The Omniscient Garbage Collector: a Resource Analysis Framework (extended abstract)

Aurélien Deharbe and Frédéric Peschanski
University Pierre et Marie Curie - Paris 6 - LIP6
Email: firstname.lastname@lip6.fr

***Abstract*—The notion of resource plays a central role in concurrent systems. In its purest form a resource is simply a unique identity one can create, use and ultimately destruct. In this paper we propose a simple yet effective characterization of resource usages and develop for it a complete analysis framework. We address qualitative issues such as the classification of resources and whether two systems exhibit similar patterns of resource usages – namely equivalent resource profiles. From the quantitative point of view, we develop the omniscient garbage collector (OGC), which decides precisely when a resource can be reclaimed or reused. This allows to bound precisely the number of resources consumed by a given system. To illustrate the approach, we study experimentally the resource consumption of pi-calculus processes using a prototype analysis tool. We propose two different resource abstractions for pi-processes: one based on the labelled transitions for open systems, and another one for closed systems. The latter notably provides a refined view of behaviors, less opaque than reductions. Beyond this experiment, the proposed framework is quite generic and can apply to many different formalisms and situations.**

I. Introduction

The notion of resource plays a central role in the study of computational systems, and even more critically in the realm of concurrency. If we abstract from its internal structure, a resource becomes a *pure name* [1], i.e. an object with a globally unique and testable identity. This is the specialty of *nominal calculi* in general, and the π-calculus [2] 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 work begins with a very simple graphical characterization of resource usages. To analyze these so-called *resource graphs*, we develop a recursive computation principle upon which most of our algorithms are built, together with an inductive principle to reason about their properties *and* a modular characterization of their complexity. Beyond the algorithmic contingency, our principal means of abstraction is a formal language – namely the *resource profile* – that can be roughly seen as a "regular-language up-to α-conversion" over resource usages. Testing resource profile equivalence reveals, we think, much about the internal behavior of the compared systems. This comes at the price of PSPACE-hardness in terms of computational complexity.

Resource consumption represents the quantitative facet of our study. 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. In practice, the objective is to design an allocator for resources which is able to reuse as much locations as possible, while maintaining a strong invariant of *conflict-freedom*. Computing the resource index requires an *omniscient garbage collector*, a NP-complete problem tightly connected to the *perfect coloring* of so-called *conflict graphs*.

Beyond the complexity results, we aim at the development of practical tools for the analysis of resource usage in concurrent systems. Using an early prototype, we propose a couple of experiments of resource consumption in the realm of the π-calculus. To illustrate the versatility 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. The latter notably provides a refined view of behaviors, less opaque than reductions. For this we introduce "slice-π", a rather standard π-calculus extended with an alternative restriction operator that allows to "slice" the behaviors so that a flexible notion of environment is reintroduced. 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 the "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 that $100\,000$ states! Moreover, it seems rather improbable that highly irregular conflict graphs can be easily constructed except *on purpose*. Complementarily, less tight but still low resource bounds can be computed very efficiently.

The outline of the paper is as follows. In Section II we establish the resource model that forms the theoretical basis of our analysis framework. The latter is developed in Section III in three steps: (1) a procedure of lattice completion resulting in generic recursive computation and inductive reasoning principles, (2) the omniscient garbage collector for the computation of resource bounds and index, and (3) an algorithmic approach to the resource equivalence problem. Our experimental study with the π-calculus is described in Section IV. A panorama of related work is given in Section V.

In this extended abstract the justification of mathematical statements is most of the time omitted or discussed informally. The complete proofs are detailed in a dedicated appendix.

II. RESOURCE MODEL

The purpose of a *resource model* is to abstract the manipulation of dynamic resources from concrete system behaviors, so that we can reason about them in isolation.

Definition 1 (resource graph): Let $\mathfrak{R}$ be a countably infinite set of *resource variables* ranging over X, Y, Z, ... A resource graph G is a connected directed graph $\langle R, V, E, \alpha, \gamma, \delta \rangle$ with :

- $R \subseteq \mathfrak{R}$ a finite[1] 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 $v_\bot \in V$ s.t. $\forall v \in V, (v, v_\bot) \notin E$ and a unique tail $v_\top$ s.t. $\forall v \in V, (v_\top, v) \notin E$.
- α(resp. γ, δ) : $V \to 2^R$ an allocation (resp. usage, delete) function(s) such that: $\alpha(v_\bot) \cup \alpha(v_\top) \cup \gamma(v_\bot) \cup \gamma(v_\top) \cup \delta(v_\bot) \cup \delta(v_\top) = \emptyset$.

□

An example of a resource graph is depicted on the left part of Fig.1. It has eight resource variables $A \dots H$ (the emphasized vertices and dashed edges will be explained later) and is sufficiently non-trivial so that it exhibits most of the "corner cases" of the model.

The properties of resource graphs we are interested in can be characterized as properties about finite paths, falling in two categories: *complete paths* and *lassos*.

Definition 2 (complete path and lasso): A complete path of a resource graph G with edge set E is of the form $\rho = v_\bot \to \cdots \to v_\top$ with pairwise distinct vertices. A lasso is a finite path $\widehat{\rho} = v_1 \to \cdots \to v_n$ such that $v_1 = v_\bot$ and $\forall i, j, 1 \leq i < j \leq n$, $v_i \neq v_j$ 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. □

There are only two complete paths in the graph of Fig.1: $v_\bot \to v_1 \to v_k \to \ldots \to v_6 \to v_8 \to v_{11} \to v_\top$ with $v_k = v_2$ or $v_k = v_3$. There are six lassos, an example being $v_\bot \to v_1 \to v_3 \to v_4 \to \ldots \to v_6 \to v_8$ with entry v_4.

We impose only minimal constraints on the nature and usage of resources in resource graphs, although some usage patterns must be enforced.

Definition 3: 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 $\rho = v_1 \to \cdots \to v_n$ of G there is at most one vertex v in ρ such that $X \in \delta(v)$. Moreover, if $\exists j,\ 1 \leq j \leq n$ s.t. $X \in \gamma(v_j)$ then $\exists i,\ 1 \leq i \leq j$ s.t. $X \in \alpha(v_i)$ and:

- if ρ is a complete path then:

$$\exists k,\ j \leq k \leq n \text{ s.t. } X \in \delta(v_k)$$

- 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.

The reasons behind most constraints are obvious, i.e. any 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 $v_\bot \to v_1 \to v_3 \to v_4 \to \ldots \to v_6 \to v_8$ with entry v_4 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 *and* it is used within the cycle, then X must "survive" the cycle and is thus said a *static resource*. For the lasso $v_\bot \to v_1 \to v_3 \to \ldots \to v_{11}$ with entry v_3 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.

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

Definition 4 (resource classification): Let G a resource graph with vertex set V.

$$\left[\begin{array}{l} \mathsf{inactive}(G) \stackrel{\text{def}}{=} \{X \mid \forall v \in V,\ X \notin \gamma(v)\} \\ \mathsf{static}(G) \stackrel{\text{def}}{=} \{X \mid X \text{ is static in a lasso of } G\} \\ \mathsf{dynamic}(G) \stackrel{\text{def}}{=} \mathsf{vars}(G) \setminus (\mathsf{static}(G) \cup \mathsf{inactive}(G)) \end{array}\right.$$

In our example the sets are $\mathsf{static}(G) = \{E, G\}$, $\mathsf{inactive}(G) = \{F\}$ and $\mathsf{dynamic}(G) = \{A, B, C, D, H\}$.

Our principal means to abstract from the relatively low-level resource graphs is to characterize resource usage as a *formal language*.

Definition 5 (resource profile): Let G a resource graph. Its resource profile is the language based on alphabet $2^{\mathsf{vars}(G)}$ defined as follows:

$$\begin{array}{rl} \mathfrak{R}_G \stackrel{\text{def}}{=} & \left\{ \begin{array}{l} \odot_{1 \leq i \leq n} \mathsf{use}(v_i) \cdot [\odot_{e \leq j \leq n} \mathsf{use}(v_j)]^* \\ \mid v_1 \to \cdots \to v_e \to \cdots \to v_n \text{ a lasso of } G \end{array} \right\} \\ & \cup \left\{ \begin{array}{l} \odot_{1 \leq i \leq n} \mathsf{use}(v_i) \\ \mid v_1 \to \cdots \to v_n \text{ a complete path of } G \end{array} \right\} \\ \text{with } \mathsf{use}(v) \stackrel{\text{def}}{=} & \{\widehat{X} \mid X \in \gamma(v) \cap \mathsf{static}(G)\} \\ & \cup (\gamma(v) \cap \mathsf{dynamic}(G)) \end{array}$$

For example, for the lasso $v_\bot \to v_1 \to v_3 \to v_4 \to v_5 \to v_6 \to v_8$ with entry v_4, the generated word is:

$$\{\}.\{H\}.\{\widehat{E}\}.\{A\}.\{\widehat{G}\}.\{A\}.\{B\}.\left[\{A\}.\{\widehat{G}\}.\{A\}.\{B\}\right]^*$$

It is important to understand the principal intuitive characteristics of the abstraction. First, we abstract away from the

[1] The consideration of infinite resource graphs is an interesting generalization of our theoretical framework. However, the whole algorithmic approach would not apply anymore. On the contrary, the class of infinite systems with finite resource graphs is particularly inspiring.

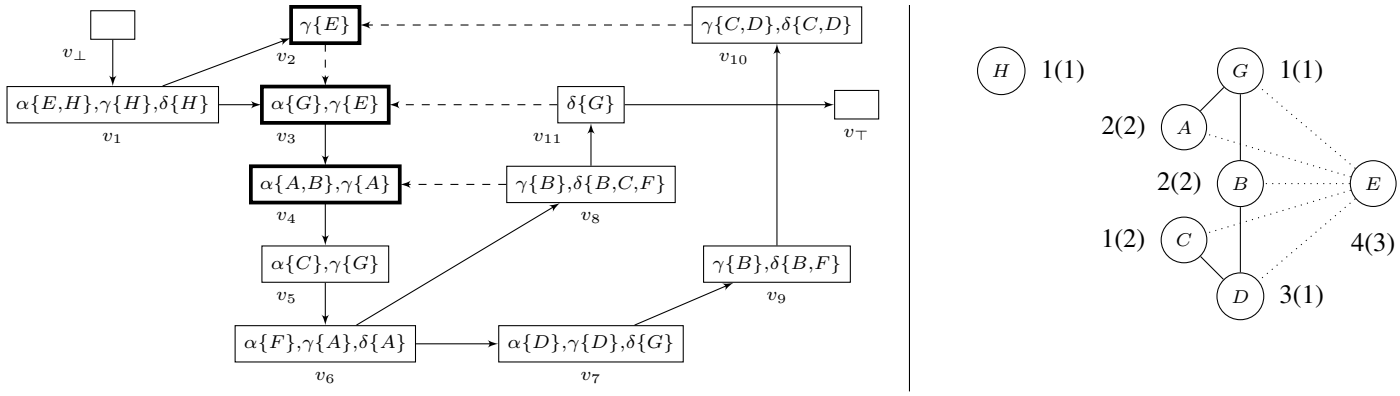


Fig. 1. Example of a resource graph (left) and the associated conflict graph with first fit (resp. perfect) coloring (right).

allocation and deletion events. The reason is that they fundamentally only play a role for the static/dynamic distinction. This explains why the static resources are explicitly recorded otherwise the information would be lost. Moreover, we only exploit language union, concatenation and Kleene star, i.e. regular language constructors. This is to abstract away from the branching structure of the graph, which play only an indirect role wrt. resource usage. Note however that the language $\mathfrak{R}_G$ is *not* itself regular since the names of resource variables are purely symbolic and subject to a form of α-conversion. This is in fact a *crucial* aspect of the model to enforce the *purity* of resource names.

The resource profile also record a fundamental information for our further analyzes: the *resource conflicts*. A conflict happens when a resource Y is used between (at least) two uses of another resource X. This means that X cannot be *garbage collected* at the time Y is used. This information is easily characterized in the resource graphs.

Definition 6 (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 ρ, denoted by $X \sharp_\rho Y$, if:

- $\rho = v_\bot \to^+ v_i \to^* v_j \to^* v_k \to \cdots$ and either $X \in \gamma(v_i) \cap \gamma(v_k)$ and $Y \in \gamma(v_j)$, or $Y \in \gamma(v_i) \cap \gamma(v_k)$ and $X \in \gamma(v_j)$.
- Moreover, if ρ is a lasso $v_1 \to \cdots \to v_e \to \cdots \to v_n$ and X is static in $\widehat{\rho}$, a conflict between X and Y occurs if there exists i such that $e \leq i \leq n$ and $Y \in \gamma(v_i)$.

If there is a path ρ such that $X \sharp_\rho Y$ 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 the right part of Fig. 1 (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 $v_7 \to v_9 \to v_{10}$ 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).

An obvious but fundamental property is that the resource conflicts are preserved by the resource profile abstraction.

Proposition 1: A resource conflict $X \sharp Y$ occurs in G iff there is a word of the form $w_1.a.w_2.b.w_3.c.w_4$ in $\mathfrak{R}_G$ with $X \in a,\ Y \in b,\ X \in c$ or $Y \in a,\ X \in b,\ Y \in c$.

Most importantly, resource variables can be unified if they are not in conflict, which leads to the following notion.

Definition 7 (conflict-free equivalence): An equivalence relation $\mathcal{E}$ over a set of resource variables is said conflict-free if $(X, Y) \in \mathcal{E} \implies \neg(X \sharp Y)$. □

From this we can finally define a proper notion of equivalence for resource profiles.

Definition 8 (resource profile equivalence): Let G and H be resource graphs. Their profiles $\mathfrak{R}_G$ and $\mathfrak{R}_H$ are equivalent, denoted by $\mathfrak{R}_G \approx \mathfrak{R}_H$, iff either $\mathfrak{R}_G = \mathfrak{R}_H$ or $\mathsf{vars}(G) \cap \mathsf{vars}(H) = \emptyset$ and there exists a conflict-free equivalence $\mathcal{E}$ over $\mathsf{vars}(G) \cup \mathsf{vars}(H)$ such that $\mathfrak{R}_G / \mathcal{E} = \mathfrak{R}_H / \mathcal{E}$. □

The quotient $\mathfrak{R}_G / \mathcal{E}$ corresponds to the renaming of each use X by its equivalence class in $\mathcal{E}$, i.e. $[X]_\mathcal{E}$. Hence, resource profiles are equivalent up-to conflict-free renamings. The condition $\mathsf{vars}(G) \cap \mathsf{vars}(H) = \emptyset$ may appear as a strong constraint, but it is necessary to emphasize the *arbitrary* role played by the names of resource variables. If we allowed the same variable X to occur on both sides of the equivalence, then we would artificially equate the corresponding resources with no other criterion than their name ! Thankfully, we can always perform a kind of α-conversion to satisfy the disjointedness requirement.

Proposition 2: Let σ be a bijective renaming of the resource variables of a given graph G such that $\mathsf{dom}(\sigma) \cap \mathsf{ran}(\sigma) = \emptyset$. Then we have $\mathfrak{R}_G \approx \mathfrak{R}_{G\sigma}$ with $G\sigma \stackrel{\text{def}}{=} \langle R\sigma, V, E, \alpha\sigma, \gamma\sigma, \delta\sigma \rangle$ the image of G by σ.

Beyond resource profile equivalence, we are interested in quantitative aspects of resource usages. These rely on the notion of *resource bounds* and *resource index*.

Definition 9 (resource bound and index): A graph G has resource bound $k \in \mathbb{N}$ iff there exists a total renaming σ of $\mathsf{vars}(G)$ onto $\mathsf{ran}(\sigma)$ with $\mathsf{dom}(\sigma) \cap \mathsf{ran}(\sigma) = \emptyset$ and $\mathsf{card}(\mathsf{ran}(\sigma)) = k$, and such that $\mathfrak{R}_G \approx \mathfrak{R}_{G\sigma}$. The resource index of G is the smallest $\chi \in \mathbb{N}$ such that G has bound χ.

Suppose for example a graph with 3 variables and resource bound $k = 2$. This means at least 2 of the variables can

be unified without contradicting the conflict graph. This is witnessed by the following proposition.

Proposition 3: Let k a resource bound for a graph G witnessed by a renaming σ_k. Then $\mathcal{E}_{\sigma_k} \stackrel{\text{def}}{=} \{X = \sigma_k(X) \mid X \in \mathsf{vars}(G)\}$ is a conflict-free equivalence denoting k distinct equivalence classes.

The resource index – the lowest resource bound – is a very accurate quantitative witness of resource consumption, although in practice finding less tight but still decent upper bounds is also pertinent.

III. Resource analysis

Based on the simple yet operative resource model defined in the previous Section, we now begin the elaboration of our practical resource analysis framework.

A. Lattice completion and graph recursor

Our starting point is a slight modification of the resource graphs, providing us: (1) a generic computation principle, (2) a corresponding reasoning principle for establishing the correctness of the algorithms, and (3) a modular characterization of their worst-case complexity. For the sake of concision, we only detail a few of the many analyzes that can be described this way, some other examples are discussed more informally.

Definition 10 (lattice completion): Let $G = \langle R, V, E, \alpha, \gamma, \delta\rangle$ be a resource graph. Define $\widehat{\mathcal{L}} \stackrel{\text{def}}{=} \{\widehat{\rho}_{e,n} \mid \widehat{\rho}_{e,n} = v_1 \to \cdots \to v_e \to \cdots \to v_n$ is a lasso of G with entry $v_e\}$. The lattice-completed resource graph is $\widetilde{G} \stackrel{\text{def}}{=} \langle R, V, \widetilde{E}, \Omega, \alpha, \gamma, \delta\rangle$ with:

$$\left[\begin{array}{l} \widetilde{E} \stackrel{\text{def}}{=} \left(E \setminus \{v_n \to v_e \mid \widehat{\rho}_{e,n} \in \widehat{\mathcal{L}}\}\right) \cup \{v_n \to v_\top \mid \widehat{\rho}_{e,n} \in \widehat{\mathcal{L}}\} \\ \Omega \stackrel{\text{def}}{=} \{v_n \to v_e \mid \widehat{\rho}_{e,n} \in \widehat{\mathcal{L}}\} \end{array}\right.$$

In the example of Fig. 1 the highlighted vertices v_2, v_3 and v_4 are the lasso entries of the graph. In the completion procedure, the edges (with dashed arrows) that point to these entries are *cut* from the graph and redirected towards $v_\top$ (this redirection is not depicted). The Ω component contains the edges that generate the cycles in the graph, which we of course need to remember for deciding certain properties (such as whether a given vertex is an entry of some lasso).

The completion procedure corresponds, algorithmically speaking, to a graph decomposition into nested strongly connected components (SCCs). The algorithm has quadratic worst-case complexity and produces an acyclic graph with interesting ordering properties.

Proposition 4: Let $\widetilde{G}$ a completed resource graph with vertices V and edges E. Then $(V, \widehat{E})$ is a complete lattice with $\widehat{E}$ the reflexive and transitive closure of E.

From now on we will assume that resource graphs are properly completed, and will denote by G its completion $\widetilde{G}$.

Definition 11 (graph recursor): Let G be a resource graph with vertex set V. The graph recursor for $f : \mathbb{Y} \times V \to \mathbb{X}$ (vertex function) and $g : 2^{\mathbb{X}} \to \mathbb{Y}$ (edge function) at $v \in V$ is:

$$\mathsf{rec}_G^v(f,g) \stackrel{\text{def}}{=} f(g\left(\{\mathsf{rec}_G^{v'}(f,g)\} \mid v' \to v \text{ in } G\}\right), v)$$

We abbreviate $\mathsf{rec}_G^{v_\top}(f,g)$ as $\mathsf{rec}_G(f,g)$.

The recursor can be implemented by a simple topological sort traversal of the completed graph, starting from $v_\bot$ and ending at the desired termination vertex v (trivially, termination occurs at worst at $v_\top$, the top-element of the ordering). Each edge is visited at most once so the traversal has linear complexity. Consequently, the recursor has a naturally modular worst-case complexity.

Proposition 5: Let G a resource graph with vertex set V and edge set E. The worst-case complexity of a graph recursor $\mathsf{rec}_G^v(f,g)$ is $O\left(\mathsf{card}(V) \times \mathfrak{C}_f + \mathsf{card}(E) \times \mathfrak{C}_g\right)$ where $\mathfrak{C}_f$ (resp. $\mathfrak{C}_g$) is the worst-case complexity of f (resp. g).

Because these depend on the concrete data-structures employed, we do not detail the operands $\mathfrak{C}_f$ and $\mathfrak{C}_g$ of the complexity calculations. However, all the recursive computations described below are efficient polytime algorithms. Beyond computation, most of our correctness proofs rely on a simple yet effective inductive principle.

Lemma 1 (graph inductor): Let $\mathcal{P} : V \to \{\mathsf{true}, \mathsf{false}\}$ be a predicate and G a resource graph with vertex set V. Then $\forall v \in V,\ \mathcal{P}(v)$ if and only if:

$$\mathcal{P}(v_\bot) \text{ and } \forall v' \to v \text{ in } G,\ \mathcal{P}(v') \implies \mathcal{P}(v).$$

A global graph property $\mathcal{P}_G$ then corresponds to proving the local property $\mathcal{P}_G(v_\top)$ using the inductor principle.

We now illustrate the recursor/inductor framework by computing the sets of *live* variables (allocated, used and not freed) and *active* variables (allocated and used, possibly freed) at a given vertex.

Definition 12 (live and active variables): Let G a resource graph. We define the live and active variables at vertex v by the following recursors:

$$\left[\begin{array}{l} \mathsf{live}_v(G) \stackrel{\text{def}}{=} \mathsf{rec}_G^v\left(\lambda x, v.(x \cup \gamma(v)) \setminus \delta(v), \cup\right) \\ \mathsf{active}_v(G) \stackrel{\text{def}}{=} \mathsf{rec}_G^v\left(\lambda x, v.x \cup \gamma(v), \cup\right) \end{array}\right.$$

To illustrate the inductor principle, we will use it to demonstrate the following property.

Proposition 6: $\mathcal{P}(v) \stackrel{\text{def}}{=} X \in \mathsf{active}_v(G)$ iff $\exists u,\ u \to^* v$ s.t. $X \in \gamma(v)$.

Proof: The recursor we consider is:

$$\mathsf{active}_v(G) = \mathsf{rec}_G^v(f, \bigcup) \text{ with } f(x,v) \stackrel{\text{def}}{=} x \cup \gamma(v).$$

- (base case) We must prove $\mathcal{P}(v_\bot)$. The left-hand side of the *iff* is *false* since we have $\mathsf{active}_v(G) = f(\bigcup \emptyset, v_\bot) = \gamma(v_\bot) = \emptyset$. This is because $v_\bot$ has no predecessor and $\gamma(v_\bot) = \emptyset$ by definition. The right-hand side is also trivially *false* for the same reason. Hence $\mathcal{P}(v_\bot)$ is *true*.
- (inductive case) The hypothesis of induction is that $\mathcal{P}(v')$ holds for any predecessor v' of v. We have to consider two sub-cases. First, we suppose that $X \in \mathsf{active}_w(G)$ for some predecessor w of v. Now, if we write $\mathsf{active}_v(G) = f(\bigcup W, v)$ then clearly $\mathsf{active}_w(G) \subseteq W$ and thus $\mathsf{active}_w(G) \subseteq$

$\mathsf{active}_v(G) = (\bigcup W) \cup \gamma(v)$, hence $X \in \mathsf{active}_v(G)$. Moreover, by hypothesis of induction $\exists u,\ u \to^* w$ s.t. $X \in \gamma(u)$ and from $u \to^* w \to v$ we conclude $\mathcal{P}(v)$ is true. The second sub-case is if $X \notin \mathsf{active}_{v'}(G)$ for any (direct) predecessor v' of v. By hypothesis of induction this means $X \notin \gamma(w)$ for any (direct or indirect) predecessor w of v. In this case it is a trivial fact that $X \in \mathsf{active}_v(G)$ *iff* $X \in \gamma(v)$ and thus $\mathcal{P}(v)$ is also true in this case.

Applying Lemma 1 we conclude that the property $\mathcal{P}(v)$ is true for any vertex v of G. ∎

We can generalize such local property by considering them at the tail $v_\top$. For instance, the resources that are still live in $v_\top$ have been used and not freed, hence because the graph has correct usage they are exactly the static resources. Similarly we can compute the set of inactive variables.

Proposition 7:

$$\left[\begin{array}{l} \mathsf{static}(G) = \mathsf{live}_{v_\top}(G) \\ \mathsf{inactive}(G) = \mathsf{vars}(G) \setminus \mathsf{active}_{v_\top}(G) \end{array}\right.$$

A relatively simple recursor can also be defined for the algorithmic construction of an automaton recognizing the language of a given resource profile.

Lemma 2: $\mathfrak{R}_G$ is recognized by the finite automaton:

$\mathcal{A}_G \stackrel{\text{def}}{=} \langle Q, q_{v_\bot}, \delta, F\rangle$ s.t. $(Q, \delta, F) = \mathsf{rec}_G^{v_\top}(f, \bigcup)$ with:

$$\mathsf{f}(x, v) \stackrel{\text{def}}{=} \begin{cases} (\{q_{v_\bot}\}, \{\}, \{q_{v_\bot}\}) \text{ if } x = \{\} \\ (Q \cup \{q_v\}, \delta', \{q\}) \text{ if } x = \{(Q, \delta, F)\}, \\ \text{with } \delta' \stackrel{\text{def}}{=} \delta \cup \{q \xrightarrow{\mathsf{use}(v)} q_v \mid q \in F\} \cup \\ \{q_v \xrightarrow{\mathsf{use}(v')} q_{v'} \mid v \to v' \in \Omega_G\} \end{cases}$$

For our running example the resulting automaton (equivalent but slightly simplified so that it fits the page) is depicted on Fig.2. Note that the only accepting state is the one generated for $v_\top$. Moreover if the construction yields an automaton that has roughly the shape of the initial resource graph, in practice many factorizations can be performed on-the-fly.

For the computation of the resource conflicts, we have two complementary algorithmic approaches: (1) define a dedicated recursor for the task (as implemented in our toolset, cf. Section IV), or (2) detect the conflicts directly on the automaton $\mathcal{A}_G$. The latter is possible since as explained previously the language $\mathfrak{R}_G$ encodes the conflict graph of G. For example the conflict $B \sharp D$ can be found by the sequence of states q_6, q_7, q_8, q_9. For the conflict $A \sharp G$ we can follow e.g. the loop $q_5, q_6, q_2, q_3, q_4, q_5$.

B. The Omniscient Garbage Collector

The α-convertibility of the variables in resource profiles is the principal obstacle to the development of efficient analysis algorithms. Our approach of the problem is inspired by the *memory allocation* metaphor. We want to assign unambiguously unique *locations* to the resource variables. Of course, we should use as few locations as possible, which underlies an important notion of *garbage collection*.

Definition 13 (allocator): Let $\mathbb{L}$ be an ordered set of locations $[\ell_1, \ldots, \ell_n]$ such that $n \leq \mathsf{card}(\mathsf{vars}(G))$. An allocator μ is a mapping from variables to locations. It is a safe allocator iff, whenever $X \sharp Y$ then $\mu(X) \neq \mu(Y)$. An omniscient allocator is a safe allocator that minimizes n.

Conflict-freedom is the only and fundamental requirement of the allocation problem. Indeed, we can use less resources (i.e. reuse resource locations) as long as this does not creates a contradiction in the conflict graph. This intuition can be characterized formally.

Lemma 3: Let μ be a safe allocator for a resource graph G. Then μ is a renaming such that $\mathfrak{R}_G \approx \mathfrak{R}_{G\mu}$ and $\mathsf{card}(\mathsf{ran}(\mu))$ is a resource bound for G.

The effective computation of resource bounds naturally relates to the algorithmics of *graph coloring* [3].

Proposition 8: Let $\sharp$ be the conflicts of a given resource graph G, and let $\mu_\sharp$ be a proper coloring of $\sharp$. Then $\mu_\sharp$ is a safe allocator for G.

The simplest of all (imperfect) coloring algorithms: *first-fit coloring* provides us with an interesting resource bound that can be computed quite efficiently.

Proposition 9: Let G be a resource graph with conflict graph $\sharp$. Then G has resource bound $d_G + 1$ where $d_G \stackrel{\text{def}}{=} \mathsf{max}_{X \in \mathsf{vars}(G)}\{Y \mid X \sharp Y\}$. Moreover, d_G can be computed in linear time in the size of $\sharp$.

In the right part of Fig. 1, the numbered labels of the nodes correspond to colorings of the conflict graph. 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 resource bound found is 4 and is less than $d_G + 1 = 6$.

For the resource index we need an omniscient allocator, which essentially corresponds to the *perfect coloring* of the conflict graph.

Proposition 10: Let $\widetilde{\mu}_\sharp$ be the safe allocator corresponding to the perfect coloring of the conflicts $\sharp$ of a resource graph G. Then $\widetilde{\mu}_\sharp$ is an omniscient allocator for G. Moreover, $\mathsf{card}(\mathsf{ran}(\widetilde{\mu}_\sharp))$ is the resource index of G.

For our example the resource index χ is 3, it is also the chromatic number of the conflict graph. The associated perfect coloring is shown in Fig. 1 by 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.

This leads to our first prominent complexity result.

Theorem 1: Computing the resource index χ of a resource graph is NP-complete.

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

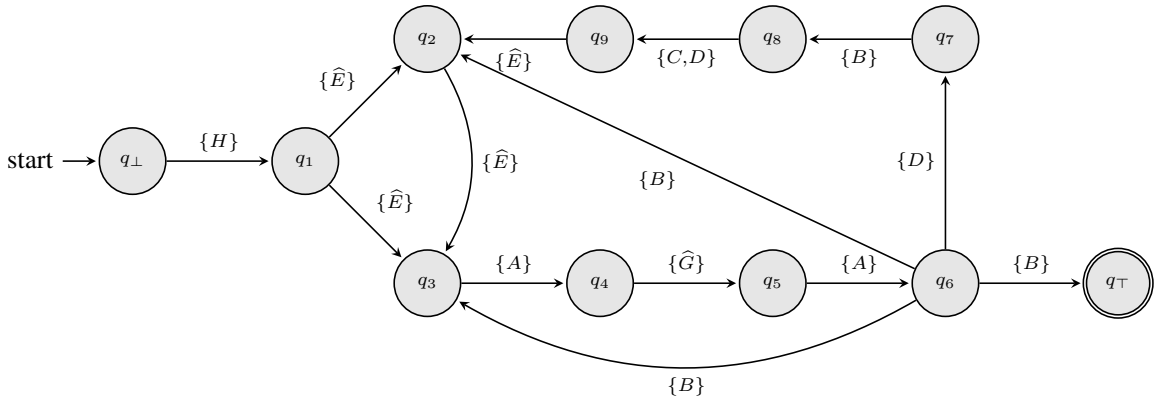


Fig. 2. An automaton recognizing the profile of the resource graph of Fig.1.

the latter. Furthermore, interesting properties of *separability* can often be exploited.

Proposition 11: Let $\sharp$ be a conflict graph. If $\sharp = \biguplus_i \sharp_i$ then $\widetilde{\mu}_\sharp = \bigcup_i \widetilde{\mu}_{\sharp_i}$. Moreover if for some resource X we have $\forall Y \in \sharp,\ X = Y \vee X \sharp Y$ then $\chi_\sharp = 1 + \chi_{\sharp \setminus \{X\}}$.

These are basic properties of graph coloring. First, disconnected components of the conflict graphs can be colored separately. In our example conflict graph this is the case of the sub-graph with node H and the one consisting of all the other resources. The second property is also useful in practice. In our example, the vertex E is connected to all the other vertices of the conflict graph (except H). The subgraph without E has chromatic number 2 and the second property tells us that the chromatic number of the complete graph is 3.

C. Algorithmic resource profile equivalence

We approach the resource profile equivalence problem in three successive algorithmic steps. In the first step, we compute omniscient allocators for the two resource graphs to compare.

Proposition 12: Let $\widetilde{\mu}_G$ and $\widetilde{\mu}_H$ be omniscient allocators for respective resource graphs G and H. Then $\mathfrak{R}_G \approx \mathfrak{R}_H$ iff $\mathfrak{R}_{G\widetilde{\mu}_G} \approx \mathfrak{R}_{H\widetilde{\mu}_H}$.

We now face a new intermediate problem, which is to find a bijection between the (allocated) resource variables of G and H that can be lifted to a conflict-free equivalence. This exactly corresponds to finding an isomorphism between the conflict graphs of G and H.

Lemma 4: Let $\widetilde{\mu}_G$ and $\widetilde{\mu}_H$ be omniscient allocators for respective resource graphs G and H. Moreover, let γ be an isomorphism between the conflicts of the profiles $\mathfrak{R}_{G\widetilde{\mu}_G}$ and $\mathfrak{R}_{H\widetilde{\mu}_H}$, and $\mathcal{E}_\gamma \stackrel{\text{def}}{=} \{X = Y \mid X \in \mathsf{dom}(\gamma) \wedge Y = \gamma(X)\}$.

Then $\mathfrak{R}_G \approx \mathfrak{R}_H$ iff $\mathfrak{R}_{G\widetilde{\mu}_G}/\mathcal{E}_\gamma = \mathfrak{R}_{H\widetilde{\mu}_H}/\mathcal{E}_\gamma$.

We remind the reader that graph isomorphism defines the GI complexity class (in NP). Now, since $\mathfrak{R}_G$ and $\mathfrak{R}_H$ are composed only of regular constructors and α-conversion does not apply anymore, we finally reduced the resource equivalence problem to the well-studied problem of equating regular languages, which is notoriously a PSPACE-complete problem. We might question whether a better algorithm could be found for the resource profile equivalence problem. The answer is *no* and the reason is that ultimately, regular language equivalence can be reduced polynomially to the resource equivalence problem (considering the reflexive case of $\approx$).

Hence our second fundamental complexity result.

Theorem 2: Resource profile equivalence is PSPACE-hard.

Definition		$D(\tilde{x}) \stackrel{\text{def}}{=} P$	
Process P, Q	::=	0	(inert)
	|	$\mathsf{new}(x)\ P$	(observable)
	|	$\mathsf{local}(x)\ P$	(inobservable)
	|	$\alpha.P$	(prefix)
	|	$P \mid Q$	(parallel)
	|	$D[\tilde{a}]$	(call)
Action α	::=	τ	(silent)
	|	$\overline{a}b$	(output)
	|	$a(x)$	(input)

Fig. 3. The syntax of the π-calculus (with slices).

IV. 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. By lack of space, the presentation remains mostly informal.

A. A π-calculus refresher

The syntax of the variant of the π-calculus we cover in the experiment[2] is given in Fig. 3. The semantics of the language for most constructors can be found in many sources (e.g. [2]). Informally, the process 0 has no transition. The scope of a name x can be restricted by either $\mathsf{new}(x)$ or $\mathsf{local}(x)$ and for the labelled transition semantics the two constructs are assumed synonymous (this will be different in reduction semantics). A prefixed process $\alpha.P$ denotes a transition with a

[2]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.

label corresponding to the action α and continuing as process P. There are four kinds of labels depending on the action α:

- a label τ is generated by a silent action τ.
- a label ab is generated by an input action $a(x)$ 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, or 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.

B. 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:

- any transition $P \xrightarrow{\overline{a}\nu b} Q$ creates a resource X_b and is interpreted as:
  $v_P \to \boxed{\alpha\{X_b\},\gamma\{X_b\},\delta\{\}} \to v_Q$
- any 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[3]. 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. This simple form of garbage collection is implemented by all the analysis tools for the pi-calculus that we are aware of.

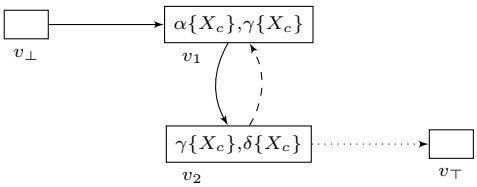
Let us consider as a first example the following process:

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

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:

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

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:



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.

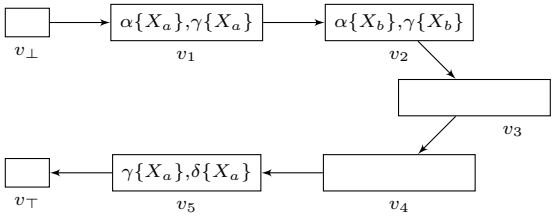
A minimal conflict can be generated by e.g.:

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

A slightly complexified variant of this process is as follows:

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

The resource graph corresponding to $C[d(y).\overline{d}y.0]$ is[4]:



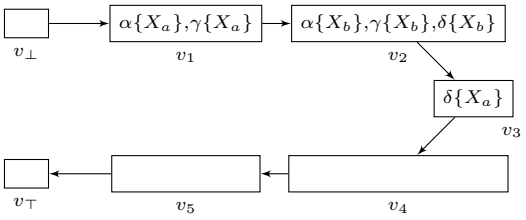
[3]We do not provide in this paper a formal proof that "*bisimilarity* implies *resource profile equivalence*" but this is rather trivial since resource profiles, under the labelled abstraction, encode partial trace sets of behaviors.

[4]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	Res. graph	Res. 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

TABLE I. EXPERIMENTAL RESULTS FOR THE RESOURCE ABSTRACTION ON LABELLED TRANSITIONS.

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:



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.

This abstraction has been implemented in a prototype tool and we analyzed several examples from the *HAL environment* [4]. 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 (but 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. Table I 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.

The heap example models a set of interacting memory cells. Each cell is a process with an input and an output channel and stores a single datum. The cells are composed in parallel, which induces a large LTS resulting from the interleaving of many internal synchronizations. Since most branching is created by pure interleaving and τ transitions, the resource graphs we obtain are rather small in comparison. The resource index for heap_n is $n-1$ is all cases, which is an invariant in the general case. This in fact counts the number of competing cells in the system, with a single non-conflicting case (the two "entry" cells are not competing with each other).

The buffer example is a variant of the heap but with less internal synchronizations and in proportion more exchanges with the environment. Hence, we observe more resources to take into account. In consequence, the size of the resource graph is not reduced in large proportions anymore. However, the resource index is still very small, with a suggested invariant of a resource index n for buffer_n, all cells being in competition. These two examples exhibit a rather low amount of conflicts if compared to the size of their state-space. This means that perfect coloring, despite its high computational complexity, is not a difficult problem in these particular cases. We firmly believe that this is the case in many realistic examples, although this remains to be confirmed experimentally.

Another interesting example is the GSM and its variants that model a simplified form of the *handover protocol* for *gsm* mobile networks. These examples show rather complex name exchanges but are quite small in term of control. What we find particularly interesting is that the different variants all result in the same resource graph (up-to resource variable renamings) and of course the same resource index. This confirms, rather surprisingly (we expected at best the same resource profiles), that in terms of resource usage the three examples are basically the same.

C. 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 "normal" 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 is such that either $a \in \Lambda$ if lacks the observed tag, and otherwise we have $\underline{a} \in \Lambda$. Of course, it might not be the case that $\{a, \underline{a}\} \subseteq \Lambda$. The inobservable names are simply absent from Λ.

For each reduction $P \to Q$ we have either:

- an *open reduction* of the form:
  $\Lambda \vdash P \xrightarrow{\langle b \rangle} (\Lambda \backslash \{b\}) \cup \{\underline{b}\} \vdash Q$ 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.

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

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

Fig. 4. Reductions of slice-π processes with a observable (left) or inobservable (right).

The complete semantics of this "slice-π" variant is provided in appendix. The main interest of this refinement is that (not unlike *barbs* [2]) it opens up some part of the behavior of the processes, and still maintains a one-to-one correspondence with the standard reductions.

In terms of resource graphs, the interpretation is now quite similar to the labelled abstraction:

- any reduction $\Lambda \vdash P \xrightarrow{\langle b \rangle} \Lambda' \vdash Q$ creates a resource X_b and is interpreted as:
  $v_{\Lambda \vdash P} \longrightarrow \boxed{\alpha\{X_b\}, \gamma\{X_b\}, \delta\{\}} \longrightarrow v_{\Lambda' \vdash Q}$
- any 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:
  $v_{\Lambda \vdash P} \longrightarrow \boxed{\alpha\{\}, \gamma\{\}, \delta\{\}} \longrightarrow v_{\Lambda \vdash Q}$

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. 4 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. This means that 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 distinct resource indices: 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.

Model	# Procs.	Reds.	Res. graph	Res. index
philos_2	7	133	20	1
philos_3	10	2992	136	1
philos_4	13	98245	4148	2

TABLE II. EXPERIMENTAL RESULTS FOR THE RESOURCE ABSTRACTION ON REDUCTIONS.

Based on this abstraction, we designed a simple example inspired by the infamous *dining philosophers*. 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 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 Table II. 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 who actually competing for food remain below the number of fork pairs, ensuring the correctness of the protocol. Although simpler analyzes are of course possible for this 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.

By lack of time, we could not experiment the second abstraction on more significant examples. And at the time of

writing, the tool – in a very early stage of development – still lacks support for computing resource equivalence problems.

V. 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* [1] in general, and in particular the π-calculus [2] 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 [5] 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. [6], [7] and [8]. 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 IV. The abstraction of *active restrictions* proposed in [8] 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:

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

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 wether 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* [2] 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 [8] is the prominent role played by the notion of *garbage collection* something already observed in e.g. the *history-dependent-automata* [4] or in the π-graphs [9]. 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* [10], 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 [11]. 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 e.g. *variable automata* [12]. One major difference is that we map the resource variables to a finite (and hopefully least) number of locations. Hence, our automata are "quasi-regular" only introducing a (non-trivial) notion of α-conversion. This is enough as long as we address pure nominal questions, however if we look "into" resources then a connection with the above-mentioned frameworks seems highly probable. This is especially the case if we address resource control issues such as in e.g. [13] (automata-based approach) or [14] (typechecking-based approach).

References

[1] A. D. Gordon, "Notes on nominal calculi for security and mobility," in *FOSAD*, ser. LNCS, vol. 2171. Springer, 2000, pp. 262–330.

[2] D. Sangiori and D. Walker, *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[3] T. Jensen and B. Toft, *Graph coloring problems*. Wiley, 2011.

[4] G. L. Ferrari, S. Gnesi, U. Montanari, M. Pistore, and G. Ristori, "Verifying mobile processes in the hal environment," in *CAV*, ser. LNCS, vol. 1427. Springer, 1998, pp. 511–515.

[5] R. M. Amadio and S. Dal-Zilio, "Resource control for synchronous cooperative threads," *Theor. Comput. Sci.*, vol. 358, no. 2-3, pp. 229–254, 2006.

[6] R. M. Amadio and C. Meyssonnier, "On decidability of the control reachability problem in the asynchronous pi-calculus," *Nord. J. Comput.*, vol. 9, no. 1, pp. 70–101, 2002.

[7] F. Rosa-Velardo and D. de Frutos-Escrig, "Decidability problems in petri nets with names and replication," *Fundam. Inform.*, vol. 105, no. 3, pp. 291–317, 2010.

[8] R. Hüchting, R. Majumdar, and R. Meyer, "A theory of name boundedness," in *CONCUR*, ser. LNCS, vol. 8052. Springer, 2013, pp. 182–196.

[9] F. Peschanski, H. Klaudel, and R. R. Devillers, "A petri net interpretation of open reconfigurable systems," *Fundam. Inform.*, vol. 122, no. 1-2, pp. 85–117, 2013.

[10] U. Montanari and M. Pistore, "Checking bisimilarity for finitary pi-calculus," in *CONCUR*, ser. LNCS, vol. 962. Springer, 1995, pp. 42–56.

[11] G. J. Chaitin, "Register allocation and spilling via graph coloring (with retrospective)," in *Best of PLDI*. ACM, 2004, pp. 66–74.

[12] O. Grumberg, O. Kupferman, and S. Sheinvald, "Variable automata over infinite alphabets," in *LATA*, ser. LNCS, vol. 6031. Springer, 2010, pp. 561–572.

[13] P. Degano, G. L. Ferrari, and G. Mezzetti, "Nominal automata for resource usage control," in *CIAA*, ser. LNCS, vol. 7381. Springer, 2012, pp. 125–137.

[14] N. Kobayashi, K. Suenaga, and L. Wischik, "Resource usage analysis for the pi-calculus," *Logical Methods in Computer Science*, vol. 2, no. 3, 2006.

APPENDIX 1: PROOF DETAILS

A. Resource model

Proof: (of Proposition 1)

if case: Suppose $X\sharp Y$ in a given resource graph G. According to Definition 6 there is a path $\rho = v_1 \to \cdots \to v_n$ such that $X\sharp_\rho Y$ with 3 possible causes. First, if ρ is a complete path then there exist $i, j, k,\ 1 < i \leq j \leq k < n$ such that $X \in \gamma(v_i),\ Y \in \gamma(v_j),\ X \in \gamma(v_k)$ or $Y \in \gamma(v_i),\ X \in \gamma(v_j),\ Y \in \gamma(v_k)$. We suppose the former, the latter being symmetric. Moreover we suppose X dynamic and Y static, the other cases being similar. Let $w_\rho = \bigodot_{i=1}^{n} \mathsf{use}(v_i)$ by the word generated in $\mathfrak{R}_G$ by ρ. Now define $a \stackrel{\text{def}}{=} \mathsf{use}(v_i)$, $b \stackrel{\text{def}}{=} \mathsf{use}(v_j)$ and $c \stackrel{\text{def}}{=} \mathsf{use}(v_j)$. We thus have $X \in a$, $\widehat{Y} \in b$ and $X \in c$. Moreover, by Definition 5 $w_\rho = w_1.a.w_2.b.w_3.c.w_4$ for some (possibly empty) words $w_1, \ldots, w_4$, hence the property.

If ρ is a lasso with entry $v_e,\ 1 < e \leq n$ then we have two sub-cases to consider. The first sub-case is identical to the previous one, the second sub-case corresponding to a cyclic conflict (second case of Definition 6). If we suppose that X is static and Y dynamic (the other cases being symmetric), the we have some $i, j,\ 1 < i < e \leq j \leq n$ such that $\widehat{X} \in \mathsf{use}(v_i)$ (by Definition 3 supposing G has correct usage) and $Y \in \mathsf{use}(v_j)$. Hence in $\mathfrak{R}_G$ the cycle is unfolded once thus if we let $a = c = \mathsf{use}(v_i)$ and $b = \mathsf{use}(v_j)$ then w_ρ is of the expected form $w_\rho = w_1.a.w_2.b.w_3.c.w_4$.

only-if case: We suppose in a resource profile $\mathfrak{R}_G$ there is a word $w = w_1.a.w_2.b.w_3.c.w_4$ such that $\widehat{X} \in a$, $Y \in b$ and $\widehat{X} \in c$ (the other cases are similar). By Definition 5 we know that w is generated by either a finite path $\rho = v_1 \to \cdots \to v_n$ that is either a complete path or a lasso of G. If it is a complete path then w is of the form $w_1.\{\widehat{X}, a'\}.w_2.\{Y, b\}.w_3.\{\widehat{X}, c'\}.w_4$ and in the path ρ we have i, j, k such that $X \in \gamma(v_i),\ Y \in \gamma(v_j),\ X \in \gamma(v_k)$ and thus $X\sharp_\rho Y$. If ρ is a lasso there we have either the same situation as previously (a succession of X, Y, X in the path) or w corresponds to the unfolding of a word of the form $w'.(w'')^*$ and $w'' = w''_1.a.w''_2.b.w''_3$ and thus $c = a$ and in consequence $X\sharp_\rho Y$. ■

Proof: (of Proposition 2)

We trivially have $\mathsf{vars}(G\sigma) = R\sigma = \mathsf{vars}(G)\sigma$. From the hypothesis $\mathsf{dom}(\sigma) \cap \mathsf{ran}(\sigma) = \emptyset$, we thus obtain $\mathsf{vars}(G) \cap \mathsf{vars}(G\sigma) = \emptyset$. Now let $\mathcal{E}_\sigma \stackrel{\text{def}}{=} \{X = \sigma(X) \mid X \in \mathsf{vars}(G)\}$ the equivalence relation associated σ. Now suppose $(Z, Z') \in \mathcal{E}_\sigma$ such that $Z\sharp Z'$. Since σ is a bijection we have $Z \in \mathsf{vars}(G)$ and $Z' = \sigma(Z_1) \in \mathsf{vars}(G\sigma)$ or the converse (since $\mathcal{E}_\sigma$ is symmetric). Moreover since the resource graph is not modified beyond variable renaming, $Z\sharp Z'$ iff $Z\sharp\sigma^{-1}Z' \iff Z\sharp Z$. This contradicts Definition 6 which imposes the conflict relation to be irreflexive, hence $\neg(Z\sharp Z')$ as required.

■

Proof: (of Proposition 3)

Following definition 9 it is easy to obtain $\mathfrak{R}_G/\mathcal{E}_{\sigma_k} = \mathfrak{R}_{G\sigma}/\mathcal{E}_{\sigma_k}$. Hence since $\mathsf{vars}(G) \cap \mathsf{vars}(G\sigma) = \emptyset$ by definition 8 it must be the case that $\mathcal{E}_{\sigma_k}$ is a conflict-free equivalence as required. Moreover since $\mathsf{card}(\mathsf{ran}(\sigma_k)) = k$ and σ is total and onto, it follows naturally that $\pi_k \stackrel{\text{def}}{=} \{C \mid \forall X, Y \in C,\ [X]_{\mathcal{E}_{\sigma_k}} = [Y]_{\mathcal{E}_{\sigma_k}}\}$ has cardinality k. ■

B. Lattice completion and graph recursor

Proof: (of Proposition 4)

A standard result of lattice theory is that if a finite ordered set S has a greatest (resp. a least) element and every pair of elements has a meet (resp. a join), then S is a lattice. Moreover, every finite lattice is a complete lattice.

Let $\widetilde{G} \stackrel{\text{def}}{=} \langle R, V, \widetilde{E}, \Omega, \alpha, \gamma, \delta\rangle$ be a complete resource graph. We interpret $\widetilde{G}$ has a partial ordering considering V as the carrier set and the reflexive and transitive closure of E as the ordering relation. This is indeed an proper partial ordering since the completion procedures removes and cycles, and thus the graph structure is that of a directed acyclic graph (DAG). Moreover, by definition we have $v_\top$ (resp. $v_\bot$) is a greatest (resp. least) element of the ordering.

Now, let v and v' be two vertices of $\widetilde{G}$. Suppose (v, v') has no meet, which means that (1) either they have no lower bound or (2) that the set $\mathcal{M} \stackrel{\text{def}}{=} \{M \subseteq V \mid m \in W,\ m \to^* v \wedge m \to^* v'\}$ has no maximal element. Case (1) would contradict the existence of $v_\bot$. For case (2) the contradiction originates from the connectivity of the resource graph. Hence the meet of (v, v') *does* exist. In conclusion, $\widetilde{G}$ is a complete lattice. ■

Proof: (of Lemma 1)

The existence and constructibility of the least fixed point is justified by Knaster-Tarski's fixed point theorem for complete lattices (standard proof as found in many textbooks). ■

Proof: (of Proposition 7)

According to definition 3, a resource is static means it is used (and thus allocated) at some point, and not deleted in at least one path of the resource graph. This suggests the following local property.

$$\mathcal{P}(v) \stackrel{\text{def}}{=} X \in \mathsf{live}_v(G) \text{ implies } \exists u,\ u \to^* v \text{ s.t. } X \in \gamma(u) \text{ and } \forall w,\ w \to^+ v \text{ then } X \notin \delta(w)$$

For the base case, we have $\mathsf{live}_{v_\top}(G) = \emptyset$ hence $\mathcal{P}(v_\bot)$ is vacuously true.

For the inductive case we suppose $\mathcal{P}(v')$ to hold for any vertex v' such that there is an edge $v' \to v$. We now proceed by contraction, supposing that $\mathcal{P}(v)$ does not hold, which means $X \in \mathsf{live}_v(G)$ but the right-hand side of the implication is false. Now the two main sub-cases is whether $X \in \mathsf{live}_{v'}(G)$ or not, for v' a direct predecessor of v. If X is live at v' then the only way contradict the right-hand side of the implication is to have $X \in \mathsf{delta}(v)$ but by definition of $\mathsf{live}_v(G)$ this may not be the case. Now if $\mathsf{live}_{v'}(G)$ is false, then it must be the case that $X \in \mathsf{gamma}(v) \setminus \mathsf{delta}(v)$ for $X \in \mathsf{live}_v(G)$ to be true, which automatically makes the right-hand side true because the graph is assumed to have correct usage.

By Lemma 1 we thus conclude that $\mathcal{P}(v)$ holds for any vertex v. Now, since the resource graph G is assumed with correct usage, a variable X that is used but not deleted must be static, and according to properties $\mathcal{P}$ and $\mathcal{Q}$ we have $X \in \mathsf{live}_G^{v_\top}$ hence the result. ■

Proof: (of Lemma 2).

For a given vertex $v \in V$, $\mathsf{rec}_G^v(f, \bigcup)$ defines an automaton $\mathcal{A}_G^v$ and we have $\mathcal{A}_G = \mathcal{A}_G^{v_\top}$ by definition.

There is a trivial isomorphism between the graph G and its corresponding automaton $\mathcal{A}_G$. Here the graph G is interpreted as the original resource graph before the completion procedure. Each vertex has a corresponding state q_v and the resource usage at v is reflected by a transition from each state q of a predecessor v' of v with destination q_v. Note also that at the end of step v the only final state is q_v. Of course, before reaching the vertex $v_\top$ the automaton only reflects a part of the graph that we denote by G^v such that $G^{v_\top} = G$. Because G^v lacks a top element, we simply add $v \to v_\top$ except if $v_\top$ is already a vertex of G^v (which means that $v = v_\top$ anyway). At v we must also add the deletions for the active variables appearing in complete paths. This modifications required for the graph to be with correct usage is not difficult and will not be detailed.

By induction we can prove that $\mathfrak{R}_{G^v}$ is recognized by the finite state automaton $\mathcal{A}_{G^v}$. For the base case at $v_\bot$, $\mathsf{rec}_G^{v_\bot}(f, \bigcup)$ yields an automat that recognizes only the empty language, as required. For the inductive case we suppose that for each predecessor of v, the automaton $\mathcal{A}_G^{v'}$ recognizes the language $\mathfrak{R}_{G^{v'}}$. For each complete path in one of the $\mathfrak{R}_{G^{v'}}$'s there are two possibilies. Either v closes the path so that it becomes the entry of a lasso, and in this case $\mathfrak{R}_{G^v}$ recognizes, from an "old" word $w_1.w_2$ a "new" word of the form $w_1.w_2.\mathsf{use}(v)(w_2.\mathsf{use}(v))^*$. In $\mathcal{A}_G^v$ a new transition $q_v \xrightarrow{\mathsf{use}(v)} q_{v'}$ is inserted, which exactly corresponds to the "switch" from the old to the new word. Otherwise, if only a complete path is uncovered, then the recognized word w becomes $w.\mathsf{use}(v)$ and a corresponding transition is built in $\mathcal{A}_G^v$. Since the words on the one side and the states/transitions on the other side are both assembled with set union, the automaton $\mathcal{A}_G^v$ recognizes the whole $\mathfrak{R}_{R_G^v}$.

Following the inductive principle of Lemma 1 the proof is completed. ■

C. Omniscient garbage collector

Proof: (of Lemma 3)

That E_μ is a conflict-free equivalence is by the safety condition in Definition 13. We now define $E' \stackrel{\text{def}}{=} \{X = [X]_{E_\mu} \mid X \in \mathsf{vars}(G)\}$, which is also trivially a conflict-free equivalence. It is a simple fact that $\mathfrak{R}_G/E' = \left(\mathfrak{R}_G/E_\mu\right)/E'$ and thus $\mathfrak{R}_G \approx \mathfrak{R}_G/E_\mu$ as required.

Moreover, μ is by definition total on $\mathsf{vars}(G)$ and is onto $\mathsf{ran}(\mu)$. It is also the case that $\mathsf{vars}(G) \cap \mathsf{ran}(\mu) = \emptyset$, hence following definition 9 we can conclude $\mathsf{card}(\mathsf{ran}(\mu))$ is a resource bound for G. ■

Proof: (of Proposition 8)

The proper coloring of a graph is such that no two adjacent vertices possess the same color. Hence, if $\mu_\sharp$ is a proper coloring of a conflict graph over $\mathsf{vars}(G)$ then $X \sharp Y$ (i.e. X and Y are adjacent in the conflict graph) implies $\mu_\sharp(X) \neq \mu_\sharp(Y)$ for any $X, Y \in \mathsf{vars}(G)$, hence $\mu_\sharp$ is a safe allocator for G. ■

Proof: (of Proposition 9)

This is a standard result of graph coloring. ■

Proof: (of Proposition 10)

Let $\widetilde{\mu}$ be a perfect coloring of the conflicts $\sharp$ of a resource graph G. By Proposition 8 we know that it is already a safe allocator and according to Lemma 3 G has resource bound $k = \mathsf{card}(\mathsf{ran}(\widetilde{\mu}))$. The existence of a proper coloring – hence a safe allocator – μ such that $\mathsf{card}(\mathsf{ran}(\mu)) < k$ would contradict the fact that $\widetilde{\mu}$ is perfect, i.e. that it minimizes the number of colors used to decorate the conflict vertices.

The number k – the chromatic number of $\sharp$ – is thus a minimal resource bound, i.e. the *resource index* of G. ■

Proof: (of Theorem 1)

For the NP-hardness part of the proof, the objective is to start with an arbitrary graph $\sharp$ that must be perfectly colored. It is simpler to consider the nodes of $\sharp$ to be a set of resources variables in $\mathcal{R}$ and to consider a natural ordering $\leq$ on $\mathcal{R}$. To reduce the problem in terms of the resource index computation, we must construct a resource graph G such that its conflict graph is $\sharp$ itself. The construction does not require much creativity so we only sketch it. First we construct a sequence C containing the edges of $\sharp$ following a lexicographic ordering on variable pairs, and we remove all the symmetric pairs. This ordering is required so that no interference is created among the conflicts. Now, we take each conflict $X \sharp Y$ in order and we simply create a sub-path $u \to v \to w$ such that $X \in \gamma(u) \cap \gamma(w)$ and $Y \in \gamma(v)$. At the first occurrence of a variable X we add a resource creation event, and we similarly add a deletion even at the last occurrence. Finally we add the root and tail vertices. The length of the resulting path is a function linear in $\mathsf{card}(C)$ and thus also in the number of edges in the graph $\sharp$. It is a simple fact to then demonstrate that the path exhibit the conflicts in $\sharp$. Now finding an omniscient allocator solves the initial perfect coloring problem.

For the completeness part we know that the conflict graph can be implemented by a polynomial graph recursor. Then perfecct graph coloring is exactly what remains to be performed. ■

Proof: (of Proposition 11

These are basic properties of graph coloring algorithms. ■

D. Resource profile equivalence

Proof: (of Proposition 12)

By Lemma 3 we know that $\mathfrak{R}_G \approx \mathfrak{R}_{G\widetilde{\mu}_G}$ and $\mathfrak{R}_H \approx \mathfrak{R}_{G\widetilde{\mu}_H}$.

The result thus follows trivially by symmetry and transitivity arguments. ■

Proof: (of Lemma 4)

if case: The hypothesis is $\mathfrak{R}_G \approx \mathfrak{R}_H$ and since $\widetilde{\mu}_G$ and $\widetilde{\mu}_H$ are safe allocators, we have $\mathfrak{R}_{G\widetilde{\mu}_G} \approx \mathfrak{R}_{H\widetilde{\mu}_H}$ by Lemma 3.

This implies the existence of a conflict-free equivalence $\mathcal{E}$ such that $\mathfrak{R}_{G\widetilde{\mu}_G}/\mathcal{E} = \mathfrak{R}_{H\widetilde{\mu}_H}/\mathcal{E}$ by definition of $\approx$.

Now, let $\gamma_{\mathcal{E}} \stackrel{\text{def}}{=} \{X \mapsto Y \mid X \in \mathsf{vars}(G\widetilde{\mu}_G),\ Y \in \mathsf{vars}(H\widetilde{\mu}_H),\ (X,Y) \in \mathcal{E}\}$.

We remark that $\mathsf{card}(\mathsf{dom}(\gamma_{\mathcal{E}})) = \mathsf{card}(\mathsf{ran}(\gamma_{\mathcal{E}}))$ by a simple consideration of language equivalence (the alphabets must be the same on both sides of language equality) and moreover $\gamma_{\mathcal{E}} \circ \gamma_{\mathcal{E}}^{-1} = \mathsf{id}$ since $\mathcal{E}$ is symmetric. In consequence $\gamma_{\mathcal{E}}$ is an isomorphism, which is enough to conclude the proof.

only-if case: We suppose $\mathfrak{R}_{G\widetilde{\mu}_G}/\mathcal{E}_\gamma = \mathfrak{R}_{H\widetilde{\mu}_H}/\mathcal{E}\gamma$ for a given isomorphism γ. According to the definition of $\approx$ this means $\mathfrak{R}_{G\widetilde{\mu}_G} \approx \mathfrak{R}_{H\widetilde{\mu}_H}$ and by Proposition 12 we have $\mathfrak{R}_G \approx \mathfrak{R}_H$ as required.

■

Proof: (of Theorem 2)

We adopt a constructive proof by considering the basic language of regular expressions. A regular expression e is defined inductively on an alphabet Σ as either:

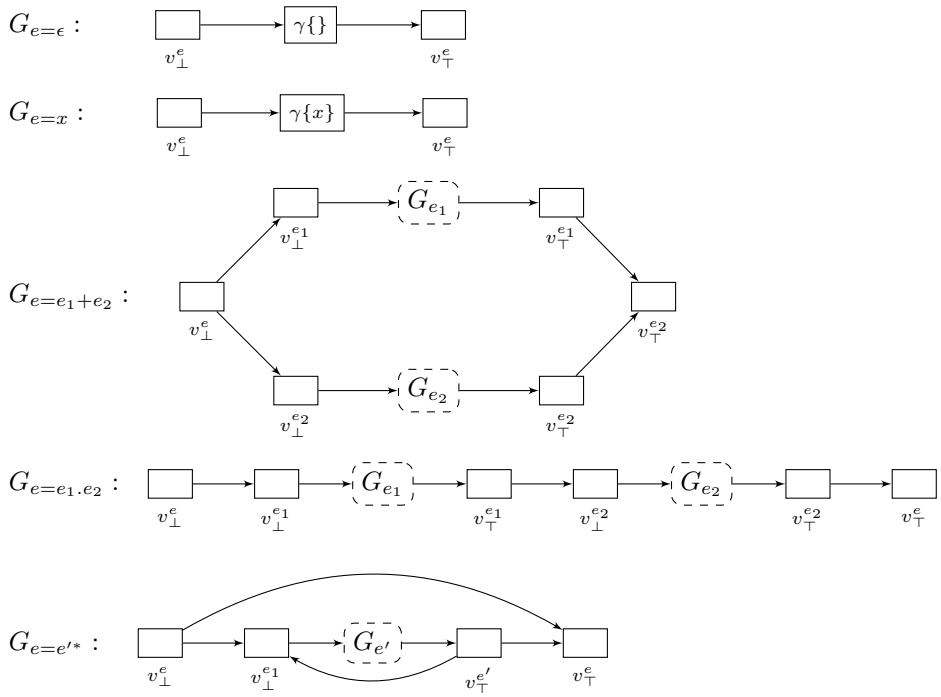
- empty language: $e = \emptyset$
- empty symbol: $e = \epsilon$
- singleton: $e = x$ for any symbol $x \in \Sigma$
- union: $e = e_1 + e_2$ with e_1, e_2 regular expressions
- concatenation: $e = e_1.e_2$ with e_1, e_2 regular expressions
- Kleene star: $e = e'^*$ with e' a regular expression.

Regular expressions notoriously characterize the regular languages. We will denote by L_e the regular language specified by a regular expression e.

For our hardness proof, we will mimic the construction of non-deterministic finite automata (NFA) from regular expressions. However in our case, for each kind of regular expression we will produce a resource graph with correct resource usage. The construction is relatively simple and tedious to formalize so we adopt a semi-formal presentation.

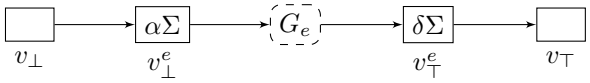
We first build the structure of resource graph G_e corresponding to a regular expression e is a tuple $\langle R_e = 2^{\Sigma \cup \{\epsilon\}}, V_e, E_e, \alpha_e, \gamma_e, \delta_e \rangle$ constructed recursively as follows:

$G_{e=\emptyset}$:



The only non-trivial case is the last one, because in resource graph we must generate lassos, i.e. paths generated from non-reflexive transitive concatenation closures. Hence use the following encoding: $e^* = \epsilon + e^+$. Now, the graph G_e obtained finally only lacks proper allocations and deletions (i.e. the α_e and δ_e components). Since allocations must be unique, the only possibility is to allocate all the resources (i.e. symbols of the initial regular expression) at a "dummy" bottom node. Then we have to delete the resources just "before" the top node at the end. This gives the longest possible lifespan for the variables.

The final construction is thus something like the following:



By a simple structural induction, it is easy (albeit cumbersome) to demonstrate that the profile $\mathfrak{R}_{G_e}$ is indeed exactly the language L_e specified by the regular expression e. The proof simply corresponds to the inductive construction of an isomorphism between the construction of a NFA recognizing e on the one side, and the automaton $\mathcal{A}_{G_e}$ constructed using the recursor described in the paper. Our construction of the resource graph is in fact exactly the construction of a NFA from a regular expression, up-to some subtilities (e.g. root and tails vertices, unfolding of Kleene star, etc.). We thus omit the formalities here.

Hence, deciding $\mathfrak{R}_{G_{e_1}} \approx \mathfrak{R}_{G_{e_2}}$ results in establishing $L_{e_1} = L_{e_2}$. Since $\mathfrak{R}_{G_e}$ is obtained from a regular expression e in polytime, the reduction is completed and we can conclude PSPACE-hardness for the considered problem.

■

APPENDIX 2: THE SLICE-PI CALCULUS

A. Syntax

The syntax of the slice-π calculus is as follows:

Definition		$D(\tilde{x}) \stackrel{\text{def}}{=} P$	
Process P, Q	::=	0	(inert)
	|	$\alpha.P$	(prefix)
	|	$\mathsf{new}(x)\ P$	(observable)
	|	$\mathsf{local}(y)\ Q$	(inobservable)
	|	$P \mid Q$	(parallel)
	|	$D[\tilde{a}]$	(call)
Action α	::=	τ	(silent)
	|	$\overline{a}b$	(output)
	|	$a(x)$	(input)

The structural congruence the least relation on processes satisfying:

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

B. Semantics

A system term in slice-π is of the form $\Lambda \vdash P$ where Λ is a set of names and P a process term. A reduction in slice-π is of the form $\Lambda \vdash P \xrightarrow{\mu} \Lambda' \vdash P'$ where μ is a label either:

- $\langle b \rangle$ for an open reduction when first observing b.
- b for a transparent reduction when observing b once more.
- $\bullet$ for an opaque reduction.

The reduction rules are as follows:

a) Step:

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

b) Observable:

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

c) Inobservable:

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

d) Synchronizations:

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

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

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

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

e) Context:

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

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