deterministic choice and match/mismatch. Note that our prototype tool has support for both

— a label τ is generated by a silent action τ or by a synchronization.
— a label ab is generated by an input action $a(b)$ for any name b received along channel a and bound to the variable x (in early semantics).
— a label $\overline{a}b$ is generated by an output action $\overline{a}b$ of datum b along channel a, under the provision that a and b are not restricted (i.e. in the scope of a new or a local.
— a label $\overline{a}\nu b$ is a bound output generated by an output action $\overline{a}b$ where b is restricted, unlike a.

The construct $P \mid Q$ expresses the parallel composition (in terms of interleaving) of the sub-processes P and Q. These cover the independent evolution of the processes, of alternatively the synchronization for a composition of the form $\overline{a}b.P \mid a(x).Q$. The latter generates a transition with label τ and a continuation of the form: $P \mid Q\{a/x\}$. Finally, the language has *tail calls* that corresponds to possibly recursive unfoldings of process definitions.

6.2. Abstracting transition labels

The first step of our experiment is to generate a resource graph that reflects the behavior of a π-calculus process in terms of resource usage. A natural interpretation consists in interpreting almost directly the labelled transition system (LTS) as a resource graph. Under this interpretation, each transition $P \xrightarrow{\mu} Q$ is associated to three vertices v_P, v_μ and v_Q and the edges (v_P, v_μ) and (v_μ, v_Q). The resource usage is then specified by the values associated to $\alpha(v_\mu)$, $\gamma(v_\mu)$ and $\delta(v_\mu)$. Schematically, we have:

— a transition $P \xrightarrow{\overline{a}\nu b} Q$ creates a resource X_b and is interpreted as:

$$v_P \rightarrow \boxed{\alpha\{X_b\},\gamma\{X_b\},\delta\{\}} \rightarrow v_Q$$

— a transition $P \xrightarrow{\overline{a}b} Q$ such that there is a resource X_b for b is interpreted as:

$$v_P \longrightarrow \boxed{\alpha\{\},\gamma\{X_b\},\delta\{\}} \longrightarrow v_Q$$

— any other transition $P \xrightarrow{\mu} Q$ is interpreted as:

$$v_P \longrightarrow \boxed{\alpha\{\},\gamma\{\},\delta\{\}} \longrightarrow v_Q$$

In this first abstraction, the rationale is: *every data sent to the environment count as resource uses*. Hence, any bound output counts as the creation of a fresh resource as well as a use, and each output of a name associated to a resource counts as a simple use. There are possible variations, such as counting the channel itself as a use (e.g. recording a use with $\overline{b}a$ in case b is associated to a resource X_b), or also taking input into consideration. It is then possible to distinguish between input or output resource uses. In all these possible interpretations, the *leitmotiv* is that resource profile equivalence should be a necessary (although insufficient) condition for bisimilarity[2]. We also require the destruction of resources through δ's. A simple and effective heuristic is to insert a $\delta\{X_b\}$ when there is no further free occurrence of the name b in the process.

Let us consider as a first example the following process:

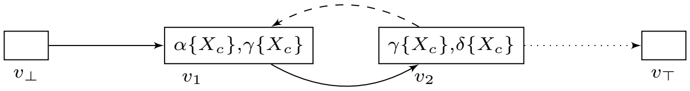
$$P \overset{\text{def}}{=} \mathsf{new}(c)\ \overline{a}c.\overline{b}c.P$$

This is a special case of a common pattern for generating fresh names. Here, the restricted name c is sent first along a and then b towards the environment. The whole process is then iterated, leading to the following derivations:

[2] We do not provide in this report a formal proof that "*bisimilarity* implies *resource profile equivalence*" but this is rather trivial since the traces of resource profiles encode all the information stored in the transition labels.

$$P \xrightarrow{\overline{a}\nu c} \overline{b}c.P \xrightarrow{\overline{b}c} P \to \cdots$$

The first output along a corresponds to a bound output since c is restricted but the further output is not bound anymore. Given a resource variable X_c representing the name c once required fresh, we obtain the following resource graph:


$\alpha\{X_c\}$,$\gamma\{X_c\}$
$\gamma\{X_c\}$,$\delta\{X_c\}$
$v_\bot$
v_1
v_2
$v_\top$

A theoretically acceptable alternative would to have an infinite system generating an infinite number of resources. Although the version with the least fixpoint shows that exactly one resource is required for this behavior, the resource index is invariantly 1 because there can be no conflict for this process in any acceptable interpretation.

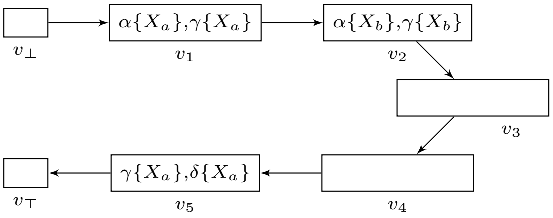
A minimal conflict can be generated by e.g.:

$$\mathsf{new}(a)\ \mathsf{new}(b)\ \overline{c}a.\overline{c}b.\overline{c}a.0$$

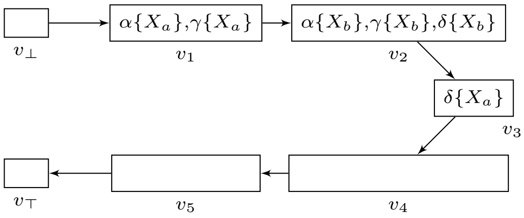
A slightly complexified variant of this process is as follows:

$$\left[\begin{array}{l} Q \overset{\text{def}}{=} \mathsf{new}(a)\ \mathsf{new}(b)\ \overline{c}a.\overline{c}b.\overline{d}a.d(x).\overline{c}x.0 \\ C[X] \overset{\text{def}}{=} \mathsf{new}(d)\ \ [Q \mid X] \end{array}\right.$$

The resource graph corresponding to $C[d(y).\overline{d}y.0]$ is[3]:


$\alpha\{X_a\}$,$\gamma\{X_a\}$
$\alpha\{X_b\}$,$\gamma\{X_b\}$
$v_\bot$
v_1
v_2
v_3
$\gamma\{X_a\}$,$\delta\{X_a\}$
$v_\top$
v_5
v_4

This maintains the conflict $X_a \sharp X_b$ and thus the resource index of the system is 2, whereas if we consider the variant $C[d(y).\overline{d}c.0]$ then the resource graph becomes:


$\alpha\{X_a\}$,$\gamma\{X_a\}$
$\alpha\{X_b\}$,$\gamma\{X_b\}$,$\delta\{X_b\}$
$v_\bot$
v_1
v_2
$\delta\{X_a\}$
v_3
$v_\top$
v_5
v_4

The conflict $X_a \sharp X_b$ is no more and hence the resource index is 1 in this case. This illustrates the profoundly semantic nature of the proposed resource abstraction. Indeed, the behavior of X within the context $C[X]$ can be as complex as required so that in the general case (beyond finite control) one cannot decide whether the conflict should take place or not.

[3] A single-hole process context $C[X]$ is a function from process expression extended by a single occurrence of a variable X to process expressions, such that $C[P] = C[X]\{P/X\}$ for a standard notion of substitution of variables by processes. Here for example $C[d(y).\overline{d}y.0] = \mathsf{new}(d)\ \left[Q \mid d(y).\overline{d}y.0\right]$.

Model	LTS	Resource graph	Resource index
heap_4	700	86	3
heap_5	8476	303	4
heap_6	126125	1094	5
buffer_4	596	339	4
buffer_5	7173	3621	5
buffer_6	106878	49246	6
GSM	489	56	3
$\mathsf{GSM}_{\mathsf{buff}}$	164	56	3
$\mathsf{GSM}_{\mathsf{full}}$	2183	56	3

Fig. 6. Experimental results for the resource abstraction on labelled transitions.

This abstraction has been implemented in a prototype tool and we analyzed several examples from the *HAL environment* [Ferrari et al. 1998]. At present, the tool only support finite control processes and the construction of the resource graph is purely semantic. Since we do not need to preserve the whole branching structure, we can apply a few heuristics to reduce the size of the resource graphs, but in the worst case it can be as large as (although no larger than) the full LTS e.g. as produced by HAL. The problem of producing the smallest possible resource graph is open and we conjecture that its complexity is high. Fig. 6 gives the figures we obtain for the examples that are particularly interesting for the considered abstraction. For each example, we give the size of the LTS produced by HAL and we compare it with the size of the resource graph we obtain. This measure is not really significant but it still emphasizes the fact that there is an important potential of abstraction when constructing the resource graphs. A metric much more significant is the resource index that we obtain using our omniscient garbage collector. A detailed comment of the results is provided in [Deharbe and Peschanski 2014] but the main outcome of the experiment is that the conflict graph in all the examples is very small, hence its perfect coloring is always an affordable task. Moreover, the resource index always convey an important information regarding the process behaviors. As an illustration, we can relate the number n of competing cells in heap_{n+1} (resp. buffer_n) to the resource index n. In the GSM cases, the resource graph (and thus resource index) does not change, which says that despite their important syntactical difference, they all exhibit exactly the same resource usage.

6.3. Refining reductions

Abstracting from the labelled transitions is quite natural but requires a very powerful observer. In comparison, the reduction semantics are much less demanding. However, they only apply on closed systems. An intermediate approach is to model part of the observer within the system. For this we allow a process behavior to be *sliced* from the point of view of the environment. A process of the form $\mathsf{local}(x)\ P$ considers x as a classical π-calculus restriction but explicitly decorated by a tag *inobservable*. In comparison, in $\mathsf{new}(x)\ P$ the name x is tagged *observable*. Names can also be assigned the tag *observed* although not in their initial state. Now, a standard reduction $P \to Q$ of the π-calculus is refined so that it produces a "labelled" reduction of the form $\Lambda \vdash P \xrightarrow{\mu} \Lambda' \vdash Q$ in the slice-π variant. The component Λ is a set of names tagged as *observable*. A name a with the *observable* tag but lacking the *observed* tag is such that $a \in \Lambda$. Otherwise, if it is observable *observed* then $\underline{a} \in \Lambda$. Of course, it cannot be the case that $\{a, \underline{a}\} \subseteq \Lambda$.

Each reduction $P \to Q$ is now reinterpreted as either:

— an *open reduction* of the form:

$$\frac{}{\Lambda \vdash \tau.P \xrightarrow{\bullet} \Lambda \vdash P}\ \text{(step)}$$

$$\frac{\Lambda \cup \{x\} \vdash P \xrightarrow{\mu} \Lambda' \vdash P' \quad x \notin \Lambda}{\Lambda \vdash \mathsf{new}(x)\ P \xrightarrow{\mu} \Lambda' \vdash P'}\ \text{(obs)} \qquad \frac{\Lambda \vdash \xrightarrow{\mu} \Lambda' \vdash P' \quad x \notin \Lambda}{\Lambda \vdash \mathsf{local}(x)\ P \xrightarrow{\mu} \Lambda' \vdash P'}\ \text{(inobs)}$$

$$\frac{a \in \Lambda \cup \{b\} \vee \underline{a} \in \Lambda}{\Lambda \vdash \{b\} \vdash \overline{a}b.P \mid a(x).Q \xrightarrow{\langle b \rangle} \Lambda \cup \{\underline{b}\} \vdash P \mid Q\{b/x\}}\ \text{(sync-fresh)}$$

$$\frac{a \in \Lambda \vee \underline{a} \in \Lambda \quad b \notin \Lambda}{\Lambda \vdash \overline{a}b.P \mid a(x).R \xrightarrow{\langle b \rangle} \Lambda \cup \{b\} \vdash P \mid R\{b/x\}}\ \text{(sync-open)}$$

$$\frac{a \in \Lambda \vee \underline{a} \in \Lambda \cup \{b\}}{\Lambda \cup \{\underline{b}\} \vdash \overline{a}b.P \mid a(x).Q \xrightarrow{b} \Lambda \cup \{\underline{b}\} \vdash P \mid Q\{b/x\}}\ \text{(sync-obs)}$$

$$\frac{a \notin \Lambda}{\Lambda \vdash \overline{a}b.P \mid a(x).R \xrightarrow{\bullet} \Lambda \vdash P \mid R\{b/x\}}\ \text{(sync-inobs)}$$

$$\frac{\Lambda \vdash P \xrightarrow{\mu} \Lambda' \vdash P'}{\Lambda \vdash P \mid Q \xrightarrow{\mu} \Lambda' \vdash P' \mid Q}\ \text{(par)} \qquad \frac{P \equiv P' \quad \Lambda \vdash P' \xrightarrow{\mu} \Lambda' \vdash Q' \quad Q \equiv Q'}{\Lambda \vdash P \xrightarrow{\mu} \Lambda' \vdash Q}\ \text{(struct)}$$

Fig. 7. The semantics of the slice-π calculus.

$\Lambda \vdash P \xrightarrow{\langle b \rangle} (\Lambda \setminus \{b\}) \cup \{\underline{b}\} \vdash Q$ when the reduction is a synchronization passing an *observable* or *inobservable* but not yet *observed* name b along an observable channel a. As a side-effect, the name b is tagged as *observable* and also as *observed*.

— a *transparent reduction* of the form:
$\Lambda \vdash P \xrightarrow{b} \Lambda \vdash Q$ when the reduction is a synchronization passing an *observed* name b (i.e. $\underline{b} \in \Lambda$) along an observable channel (i.e. $a \in \Lambda$).

— an *opaque reduction* of the form:
$\Lambda \vdash P \xrightarrow{\bullet} \Lambda \vdash Q$ in any other case.

The complete operational semantics of the slice-π calculus is provided in Fig. 7. In terms of resource graphs, the interpretation is now quite similar to the labelled abstraction:

— a reduction $\Lambda \vdash P \xrightarrow{\langle b \rangle} \Lambda' \vdash Q$ creates a resource X_b and is interpreted as:

$$v_{\Lambda \vdash P} \longrightarrow \boxed{\alpha\{X_b\},\gamma\{X_b\},\delta\{\}} \longrightarrow v_{\Lambda' \vdash Q}$$

— a reduction $\Lambda \vdash P \xrightarrow{b} \Lambda \vdash Q$ such that there is a resource X_b for b is interpreted as:

$$v_{\Lambda \vdash P} \longrightarrow \boxed{\alpha\{\},\gamma\{X_b\},\delta\{\}} \longrightarrow v_{\Lambda \vdash Q}$$

— any other reduction $\Lambda \vdash P \xrightarrow{\bullet} \Lambda \vdash Q$ is interpreted as:

$$v_{\Lambda \vdash P} \longrightarrow \boxed{\alpha\{\},\gamma\{\},\delta\{\}} \longrightarrow v_{\Lambda \vdash Q}$$

$$
\begin{array}{l}
\{\} \vdash \mathsf{new}(a)\ \mathsf{new}(b)\ S(a,b) \\
\xrightarrow{\langle x_1\rangle} \{a,b,\underline{x_1}\} \vdash P(a) \mid Q'(a,b) \mid Q(a,b) \mid R(b) \\
\xrightarrow{\langle x_2\rangle} \{a,b,\underline{x_1},\underline{x_2}\} \vdash P(a) \mid Q'(a,b) \mid Q'(a,b) \mid R(b) \\
\xrightarrow{x_1} \{a,b,\underline{x_1},\underline{x_2}\} \vdash P(a) \mid Q(a,b) \mid Q'(a,b) \mid R(b) \\
\xrightarrow{x_2} \{a,b,\underline{x_1},\underline{x_2}\} \vdash S(a,b) \\
\ldots
\end{array}
$$

$$
\begin{array}{l}
\{\} \vdash \mathsf{local}(a)\ \mathsf{new}(b)\ S(a,b) \\
\xrightarrow{\bullet} \{b,x_1\} \vdash P(a) \mid Q'(a,b) \mid Q(a,b) \mid R(b) \\
\xrightarrow{\bullet} \{b,x_1,x_2\} \vdash P(a) \mid Q'(a,b) \mid Q'(a,b) \mid R(b) \\
\xrightarrow{\langle x_1\rangle} \{b,\underline{x_1},x_2\} \vdash P(a) \mid Q(a,b) \mid Q'(a,b) \mid R(b) \\
\xrightarrow{\langle x_2\rangle} \{b,\underline{x_1},\underline{x_2}\} \vdash S(a,b) \\
\ldots
\end{array}
$$

Fig. 8. Reductions of slice-π processes with a observable (top) or inobservable (bottom).

To illustrate the abstraction, we consider the processes $\mathsf{new}(a)\ \mathsf{new}(b)\ S(a,b)$ vs. $\mathsf{local}(a)\ \mathsf{new}(b)\ S(a,b)$ with:

$$
\left[
\begin{array}{l}
P(a) \stackrel{\text{def}}{=} \mathsf{new}(x)\ \overline{a}x.P(a) \\
Q(a,b) \stackrel{\text{def}}{=} a(y).Q'(a,b,y) \\
Q'(a,b,y) \stackrel{\text{def}}{=} \overline{b}y.Q(a,b) \\
R(b) \stackrel{\text{def}}{=} b(z).R(b) \\
S(a,b) \stackrel{\text{def}}{=} P(a) \mid Q(a,b) \mid Q(a,b) \mid R(b)
\end{array}
\right.
$$

Fig. 8 shows representative reductions of the first process with a observable (on the left) and a inobservable (on the right). In the observable case the names a and b are recorded in the first reduction as observable (i.e. put explicitly in the Λ component of the state). In the same reduction, the name x generated by P is opened (i.e. marked *observed*) by the synchronization with the leftmost process Q. A "second" x is opened in the next reduction by the synchronization between P and the rightmost Q. The "two" x's must be alpha-converted hence the introduction of x_1 and x_2 in the reductions. The Λ component of the transition contains $\{\underline{x_1},\underline{x_2}\}$ because these two observable names are actually observed. If we compare this behavior with the one of the right-hand side, a is there tagged *inobservable* since it is introduced by the local construct. This means it is not a member of the component Λ of the state, unlike b. Hence the names x_1 and x_2 are now introduced as observable but not yet observed because they are transmitted along a. In terms of resources graphs, the left-hand side reductions yield a conflict $X_{x_1} \sharp X_{x_2}$ that is absent in the rightmost process. The processes have indeed a different resource index: respectively 2 and 1. We thus obtain a level of flexibility that is quite comparable to the labelled abstraction, but without the need for an idealistically powerful observer.

Based on this abstraction, we designed a simple example inspired by the classical *dining philosophers* problem. The idea is that the environment is modeled as a process that acknowledges through an observable channel eat the fact that a philosopher actually starts eating. All the other channels (ending points for the philosophers, the forks, etc.) are created inobservables (hence restricted with local instead of new). The resource conflicts occur when distinct philosophers eat at the same time on the table, by transmitting the philosopher

Model	# Processes	Reductions	Resource graph	Resource index
philos_2	7	133	20	1
philos_3	10	2992	136	1
philos_4	13	98245	4148	2

Fig. 9. Experimental results for the resource abstraction on reductions.

channel along the environment observable eat. As a side effect, the philosophers, initially *inobservable*, inherit both the *observable* and *observed* tags in a dynamic way.

The results for some instances of the philo_n examples are listed in Fig. 9. The size of the reduction graph grows exponentially since we modelled various sub-processes running in parallel (e.g. 13 processes for philo_4). The resource graphs we obtain using similar heuristics as in the labelled case are much smaller but in a similar order of magnitude in terms of growth. The resource index (and hence the maximum conflict) is quite reassuring in that the number of philosophers actually competing for food remains below the number of fork pairs, ensuring the correctness of the protocol. Although simpler analyzes are of course possible for this specific example, the experiment emphasizes the fact that the resource index captures a deep semantic information, tightly related to the chosen resource abstraction.

Last but not least, none of our experiments (except those made *on purpose*) expose a large resource index. In fact, the perfect coloring of the conflict graphs was almost immediate in all the examples, despite the high complexity of the algorithm. In the current version of the tool we use a simple and rather slow CSP-solver for the task. This largely covers our current needs but state-of-the art SAT solvers could be used for more demanding scenarios. In cases perfect coloring would become unfeasible, we can still compute less tight but still interesting resource bounds very efficiently, using e.g. first-fit coloring.

7. RELATED WORK

Resource control and analysis is a vast topic of research. Considered in their purest form, resources are *pure names* naturally leading to *nominal calculi* [Gordon 2000] in general, and in particular the π-calculus [Sangiorgi and Walker 2001] and its numerous variants. This is a rather abstract and open-ended setting, thus not a very prolific source of effective analysis algorithms. One approach is to enrich the semantics, as e.g. in [Amadio and Dal-Zilio 2006] where a resource bound analysis is proposed for a reactive synchronous variant of the π-calculus. For more classical (and abstract) variants, related studies address decidability issues often in connection with Petri nets, such as e.g. [Amadio and Meyssonnier 2002; Rosa-Velardo and de Frutos-Escrig 2010] and [Hüchting et al. 2013]. The latter introduces the *name-bounded processes*, a significant class of infinite-state systems for which the boundedness question is answered positively. It is particularly remarkable that reachability is also decidable for this class. In comparison, we *assume* the finiteness of resource graphs, and deliberately de-emphasize the means by which they are obtained practically. Indeed, a key feature of our framework is its independence from any particular formalism. Furthermore, for a given formalism multiple resource abstractions can be experimented as illustrated in Section 6. The abstraction of *active restrictions* proposed in [Hüchting et al. 2013] only applies on reductions for closed systems. It is also different from the resource model we propose around the slice-π calculus, and to illustrate this aspect we consider the following process:

$$P(a) \overset{\text{def}}{=} \mathsf{new}(x)\,[\overline{a}x.0 \mid a(y).P(y)] \mid \tau.\mathsf{new}(z)\ \overline{z}a.0$$

In the abstraction we propose, the resource index of $\mathsf{new}(a)\ P(a)$ is 1 because the processes $\mathsf{new}(z)\ \overline{z}a$ are deadlocked after the initial τ. However, since the name a is always free in these deadlocked processes the whole process has an infinite number of active restrictions.

This particular example can be of course optimized but the deadlocked process can be complexified at will. Hence, we discuss a finer-grained abstraction that cannot be decided *locally*. Relying on an essentially semantic abstraction is not without consequences. For instance, our current implementation only works with finite control π-calculus processes. It is a very intriguing and open question whether interesting sub-classes of infinite systems with finite resource graphs could be determined, probably starting with variants of the name-bounded class itself. Another related abstraction is that of *barbed semantics* [Sangiorgi and Walker 2001] that also refine reductions but considering in this case the non-restricted channels as observables. This is to ultimately characterize an adequate notion of process equivalence – namely *strong barbed congruence* – when the reductions with observables are closed under context. While we could observe the channels instead of (or together with) the data, we require our refinement to remain in one-to-one correspondence with the plain reductions. Also particularly notable in [Hüchting et al. 2013] is the prominent role played by the notion of *garbage collection* something already observed in e.g. the *history-dependent-automata* [Ferrari et al. 1998] or in the π-graphs [Peschanski et al. 2013]. This is a side note but to our knowledge, HD-Automata Laboratory (HAL) is the only tool allowing the generation of early labelled transition system from (finitary) π-calculus processes. Indeed, the generation of the early LTS is not trivial especially because it requires the determination of the *active names* [Montanari and Pistore 1995], a notion tightly connected to the *live variables* of resource graphs.

Graph coloring relates to the very well-known problem of *register allocation* in compiler back-ends [Chaitin 2004]. However, the behavior of registers is quite specific. For example, one can always choose *not* to allocate a register, or release it prematurely and defer to the central memory. Hence, the coloring can be both partial *and* imperfect, allowing many optimization heuristics that do not apply at all in our case. This still naturally connects our study with the well-studied notion of *register automata* and related formalisms, especially finite memory automata (FMA) [Kaminski and Francez 1994] and fresh register automata (FRA) [Tzevelekos 2011]. Although the theory of ν-automata shall be further investigated, we suggest in the paper that they represent quite an expressive formalism. For instance, the automaton depicted in Fig 3 can be easily shown *not* simulable by either a FMA or a FRA. However, we show that resource profiles are quasi-regular languages recognizable by FMA. In this specific case, the ν-automata are still relevant since they can trivially be shown exponentially smaller than their FMA counterparts. To our knowledge, the problem of reducing the memory of FMA has only been investigated in the deterministic case [Benedikt et al. 2010]. Our study suggests an approach for non-deterministic FMA but only to reduce the storage size. Finally, we think that ν-automata represent an interesting formalism to address resource control issues as in e.g. [Degano et al. 2012] (automata-based approach) or [Kobayashi et al. 2006] (typechecking-based approach). This investigation is the next natural step of our study.

REFERENCES

Roberto M. Amadio and Silvano Dal-Zilio. 2006. Resource control for synchronous cooperative threads. *Theor. Comput. Sci.* 358, 2-3 (2006), 229–254.

Roberto M. Amadio and Charles Meyssonnier. 2002. On Decidability of the Control Reachability Problem in the Asynchronous pi-Calculus. *Nord. J. Comput.* 9, 1 (2002), 70–101.

Michael Benedikt, Clemens Ley, and Gabriele Puppis. 2010. Minimal memory automata. In *Alberto Mendelzon Workshop on Foundations of Databases.*

Gregory J. Chaitin. 2004. Register allocation and spilling via graph coloring (with retrospective). In *Best of PLDI.* ACM, 66–74.

Pierpaolo Degano, Gian Luigi Ferrari, and Gianluca Mezzetti. 2012. Nominal Automata for Resource Usage Control. In *CIAA (LNCS)*, Vol. 7381. Springer, 125–137.

Aurélien Deharbe and Frédéric Peschanski. 2014. The Omniscient Garbage Collector: a Resource Analysis Framework. In *ACSD.* Springer.

Gian Luigi Ferrari, Stefania Gnesi, Ugo Montanari, Marco Pistore, and Gioia Ristori. 1998. Verifying Mobile Processes in the HAL Environment. In *CAV (LNCS)*, Vol. 1427. Springer, 511–515.

Andrew D. Gordon. 2000. Notes on Nominal Calculi for Security and Mobility. In *FOSAD (LNCS)*, Vol. 2171. Springer, 262–330.

Reiner Hüchting, Rupak Majumdar, and Roland Meyer. 2013. A Theory of Name Boundedness. In *CONCUR (LNCS)*, Vol. 8052. Springer, 182–196.

Edward A. Isper, Jr. 1989. The Infinite State Acceptor and Its Application to AI. *SIGART Bull.* 107 (Jan. 1989), 29–31.

Tommy Jensen and Bjarne Toft. 2011. *Graph coloring problems.* Wiley.

Michael Kaminski and Nissim Francez. 1994. Finite-Memory Automata. *Theor. Comput. Sci.* 134, 2 (1994), 329–363.

Naoki Kobayashi, Kohei Suenaga, and Lucian Wischik. 2006. Resource Usage Analysis for the pi-Calculus. *Logical Methods in Computer Science* 2, 3 (2006).

Ugo Montanari and Marco Pistore. 1995. Checking Bisimilarity for Finitary pi-Calculus. In *CONCUR (LNCS)*, Vol. 962. Springer, 42–56.

Frédéric Peschanski, Hanna Klaudel, and Raymond R. Devillers. 2013. A Petri Net Interpretation of Open Reconfigurable Systems. *Fundam. Inform.* 122, 1-2 (2013), 85–117.

Fernando Rosa-Velardo and David de Frutos-Escrig. 2010. Decidability Problems in Petri Nets with Names and Replication. *Fundam. Inform.* 105, 3 (2010), 291–317.

Davide Sangiorgi and David Walker. 2001. *The π-calculus: a Theory of Mobile Processes.* Cambridge University Press.

Nikos Tzevelekos. 2011. Fresh-register automata. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011.* ACM Press, 295–306.

A. APPENDIX: PROOF DETAILS

A.1. Proof of Propositions 3.5 and 3.6 (binding condition and conflict freedom)

PROOF. (Proposition 3.5) There are two cases to consider. First, if the binding $X \mapsto \nu$ is not created at position k then it is said an *old binding*. By Def. 3.3 if there is no $j,\ j < k$ such that $X \in \mathsf{last}_\rho(v_j)$ vertex v (as assumed) then this binding must have been created at a previous position $i < k$. If the binding is new, then it is a simple fact that $i = k$. □

PROOF. (Proposition 3.6) We proceed by induction on k. First, for $k = 0$ we have no possible conflict and $\Gamma_0 = \emptyset$ so that the property vacuously holds. Now suppose that the property is true for *any* position $j,\ 0 < j \leq k-1$. We proceed by contradiction, showing that it cannot be the case it does not hold anymore at position k. We thus assume $X \sharp_\rho^k Y$ and $\Gamma_k(X) = \Gamma_k(Y)$. There are three cases to consider. First, if $Y \in \mathsf{first}_\rho(v_k) \setminus \mathsf{last}_\rho(v_k)$ then it must consume a symbol ν_Y in the subset $\mathsf{new}(\alpha_k)$ of Def. 3.2. The same definition imposes that this symbol must not be already present in the range of Γ_{k-1}. As such, if $\Gamma_{k-1}(X) = \nu_X$ for some name ν_X then it must be the case that $\nu_Y \neq \nu_X$. Hence, $\Gamma_k(X) \neq \Gamma_k(Y)$, contradicting the hypothesis. If otherwise $X \notin \mathsf{dom}(\Gamma_{k-1})$ then the conflict imposes that $X \in \mathsf{first}_\rho(v_k) \cup \mathsf{fetch}_\rho(v_k)$. If X consumes a name ν_X at position k then we must have $\nu_X \in \mathsf{new}(\alpha_k)$ or $\nu_X \in \mathsf{trans}(\alpha_k)$. In both case a simple cardinality argument imposes that $\nu_X \neq \nu_Y$. The second case is if $Y \in \mathsf{first}_\rho(v_k) \cap \mathsf{last}_\rho(v_k)$, hence it is a transient binding. In this case there is no new binding recorded, so there is nothing left to do. Finally, if the use at position v_k is not a first use, then the binding condition (cf. Prop. 3.5) imposes that there is a position $i < k$ such that $Y \in \mathsf{first}_\rho(v_i) \cup \mathsf{fetch}_\rho(v_i)$. By the hypothesis of induction we have that $\Gamma_i(X) \neq \Gamma_i(Y)$ hence the contradiction. This finishes the proof. □

A.2. Proof of Lemma 4.4 (resource profile recognizers)

The proof requires two auxiliary propositions. The first one relates the structure of the graph G and the one of the $\mathcal{A}_G$.

PROPOSITION A.1. *Let G be a resource graph. There is a one-to-one correspondence between $\Psi(G) \cup \widehat{\Psi}(G)$ and the graph of $\mathcal{A}_G$. Moreover, for each automata path θ_ρ (resp. $\theta_{\widehat{\rho}^k}$) corresponding to a complete path ρ (resp. finite expansion $\widehat{\rho}^k$ of a lasso $\widehat{\rho}$) its starting state is initial and ending state is accepting.*

PROOF. By Def. 4.3 each vertex v_i of a complete path $\rho = \langle v_1, \ldots, v_n \rangle$ (resp. of a lasso $\rho = \langle v_1, \ldots v_{e-1} \mid v_e, \ldots, v_n \rangle$ there is a corresponding state $q_{v_i,\rho}$ (resp. $q_{v_i,\widehat{\rho}}$) in the automaton $\mathcal{A}_G$. Moreover, for each edge (v_i, v_{i+1}) of ρ (resp. $\widehat{\rho}$) there exists a unique label l such that $(q_{v_i,\rho}, l, q_{v_{i+1},\rho})$ (resp. $(q_{v_i,\widehat{\rho}}, l, q_{v_{i+1},\widehat{\rho}})$) in the transition relation of $\mathcal{A}_G$. For a lasso, there is also an exit transition $(q_{v_n,\widehat{\rho}}, \{\}, q_{v_\top})$ and there is no other state or transition defined, hence the one-to-one correspondence. Each finite path $\langle v_1, \ldots, v_n \rangle$ of G corresponding to either a complete path or the k-th expansion of a lasso, by Def. 2.1 we have: $v_1 = v_\bot$ and $v_n = v_\top$. Hence $q_{v_1} = q_{v_\bot}$ – which is initial – and $q_{v_2} = q_{v_\top}$ – which is accepting – by construction. □

The second proposition relates the transition labels of $\mathcal{A}_G$ to the resource events (use, first and last use, etc.).

PROPOSITION A.2. *Let $\mathsf{lbl}^\rho_{v \to v'}$ be a transition label of $\mathcal{A}_G$. Then:*

$$\left[\begin{array}{l} X \in \mathsf{lbl}^\rho_{v \to v'} \ \textit{iff}\ X \in \gamma(v') \\ \nu X \in \mathsf{lbl}^\rho_{v \to v'} \ \textit{iff}\ \mathsf{first}_\rho(v') \cup \begin{cases} \mathsf{fetch}_\rho(v') \ \textit{if}\ v \neq \mathsf{exit}(\rho) \\ \emptyset \ \textit{otherwise} \end{cases} \\ \overline{\nu} X \in \mathsf{lbl}^\rho_{v \to v'} \ \textit{iff}\ X \in \mathsf{last}_\rho(v') \end{array}\right.$$

PROOF. This is direct by Def .4.3. □

Proof. (Lemma 4.4) We have to prove the equivalence $w = \alpha_1 \cdots \alpha_n \in \mathfrak{R}_G \iff w \in \mathbb{L}_G$ with $\mathbb{L}_G$ the language of $\mathcal{A}_G$. Consider the path $\rho = \langle v_1, \ldots, v_n \rangle$ used to recognize word w as in Def. 3.4. By Prop. A.1 there is a corresponding path $\theta_\rho = \langle q_{v_\perp}, q_{v_1,\rho}, \ldots, q_{v_n,\rho} \rangle$ in the graph of $\mathcal{A}_G$. Consider now the run $\sigma = (q_{v_\perp}, \Gamma_0), (q_{v_1,\rho}, \Gamma_1), \ldots, (q_{v_n,\rho}, \Gamma_n)$ such that for each position k, $1 \leq k \leq n$, Γ_k is built from Γ_{k-1} according to Def. 3.3. We must show that σ is an accepting run of w by showing that each symbol α_k of the word is consumed by the configuration $(q_{v_k,\rho}, \Gamma_k)$ and moreover each Γ_k can be built as in Def. 4.2. For the consumption part, we can use the decomposition of α_k as the disjoint subsets $\mathsf{old}(\alpha_k)$, $\mathsf{trans}(\alpha_k)$ and $\mathsf{new}(\alpha_k)$ of Def. 3.2. According to the construction of Def. 4.3 we know that $(q_{v_{k-1},\rho}, \mathsf{lbl}^{\rho}_{v_{k-1} \to v_j}, q_{v_k,\rho})$ is a transition of $\mathcal{A}_G$. Hence, by Prop. A.2 we can deduce the following equalities: $\mathsf{old}(\alpha_k) = \alpha_k^{\mathsf{old}}$, $\mathsf{trans}(\alpha_k) = \alpha_k^{\mathsf{trans}}$ and $\mathsf{new}(\alpha_k) = \alpha_k^{\mathsf{new}}$ (the latter sets as defined in Def. 4.2). Thus, α is fully consumed by the configuration $(q_{v_k,\rho}, \Gamma_k)$. Moreover, also thanks to Prop. A.2 the construction of Γ_k from Γ_{k-1}, as governed by Def. 3.3, is a correct construction according to Def. 4.2. Hence, the configuration $(q_{v_k,\rho}, \Gamma_k)$ is accepting, as is the whole run σ since we left the position k arbitrary. Exactly the same reasoning steps can be followed in the converse way: from Def. 4.2 to Def. 3.4. This concludes the proof. □

A.3. Proof of Theorem 4.5 (resource profiles are quasi-regular)

The objective is to show that the resource profile of a graph G is quasi-regular, i.e. that it can be recognized by a finite memory automaton (FMA) accepting it. The reference paper for FMA is [Kaminski and Francez 1994]. For convenience, we adopt a slightly different notation, consisting in indexing the windows (i.e. registers) of FMA by variable names instead of integers, which can be trivially put in one-to-one correspondence.

Since we cannot encode the fetching (i.e. allocation without consumption) of a fresh symbol in FMA (this would at least require the power of fresh register automata), we have to find a way to remove the fetching phase. This is possible for the subclass of ν-automata recognizing the resource profiles. The idea is to unfold exactly once each lasso of the resource graph when building the recognizer.

Definition A.3. Let $G = \langle R, V, E, \alpha, \gamma, \delta \rangle$ be a resource graph, and $\widehat{\Psi}^1(G)$ the set of unfolded lassos such that:

$$\widehat{\Psi}^1(G) \stackrel{\text{def}}{=} \{\langle v_1, \ldots, v_{e-1}, v_{e_1}, \ldots, v_{n_1} \mid v_{e_2}, \ldots, v_{n_2} \rangle \mid \langle v_1, \ldots, v_{e-1} \mid v_e, \ldots, v_n \rangle \in \widehat{\Psi}(G)\}$$

The ν-automaton induced by G is $\mathcal{A}_G^{\mathsf{fma}} \stackrel{\text{def}}{=} \langle \mathsf{vars}(G), \langle Q, q_{v_\perp}, \Delta, \{q_{v_\top}\} \rangle \rangle$, with:

— $Q = \{q_{v,\rho} \mid \rho \in \Psi(G) \cup \widehat{\Psi}^1(G), v \in \rho \setminus \{v_\perp, v_\top\}\} \cup \{q_{v_\perp}, q_{v_\top}\}$

— $\Delta = \{(q_{v_i,\rho}, \mathsf{lbl}_{v_j}, q_{v_j,\rho}) \mid \rho \in \Psi(G) \cup \widehat{\Psi}^1(G), v_i \to v_j \in \rho\}$
$\cup \{(q_{v_{n_2},\widehat{\rho}}, \mathsf{lbl}_{v_{e_2}}, q_{v_{e_2},\widehat{\rho}}), (q_{v_{n_1},\widehat{\rho}}, \{\}, q_{v_\top}), (q_{v_{n_2},\widehat{\rho}}, \{\}, q_{v_\top}) \mid$
$\widehat{\rho} = \langle v_1, \ldots, v_{e_1}, \ldots, v_{n_1} \mid v_{e_2}, \ldots, v_{n_2} \rangle \in \widehat{\Psi}^1(G)\}$

with: $\mathsf{lbl}_v = \{\nu X \mid X \in \mathsf{first}_\rho(v)\} \cup \{X \mid X \in \gamma(v)\} \cup \{\overline{\nu} X \mid X \in \mathsf{last}_\rho(v)\}$

The first proof step is to show that the language of the new construction is still the expected resource profile.

Lemma A.4. *Let G be a resource graph. Then* $\mathbb{L}(\mathcal{A}_G) = \mathbb{L}(\mathcal{A}_G^{\mathsf{fma}})$.

The difference between $\mathcal{A}_G$ and $\mathcal{A}_G^{\mathsf{fma}}$ is that in the latter, all the allocations are now performed jointly with a use, hence no fetch event is required anymore.

The construction of a FMA from the ν-automaton $\mathcal{A}_G^{\mathsf{fma}}$ is as follws.

Definition A.5. Let $\mathcal{A}_G^{\mathsf{fma}} = \langle \mathcal{X}, \langle Q, q_{\mathsf{init}}, \Delta, F \rangle \rangle$ as in Def. A.3. The induced FMA is $\mathcal{M}_{\mathcal{A}_G^{\mathsf{fma}}} \stackrel{\text{def}}{=} \langle \mathcal{X}, S, s_{\mathsf{init}}, \boldsymbol{u}, \rho, \mu, F', \rangle$, such that:

— $S = \{s_i, s_j \mid s_i \xrightarrow{\sigma} s_j \in \mu\}$
— $\boldsymbol{u} = \{X \mapsto \# \mid X \in \mathcal{X}\} \cup \{\mathsf{Start} \mapsto \mathsf{Start}, \mathsf{End} \mapsto \mathsf{End}, \mathsf{Flush} \mapsto \#\}$
— $\rho(s_{i,k',\theta}) = X \text{ if } \exists(q_i, \{r1, \ldots, r_n\}, q_j) \in \Delta \text{ s.t. } \theta(r_{k'}) = r_k \wedge (r_k = \nu X \vee r_k = \nu\overline{\nu}X)$
$\rho(s'_{i,k',\theta}) = \begin{cases} X \text{ if } \exists(q_i, \{r1, \ldots, r_n\}, q_j) \in \Delta \text{ s.t. } \theta(r_{k'}) = r_k \wedge (r_k = \overline{\nu}X \vee r_k = \nu\overline{\nu}X) \\ \mathsf{Flush} \text{ otherwise} \end{cases}$

— $\mu = \{s_i \xrightarrow{\mathsf{Start}} s_{i,1,\theta} \xRightarrow{\theta(r_1)} \cdots \xRightarrow{\theta(r_n)} s_{i,n+1,\theta} \xrightarrow{\mathsf{End}} s_j \mid$
$(q_i, \lambda, q_j) \in \Delta, \theta \text{ is a permutation function on } \mathsf{dedup}(\lambda) = \{r_1, \ldots, r_n\}\}$
— $F' = \{s_i \mid q_i \in F\}$

with:

$$\left[\begin{array}{l} \mathsf{dedup}(\lambda) = \ (\lambda \setminus (\{X \mid \nu X \in \lambda \vee \overline{\nu}X \in \lambda\} \cup \{\nu X, \overline{\nu}X \mid \{\nu X, \overline{\nu}X\} \subseteq \lambda\})) \\ \qquad\qquad\quad \cup \{\nu\overline{\nu}X \mid \{\nu X, \overline{\nu}X\} \subseteq \lambda\} \\ s_i \xRightarrow{\nu X} s_j = s_i \xrightarrow{X} s'_i \xrightarrow{\mathsf{Flush}} s_j \\ s_i \xRightarrow{X} s_j = s_i \xrightarrow{X} s'_i \xrightarrow{\mathsf{Flush}} s_j \\ s_i \xRightarrow{\overline{\nu}X} s_j = s_i \xrightarrow{X} s'_i \xrightarrow{X} s_j \\ s_i \xRightarrow{\nu\overline{\nu}X} s_j = s_i \xrightarrow{X} s'_i \xrightarrow{X} s_j \end{array}\right.$$

The encoding of transitions is made in two steps. First, each label set $\mathsf{dedup}(\lambda)$, in which we deduplicate bind and usage (resp. unbind and usage) of a same variable, is splitted into single events, by enumeration of all possible permutations of its elements. Then, each obtained transition is doubled. The goal is to correctly handle unbinds of variables, which must consume a symbol and place a fresh symbol into the corresponding window register. According to this translation, ν-words have to be also transformed in order to be accepted by the FMA device. The ν-word encoding consists in placing tags to keep the information about where starts and ends a set of symbols. Then each occurence of ν-symbol is followed by the occurence of a fresh symbol taken in a separate namespace $\overline{\mathcal{V}}$. This new information is mandatory to reset window registers when the corresponding variables must be unbound. This addition must follow each symbol occurence since it is not possible to find the binding and unbinding instant starting only with a ν-word. Formally, encoding of a ν-word is defined as follows:

— $[\![w]\!]_{\mathsf{fma}} = [\![\alpha_1]\!]_{\mathsf{fma}} \cdots [\![\alpha_n]\!]_{\mathsf{fma}}$
— $[\![\alpha_i]\!]_{\mathsf{fma}} = \mathsf{Start}.\bigodot_{1 \leq j \leq m}(\theta(\nu_i).\overline{\nu_{i,j}}).\mathsf{End}$
with $\alpha_i = \{r_1, \ldots, r_m\}$ and θ an arbitrary permutation function on α_i

The last step of the correspondance proof is to show that the language of a ν-automaton obtained from a resource graph and the language of its induced FMA are the same (upto the encoding of the ν-words).

Lemma A.6. *Let $\mathcal{A}_G^{\mathsf{fma}}$ be the ν-automaton induced by a resource graph G, and $\mathcal{M}_{\mathcal{A}_G^{\mathsf{fma}}}$ the finite memory automaton induced by $\mathcal{A}_G^{\mathsf{fma}}$. Then $\mathbb{L}(\mathcal{A}_G^{\mathsf{fma}}) = \{w \mid [\![w]\!]_{\mathsf{fma}} \in \mathbb{L}(\mathcal{M}_{\mathcal{A}_G^{\mathsf{fma}}})\}$.*

Since there exists an encoding of resource graph ν-automata in FMA, we can conclude that resource graph ν-automata language are quasi-regular.

A.4. Proof of Lemma 5.7 (nominal resource bound)

Proof. (of Lemma 5.7) First, the hypothesis $\mathsf{inactive}(G) = \emptyset$ is simply because an unused resource can be trivially removed from the resource graph G without any impact on the resource profile. Failing to remove them gratuitously complexifies the bound calculations. Our proof scheme is to rely on Lemma 4.4, which says that :

$w \in \mathfrak{R}_G$ iff there exists a path $\rho = \langle v_1, \ldots, v_n \rangle$ in G and an associated run $\sigma = (q_{v_\perp}, \Gamma_0), (q_{v_1}, \Gamma_1), \ldots, (q_{v_n}, \Gamma_n)$ such that σ accepts w.

Based on the latter assumption, we define a renaming ζ from $\mathcal{V}$ (infinite alphabet of $\mathcal{A}_G$) to $\mathcal{V}_k$. First, the renaming of the run σ is as follows :

$$\left[\begin{array}{l}\zeta(\sigma) = (q_{v_\perp}, \Gamma_0), (q_{v_1}, \zeta(\Gamma_1)), \ldots, (q_{v_n}, \zeta(\Gamma_n)) \\ \forall i,\ 0 \leq i \leq n,\ \zeta(\Gamma_i) = \{\nu_X \mid X \in \mathsf{dom}(\Gamma_i)\}\end{array}\right.$$

Hence, in all the bindings Γ_i we just bind variables to their corresponding pure name in $\mathcal{V}_k$. To preserve the acceptance relation, we thus have to rename the word w so that the symbols recorded in the Γ_i's now get recorded in the $\zeta(\Gamma_i)$'s.

For this, we use the decomposition of α_i of w as the disjoint sets α_i^{old} (consumption from old bindings), α_i^{old} (transient bindings) and α_i^{new} (new bindings) as in Def. 4.2.

Let $\zeta(\alpha_i^{\mathsf{old}}) = \{\nu_X \mid \Gamma_i(X) = \nu \wedge \nu \in \alpha_i^{\mathsf{old}}\}$. Thus for an old binding of variable X, we have $\zeta(\Gamma_i)(X) = \nu_X$ and thus the renamed binding is still accepted at position i of the renamed run. Next, $\zeta(\alpha_i^{\mathsf{trans}}) = \{\nu_X \mid X \in \mathsf{first}_\rho(v_i) \cap \mathsf{last}_\rho(v_i)\}$, hence there are enough fresh names so that the transient bindings are also consumed in the renamed run. Finally, $\zeta(\alpha_i^{\mathsf{new}}) = \{\nu_X \mid \Gamma_{i+1}(X) = \nu \wedge \nu \in \alpha_i^{\mathsf{new}}\}$ if $i < n$ and $\zeta(\alpha_n^{\mathsf{new}}) = \alpha_n^{\mathsf{new}} = \emptyset$ (since there is no new binding in the final step of the run). Since $\zeta(\Gamma_{i+1})(X) = \nu_X$ the latter subset gets also consumed as required. Thus, $\zeta(\alpha_i)$ is naturally consumed by configuration $(q_{v_{i,\rho}}, \zeta(\Gamma_i))$.

Hence, $\zeta(\sigma)$ accepts $\zeta(w) = \zeta(\alpha_1) \cdots \zeta(\alpha_n)$ thus $\zeta(w) \in \zeta(\mathfrak{R}_G)$ iff $\zeta(\sigma)$ accepts w, which concludes the proof. □

A.5. Proof of Lemma 5.12 and Lemma 5.13 (unifying variables)

The following proposition will play an important role in a later result. It says that non-conflicting resources may not be used at the same time (otherwise a conflict would occur).

Proposition A.7. *Let G be a resource graph, $\rho = \langle v_1, \ldots, v_n\rangle$ a finite path of G and $E \subseteq \mathsf{vars}(G)$ a set of resource variables. If $\forall X, Y \in E$, $\neg(X \sharp_\rho^i Y)$ for a given position i, $1 \leq i \leq n$ then $\mathsf{card}(\gamma(v_i)) \cap E \leq 1$.*

Proof. We proceed by contradiction. If we suppose $\mathsf{card}(\gamma(v_i)) \cap E \geq 2$, then there must be $X, Y \in E$ such that $X \neq Y$ and $\{X, Y\} \subseteq \gamma(v_i)$. By Def. 2.5 we would have $X \sharp_\rho^i Y$ (and also $Y \sharp_\rho^i X$) which contradicts the hypothesis $\neg(X \sharp_\rho^i Y)$. □

Proof. (of Lemma 5.12) Our main hypothesis is: $w \in \mathfrak{R}_G^k$ iff there is a path $\rho = \langle v_1, \ldots, v_n$ and a run $\sigma = (q_{v_\perp}, \Gamma_0), (q_{v_{1,\rho}}, \Gamma_1), \ldots, (q_{v_{n,\rho}}, \Gamma_n)$ of $\mathcal{A}_G^k$ such that σ accepts w. We also consider that the alphabet of pure names is $\mathcal{V}_k = \{\nu_X \mid X \in \mathsf{vars}(G)\}$. Now, for each position i, $1 \leq i \leq n$, we let :

$\mathsf{unify}_{E \triangleleft Z}(\Gamma_i) = \ \{X \mapsto \nu_X \mid X \in \mathsf{dom}(\Gamma_i) \setminus E\} \cup \ \{Z \mapsto \nu_Z \mid X \in \mathsf{dom}(\Gamma_i) \cap E\}$

and $\mathsf{unify}_{E \triangleleft Z}(\sigma) = (q_{v_\perp}, \Gamma_0), (q_{v_{1,\rho}}, \mathsf{unify}_{E \triangleleft Z}(\Gamma_1)), \ldots, (q_{v_{n,\rho}}, \mathsf{unify}_{E \triangleleft Z}(\Gamma_n))$.

Hence, in each binding context Γ_i the variables of E are all replaced by the unique mapping $Z \mapsto \nu_Z$ and the other variables are left unchanged.

We now apply the renaming corresponding to the unification on the symbols of w. We consider the decomposition of each α_i as the disjoint union $\alpha_i^{\mathsf{old}} \cup \alpha_i^{\mathsf{trans}} \cup \alpha_i^{\mathsf{new}}$. And we let :

$$\mathsf{unify}_{E \triangleleft Z}(\alpha_i^{\mathsf{old}}) = \{\nu_X \in \alpha_i^{\mathsf{old}} \mid X \in \mathsf{dom}(\Gamma_i) \setminus E\} \cup \left[\begin{array}{l}\{\nu_Z\} \text{ if } \exists Y \in E \cap (\gamma(v_i) \setminus \mathsf{first}_\rho(v_i)) \\ \emptyset \text{ otherwise}\end{array}\right.$$

The symbols of α_i^{old} that are left unchanged are trivially consumed by configuration $(q_{v_{i,\rho}}, \mathsf{unify}_{E \triangleleft Z}(\Gamma_i))$ since in this case the binding context is left unchanged. If a variable Y of E is used at position i, then it must be bound to a dedicated pure name, which we name ν_Z. By Proposition A.7 only one pure name is enough. And since $\mathsf{unify}_{E \triangleleft Z}(\Gamma_i)(Z) = \nu_Z$ (cf. above) the configuration at step i also accepts ν_z. If otherwise no variable of E is used at step i then there is nothing left to test.

For the transient and new bindings, the proof scheme is exactly the same, based on the following definitions:

$$\left[\begin{array}{rl}\mathsf{unify}_{E\lhd Z}(\alpha_i^{\mathsf{trans}}) = & \{\nu_X \in \alpha_i^{\mathsf{trans}} \mid X \in (\mathsf{first}_\rho(v_i) \cap \mathsf{last}_\rho(v_i)) \setminus E\} \\ & \cup \left[\begin{array}{l}\{\nu_Z\} \text{ if } \exists Y \in E,\ Y \in \mathsf{first}_\rho(v_i) \cap \mathsf{last}_\rho(v_i) \\ \emptyset \text{ otherwise}\end{array}\right. \\ \mathsf{unify}_{E\lhd Z}(\alpha_i^{\mathsf{new}}) = & \{\nu_X \in \alpha_i^{\mathsf{new}} \mid X \in (\mathsf{first}_\rho(v_i) \setminus \mathsf{last}_\rho(v_i)) \setminus E\} \\ & \cup \left[\begin{array}{l}\{\nu_Z\} \text{ if } \exists Y \in E,\ Y \in \mathsf{first}_\rho(v_i) \setminus \mathsf{last}_\rho(v_i) \\ \emptyset \text{ otherwise}\end{array}\right.\end{array}\right.$$

We deduce that $\mathsf{unify}_{E\lhd Z}(\alpha_i) = \mathsf{unify}_{E\lhd Z}(\alpha_i^{\mathsf{old}}) \cup \mathsf{unify}_{E\lhd Z}(\alpha_i^{\mathsf{trans}}) \cup \mathsf{unify}_{E\lhd Z}(\alpha_i^{\mathsf{new}})$ is accepted by configuration $(q_{v_{i,\rho}}, \mathsf{unify}_{E\lhd Z}(\Gamma_i))$. Thus w is accepted by run $\mathsf{unify}_{E\lhd Z}(\sigma)$ as expected. This run is trivially a possible run of $\mathsf{unify}_{E\lhd Z}(\mathcal{A}_G^k)$, which concludes the proof. □

PROOF. (of Lemma 5.13) The proof is almost the same as the one for Lemma 5.12 except that we consider a complete partition of $\mathsf{vars}(G)$. It is a simple fact that the unification of Π can be done in an arbitrary order since the renamings operate on a distinct domain (a subset E of variables and their unifier Z) and distinct range (the name ν_Z corresponding to the unifier Z). Moreover, by iterating Lemma 5.12 we have that $\mathcal{R}^{k-\sum_{E\in\Pi}\mathsf{card}(E)+\mathsf{card}(\Pi)}$ is recognized by $\mathsf{unify}_{\Pi\lhd\mathcal{Z}}(\mathcal{A}_G^k)$. And since $\sum_{E\in\Pi}\mathsf{card}(E) = \mathsf{card}(\mathsf{vars}(G))$ we obtained the expected bound $\mathsf{card}(\Pi)$. □

A.6. Proof of Lemma 5.14 (resource index)

We first show that unifying conflicting variables would contradict the process of recognizing resource profiles.

LEMMA A.8. *Let $w = \alpha_1 \cdots \alpha_n$ be a ν-word accepted by a run $\sigma = (q_{v_\perp}, \Gamma_0), (q_{v_{1,\rho}}, \Gamma_1), \ldots, (q_{v_{n,\rho}}, \Gamma_n)$ of a ν-automaton $\mathcal{A}_G^k$ (with k variables) of a resource graph G. Then* $\mathsf{unify}_{\{X,Y\}\lhd Z}(w)$ *is accepted by* $\mathsf{unify}_{\{X,Y\}\lhd Z}(\sigma)$ *iff* $\forall i,\ 1 \le i \le n,\ \neg(X \sharp_\rho^i Y) \wedge \neg(Y \sharp_\rho^i X)$.

PROOF. We proceed by contradiction. Suppose there is a position j of ρ such that $X \sharp_\rho^j Y$ (the symmetric case is similar). This means that $\exists i, k,\ 1 \le i \le j \le k \le n$ such that $X \in \gamma(v_i) \cap \gamma(v_k)$ and $Y \in \gamma(v_j)$. There must be a pure name ν_X such that $\forall l,\ i \le l \le k,\ \Gamma_l(X) = \nu_X$ (by the binding condition, cf. Prop. 3.5). And at position j the variable Y must be either bound to a name ν_Y (old or new binding), or the latter name must be consumed immediately (transient binding). In both cases, ν_Y must be distinct from ν_X otherwise the conflict freedom property (cf. Prop.3.6) would fail. However, $\mathsf{unify}_{\{X,Y\}\lhd Z}(\sigma)$ rename both ν_X and ν_Y by a single name ν_Z, thus trivially the run cannot accept w anymore. □

PROOF. We start with the resource profile $\mathfrak{R}_G^{\mathsf{card}(\Pi)}$ recognized by the automaton $\mathcal{A}$ with variables $\mathcal{X} = \{X_1, \ldots, X_{\mathsf{card}(\Pi)}\}$. Suppose the resource profile $\mathfrak{R}_G^k$ such that $k = \mathsf{card}(\Pi) - 1$ is recognizable.

Then two independent subsets of $\mathsf{vars}(G)$ must be unified, which means there are two variables $X, Y \in \mathcal{X}$ such that $X \sharp Y$ (since they are dependent wrt. the partition of $\sharp$) provided $\mathfrak{R}_G^k$ is recognized by $\mathsf{unify}_{\{X,Y\}\lhd Z}(\mathcal{A})$ for some $Z \notin \mathcal{X}$. But this would contradict Lemma A.8, hence there is no resource bound lower than $\mathsf{card}(\Pi)$. □