Parametrized automata simulation and application to service composition

Walid Belkhir, Yannick Chevalier, Michael Rusinowitch

Parametrized automata simulation and application to service composition ☆

Walid Belkhir [a], Yannick Chevalier [b], Michael Rusinowitch [a]

[a] *INRIA Nancy–Grand Est & LORIA, 54600 Villers-lès-Nancy, France*
[b] *Université Paul Sabatier & IRIT Toulouse, France*

Keywords: Parametrized automata Service Simulation preorder Composition synthesis Infinite alphabet

A B S T R A C T

The service composition problem asks whether, given a client and a community of available services, there exists an agent (called the mediator) that suitably delegates the actions requested by the client to the available community of services. We address this problem in a general setting where the agents communication actions are parametrized by data from an infinite domain and possibly subject to constraints. For this purpose, we define *parametrized automata* (PAs), where transitions are guarded by conjunction of equalities and disequalities. We solve the service composition problem by showing that the simulation preorder of PAs is decidable and devising a procedure to synthesize a mediator out of a simulation preorder. We also show that the Nonemptiness problem of PAs is PSPACE-complete.

1. Introduction

Service Oriented Architectures (SOA) consider services as self-contained components that can be published, invoked over a network and combined with other services through standardized protocols in order to dynamically build complex applications (Alonso et al., 2004; Reisig, 2008). Service composition is required when none of the existing services can fulfil some client needs but a suitable coordination of them would satisfy the client requests. How to find the right combination and how to orchestrate this combination are among the key issues for service architecture development.

☆ This research is partly supported by FP7 NESSoS project (Grant no. 256980), and extends substantially results presented at SYNASC 2013.

E-mail addresses: walid.belkhir@inria.fr (W. Belkhir), ychevali@irit.fr (Y. Chevalier), rusi@loria.fr (M. Rusinowitch).

http://dx.doi.org/10.1016/j.jsc.2014.09.029

Service composition has been studied in many works e.g. Hull and Su (2005), Martín et al. (2012), Berardi et al. (2008). The related problem of system synthesis from libraries of reusable components has been thoroughly investigated too Lustig and Vardi (2009).

In this paper we address the composition synthesis problem for web services in which the agents are *parametrized*, i.e. the client and the available services exchange data ranging over an infinite domain and they are possibly subject to some *data constraints*. More precisely, the composition synthesis problem we consider can be stated as follows: (e.g. Nourine and Toumani, 2012; Cheikh, 2009): given a client and a community of available services, compute a mediator which will enable communication between the client and the available services in such a way that each client request is forwarded to an appropriate service.

This problem was reduced to show that there exists a simulation relation between the target service (specifying an expected service behaviour for satisfying the client requests) and the asynchronous product of the available services. If such a simulation relation exists then it can be easily used to generate a mediator, that is a function that selects at each step an available service for executing an action requested by the client.

One of the most successful approaches to composition amounts to abstract services as finite-state automata (FA) and apply available tools from automata theory to synthesize a new service satisfying the given client requests from an existing community of services. However it is not obvious whether the automata-based approach to service composition can still be applied with infinite alphabets since simulation often gets undecidable in extended models like Colombo (Akroun et al., 2013). Starting from the approach initiated in Belkhir et al. (2013) our objective is to define expressive classes of automata on infinite alphabets which are well-adapted to the specification and composition of services and enjoy nice closure properties and decidable simulation preorder. Compared to our previous work (Belkhir et al., 2013) we introduce a strictly more expressive service specification formalism thanks to the use of guarded transitions.

1.1. Contributions

In this paper we rely on automata-based techniques to tackle the problem of composition synthesis of parametrized services. We introduce an extension of automata called *parametrized automata* or PAs, that allows a natural specification and decidable synthesis of parametrized services. In PAs, the transitions are labelled by letters or variables ranging over an infinite alphabets and guarded by conjunction of equalities and disequalities. Besides, some variables can be refreshed in some states, that is, these variables can be released so that new letters can be bound to them. Refreshing mechanism is useful when computations start new sessions.

We introduce a simulation preorder for PAs and show its decidability. The proof relies on a game-theoretic characterization of simulation. We show how this result can be applied to the synthesis of a mediator for web services. Although not detailed here, the simulation decision procedure can help to solve language containment problems which are important ones in formal verification. The potential applicability of our model in verification also follows from the fact that PAs are closed under intersection, union, concatenation and Kleene operator. An advantage of PAs with respect to alternative automata models is the succinct service representation they permit, thanks to the use of variable and guards (e.g. with negative conditions). This benefit will be formally supported in the paper. Finally we have shown that for PAs the Nonemptiness problem is PSPACE-complete, and the membership problem is NP-complete.

1.2. Related work

Logic-based techniques for the synthesis problem (e.g. Feuillade and Pinchinat, 2007; Pathak et al., 2007) have been considered in the literature. Narayanan and McIlraith (2002) rely on the situation calculus and Golog logic programming language for automatic construction of Web services. Waldinger (2001) employs Snark theorem-prover to get software agents distributed over the net to cooperate and answer a query or perform a task. Many works have also addressed service composition by elaborating planning techniques from artificial intelligence (e.g. Pistore et al., 2005; Hoffmann, 2008; Hoffmann et

al., 2007). Service composition problem, in the case of finite sets of actions, is reduced in Balbiani et al. (2010) to a decidable control problem for nondeterministic communicating automata.

In the *Roman* model web services behaviours are specified with activity-based finite state automata. The corresponding services composition problem was reduced to PDL satisfiability (e.g. Berardi et al., 2003; Cheikh, 2009). Thanks to the relation between PDL and Description Logic (DL), several DL reasoning tools can be applied in the latter case. The composition problem can also be reduced to computing a simulation preorder as in Berardi et al. (2008). This approach cannot be extended to the data-centric `Colombo` model of Berardi et al. (2005) for services since as we mentioned above simulation is undecidable in this case. Known decidable cases of the composition synthesis problem in Colombo framework need restrictions such as determinism or finite domain for values.

The $\texttt{Colombo}^{k,b}$ model is a restriction of the `Columbo` model in which suitable hypotheses ensure that only a finite number of domain values (and thus a finite number of different records) are considered. In Berardi et al. (2005) the composition problem for this model was proven decidable when the clients are represented as deterministic guarded automata. Besides, their proof relies on (a priori) bounding two parameters of the synthesized mediator. More precisely, they assume that the mediator is (p, q)-bounded, where p is the number of states, and q is the number of variables representing records in its global store. They conjecture that their decidability result can be obtained without mentioning these bounds. We have been able to prove that there was no need for a bound q on variables in the synthetised service nor on its number p of states.

In De Giacomo et al. (2013), the authors address a related problem of synthesizing a controller generator from which one can compute new controller when the environment or the available services change. The controller implements a fully controllable target behaviour by suitably coordinating available partially controllable behaviours that are to execute within a shared, fully observable, but non-deterministic environment. The behaviours and the environment are represented as finite state transition systems. They construct a table of states related in a simulation, and are able to update this table on the fly when the available services change. For efficiency reasons they also consider *safety games* for generating a new controller from this table.

Though our results only address the synthesis of one controller in an unmutable setting, our underlying model is more general than finite transition systems as it allows the exchange of data from an infinite domain. Though our simulation game also uses a bounded number of constants, it is worth mentionning that an unbounded execution of the synthesized controller can handle an unbounded number of different constants.

In Hassen et al. (2008) the authors also obtain an interesting positive result for the case where services are described by finite state machines but the number of instances of existing services that can be used in a given composition is not bounded a priori.

Several automata models have been designed recently for infinite alphabets. However they have not been applied to service composition. `Usages` nominal calculus (Degano et al., 2012) is a computational model based on infinite alphabets and dynamic creation of resources. PAs are incomparable with `Usages`. We give a procedure to decide the simulation preorder for PAs. Simulation has never been investigated for related classes of automata over infinite alphabets (Neven et al., 2004; Kaminski and Zeitlin, 2010; Degano et al., 2012), with the exception of the strictly less expressive class of fresh-variable automata in Belkhir et al. (2013). Compared to Belkhir et al. (2013) the proof technique has been improved and allows us to extract easily a mediator from a simulation game. Moreover we have shown that the complexity of Nonemptiness for PAs is higher than for the class defined in Belkhir et al. (2013).

Web services behaviours have been described by contracts in several works (Castagna et al., 2007). A theory of contracts that formalises the compatibility of a client to a service has been devised in Castagna et al. (2009). Contracts ensures that every possible interaction between compatible clients and services can be completed successfully. The finite contracts studied in Castagna et al. (2009) seem to be less expressive than PAs for specifying behaviours.

A service behaviour can be expressed by a Petri net too (see Hamadi and Benatallah, 2003), where actions are modelled by transitions and the state is modelled by places. However it seems hard to extend our synthesis result to Petri nets service specification because of the negative result of Jancar

(1995). However simulation between finite state machines and well-structured transition systems (and so Petri nets) is decidable as shown in Finkel and Schnoebelen (2001).

1.3. Paper organization

Section 2 recalls standard notions. Section 3 introduces the new class of parametrized automata. Section 4 studies closure properties and the complexity of Nonemptiness for PAs. Section 5 uses the parametrized automata to solve the composition synthesis problem, more precisely, Section 5.1 introduces the simulation preorder of PAs, Section 5.2 shows its decidability, Section 5.3 applies these results to service composition by devising a procedure for mediator synthesis, and Section 5.4 applies this procedure to an example. Future work directions are given in Section 6.

2. Preliminaries

Let $\mathcal{X}$ be a finite set of variables, Σ an infinite alphabet of letters. A substitution σ is an idempotent mapping $\{x_1 \mapsto \alpha_1, \ldots, x_n \mapsto \alpha_n\} \cup \bigcup_{a\in\Sigma}\{a \mapsto a\}$ with variables $x_1, \ldots, x_n$ in $\mathcal{X}$ and $\alpha_1, \ldots, \alpha_n$ in $\mathcal{X} \cup \Sigma$. We call $\{x_1, \ldots, x_n\}$ its *proper domain*, and denote it by $dom(\sigma)$. We denote by $Dom(\sigma)$ the set $dom(\sigma) \cup \Sigma$. We denote by $codom(\sigma)$ the set $\{a \in \Sigma \mid \exists x \in dom(\sigma) \text{ s.t. } \sigma(x) = a\}$. If all the $\alpha_i, i = 1 \ldots n$ are letters then we say that σ is ground. The empty substitution (i.e., with an empty proper domain) is denoted by $\emptyset$. The set of substitutions from $\mathcal{X} \cup \Sigma$ to a set A is denoted by $\zeta_{\mathcal{X},A}$, or by $\zeta_{\mathcal{X}}$, or simply by ζ if there is no ambiguity. If σ_1 and σ_2 are substitutions that coincide on the domain $dom(\sigma_1) \cap dom(\sigma_2)$, then $\sigma_1 \cup \sigma_2$ denotes their union in the usual sense. If $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$ then we denote by $\sigma_1 \uplus \sigma_2$ their *disjoint* union. We define the function $\mathcal{V} : \Sigma \cup \mathcal{X} \longrightarrow \mathcal{P}(\mathcal{X})$ by $\mathcal{V}(\alpha) = \{\alpha\}$ if $\alpha \in \mathcal{X}$, and $\mathcal{V}(\alpha) = \emptyset$, otherwise. For a function $F : A \to B$, and $A' \subseteq A$, the restriction of F on A' is denoted by $F_{|A'}$.

Definition 1. A two-players game is a tuple $\langle \mathrm{Pos}_E, \mathrm{Pos}_A, M, p^\star \rangle$, where $\mathrm{Pos}_E, \mathrm{Pos}_A$ are disjoint sets of positions: `Eloise`'s positions and `Abelard`'s positions. $M \subseteq (\mathrm{Pos}_E \cup \mathrm{Pos}_A) \times (\mathrm{Pos}_E \cup \mathrm{Pos}_A)$ is a set of *moves*, and $p^\star$ is the starting position. A strategy for the player `Eloise` is a function $\rho : \mathrm{Pos}_E \to \mathrm{Pos}_E \cup \mathrm{Pos}_A$, such that $(\wp, \rho(\wp)) \in M$ for all $\wp \in \mathrm{Pos}_E$. A (possibly infinite) play $\pi = \langle \wp_1, \wp_2, \ldots \rangle$ *follows* a strategy ρ for player `Eloise` iff $\wp_{i+1} = \rho(\wp_i)$ for all $i \in \mathbb{N}$ such that $\wp_i \in \mathrm{Pos}_E$. Let $\mathcal{W}$ be a (possibly infinite) set of plays. A strategy ρ is *winning* for `Eloise` from a set $S \subseteq \mathrm{Pos}_E \cup \mathrm{Pos}_A$ according to $\mathcal{W}$ iff every play starting from a position in S and following ρ belongs to $\mathcal{W}$.

3. Parametrized automata

In this section we define formally the class of PAs. Firstly, we illustrate the practical use of PAs through a service composition problem.

3.1. A motivating example

In Fig. 1 we have an e-commerce Web site allowing clients to open files, search for items in a large domain that can be abstracted as infinite and save them to an appropriate file depending on the type of the items (whether they are in promotion or not). The three agents: CLIENT, FILE and SEARCH communicate with messages ranging over a possibly infinite set of terms. The problem is to check whether FILE and SEARCH can be composed in order to satisfy the CLIENT requests. Following Berardi et al. (2008) the problem reduces to find a simulation between CLIENT and the asynchronous product of FILE and SEARCH. We emphasize that the variables x and y are refreshed (i.e. freed to get a new value) when passing through the state p_0. In the same way variables z and w are refreshed at p_2. The variables m and n are refreshed at q_0; the variables i and j are refreshed at r_0. For saving space, a transition labelled by a term, say `write(m, n)`, abbreviates successive transitions labelled by the root symbol and its arguments, here `write`, `m` and `n`, respectively. We notice that this example cannot be handled within the subclass of fresh-variable automata (Belkhir et al., 2013) since they do not have

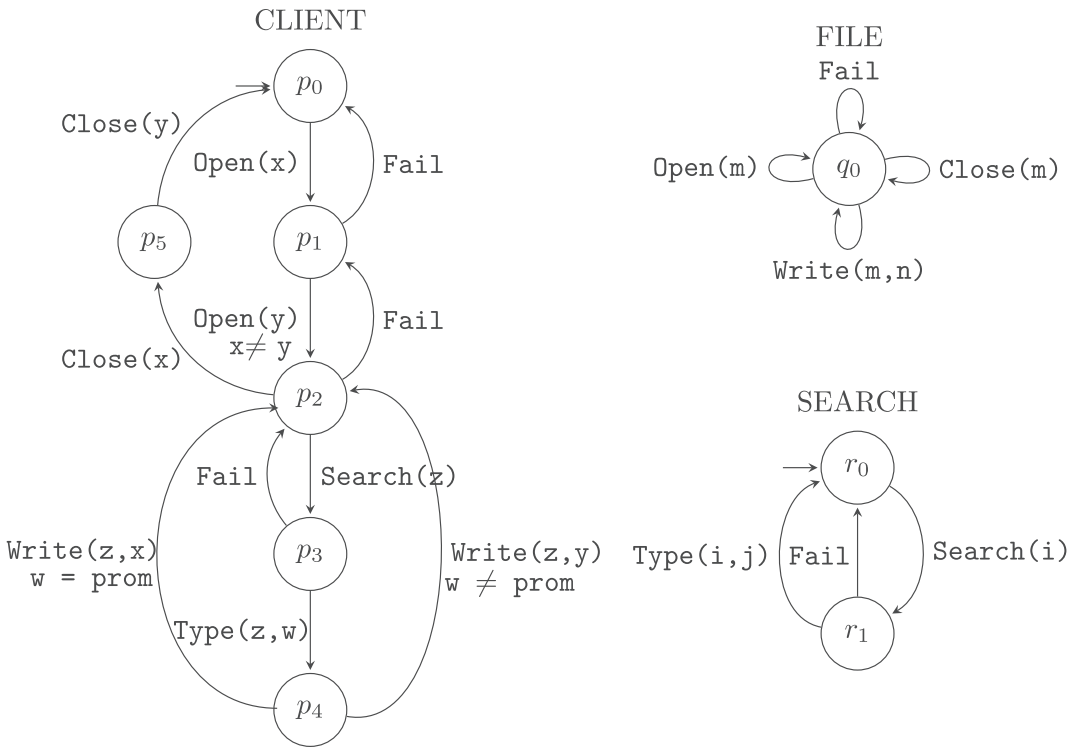


Fig. 1. PROM example.

guards. In Subsections 5.3 and 5.4 we will show how our results permit to synthesize a mediator for suitably scheduling the available services and provide the target service to the client.

Before introducing formally the class of PAs, let us first explain the main ideas behind them. The transitions of a PA are labelled with letters or variables ranging over an infinite set of letters. These transitions can also be labelled with guards consisting of equalities and disequalities. Its guard must be true for the transition to be fired. We emphasize that while reading a guarded transition some variables of the guard might be free and we need to *guess* their value. Finally, some variables are refreshed in some states, that is, variables can be *freed* in these states so that new letters can be assigned to them. Firstly, we introduce the syntax and semantics of guards.

Definition 2. The set $\mathbb{G}$ of guards over $\Sigma \cup \mathcal{X}$ is inductively defined as follows:

$$G := \texttt{true} \mid \alpha = \beta \mid \alpha \neq \beta \mid G \wedge G,$$

where $\alpha, \beta \in \Sigma \cup \mathcal{X}$. We write $\sigma \models g$ if a substitution σ satisfies a guard g.

We notice that adding the disjunction operator to the guards would not increase the expressivity of our model. A guard is atomic iff it is either `true`, an equality, or an inequality. Let g_i, $i = 1, \ldots, n$, be atomic guards. Then we define the *free* variables of a guard by extending the function $\mathcal{V}$ to a mapping $\Sigma \cup \mathcal{X} \cup \mathbb{G} \to \mathcal{P}(\mathcal{X})$ as follows: $\mathcal{V}(\bigwedge_{i=1,n} g_i) = \bigcup_{i=1,n} \mathcal{V}(g_i)$ and $\mathcal{V}(\alpha \sim \beta) = \mathcal{V}(\alpha) \cup \mathcal{V}(\beta)$, where $\sim$ belongs to $\{=, \neq\}$ and $\alpha, \beta \in \Sigma \cup \mathcal{X}$. The finite set of letters of a guard g can be defined similarly and it will be denoted by Σ_g. The application of a substitution γ to a guard g, denoted by $\gamma(g)$, is defined in the usual way. We shall write $\sigma \vdash g$ if there exists a substitution γ s.t. $\sigma \uplus \gamma \models g$. The formal definition of PAs follows.

Definition 3. A *PA* is a tuple $\mathcal{A} = \langle \Sigma, \mathcal{X}, Q, Q_0, \delta, F, \kappa \rangle$ where

- Σ is an infinite set of letters,
- $\mathcal{X}$ is a finite set of variables,
- Q is a finite set of states,

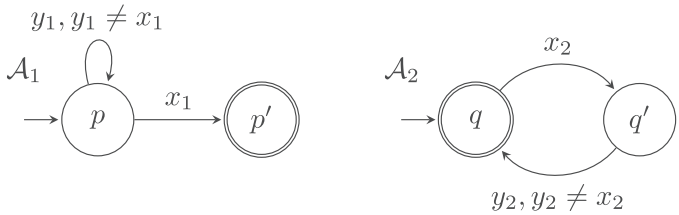


Fig. 2. Two PAs $\mathcal{A}_1$ and $\mathcal{A}_2$ where the variable y_1 is refreshed in the state p, and the variables x_2, y_2 are refreshed in the state q.

- $Q_0 \subseteq Q$ is a set of initial states,
- $\delta : Q \times (\Sigma_{\mathcal{A}} \cup \mathcal{X} \cup \{\varepsilon\}) \times \mathbb{G} \to 2^Q$ is a transition function where $\Sigma_{\mathcal{A}}$ is a finite subset of Σ,
- $F \subseteq Q$ is a set of accepting states, and
- $\kappa : \mathcal{X} \to 2^Q$ is called the refreshing function.

The run of a PA is defined over *configurations*. A configuration is a pair (γ, q) where γ is a substitution such that for all variables x in $dom(\gamma)$, $\gamma(x)$ is the current value of x, and q is a state of the PA. Intuitively, when a PA $\mathcal{A}$ is in state q, and (γ, q) is the current configuration, and there is a transition $q \xrightarrow{\alpha, g} q'$ in $\mathcal{A}$ then:

i) if α is a free variable (i.e. $\alpha \in \mathcal{X} \setminus dom(\gamma)$) then α stores the input letter and some values for all the other free variables of $\gamma(g)$ are *guessed* such that $\gamma(g)$ holds, and $\mathcal{A}$ enters the state $q' \in \delta(q, \alpha, g)$,
ii) if α is a bound variable or a letter (i.e. $\alpha \in Dom(\gamma)$) and $\gamma(\alpha)$ is equal to the input letter l then some values for all the free variables of $\gamma(g)$ are *guessed* such that $\gamma(g)$ holds, and $\mathcal{A}$ enters the state $q' \in \delta(q, \alpha, g)$.

In both cases when $\mathcal{A}$ enters state q' all the variables which are refreshed in q' are freed. Thus the purpose of guards is to compare letters and to guess new letters that might be read afterward.

For a PA $\mathcal{A}$, we shall denote by $\Sigma_{\mathcal{A}}$ the finite set of letters that appear in the transition function of $\mathcal{A}$. We shall denote by $\kappa^{-1} : Q \to 2^{\mathcal{X}}$ the function that associates to each state of the PA the set of variables being refreshed in this state. That is, $\kappa^{-1}(q) = \{x \in \mathcal{X} \mid q \in \kappa(x)\}$.

The formal definitions of run and recognized language follow.

Definition 4. Let $\mathcal{A} = \langle \Sigma, \mathcal{X}, Q, Q_0, \delta, F, \kappa \rangle$ be a PA. We define a transition relation over the configurations as follows: $(\gamma_1, q_1) \overset{a}{\Rightarrow} (\gamma_2, q_2)$, where $a \in \Sigma \cup \{\varepsilon\}$, iff there exists a substitution σ such that $dom(\sigma) \cap dom(\gamma_1) = \emptyset$ and either:

i) $a \in \Sigma$ and in this case there exists a label $\alpha \in \Sigma \cup \mathcal{X}$ such that $q_2 \in \delta(q_1, \alpha, g)$, $(\gamma_1 \uplus \sigma)(\alpha) = a$, $(\gamma_1 \uplus \sigma) \models g$ and $\gamma_2 = (\gamma_1 \uplus \sigma)_{|D}$, with $D = Dom(\gamma_1 \uplus \sigma) \setminus \kappa^{-1}(q_2)$. Or,
ii) $a = \varepsilon$ and in this case $(\gamma_1 \uplus \sigma) \models g$ and $\gamma_2 = (\gamma_1 \uplus \sigma)_{|D}$, with $D = Dom(\gamma_1 \uplus \sigma) \setminus \kappa^{-1}(q_2)$.

We denote by $\Rightarrow^\star$ the reflexive and transitive closure of $\Rightarrow$. For two configurations $\boldsymbol{c}, \boldsymbol{c}'$ and a letter $a \in \Sigma$, we write $\boldsymbol{c} \overset{a}{\to} \boldsymbol{c}'$ iff there exists two configurations $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ such that $\boldsymbol{c} \overset{\varepsilon}{\Rightarrow} {}^\star\boldsymbol{c}_1 \overset{a}{\Rightarrow} \boldsymbol{c}_2 \overset{\varepsilon}{\Rightarrow} {}^\star\boldsymbol{c}'$. A finite word, or a trace, $w = a_1 a_2 \dots a_n \in \Sigma^*$ is *recognized* by $\mathcal{A}$ iff there exists a run $(\gamma_0, q_0) \overset{a_1}{\to} (\gamma_1, q_1) \overset{a_2}{\to} \dots \overset{a_n}{\to} (\gamma_n, q_n)$, such that $q_0 \in Q_0$ and $q_n \in F$. The set of words recognized by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

Example 1. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the PAs depicted above in Fig. 2 where the variable y_1 is refreshed in the state p, and the variables x_2, y_2 are refreshed in the state q. That is, $\mathcal{A}_1 = \langle \Sigma, \{x_1, y_1\}, \{p, p'\}, \{p\}, \delta_1, \{p'\}, \kappa_1 \rangle$ with

$$\begin{cases} \delta_1(p, y_1, (y_1 \neq x_1)) = \{p\} \quad \text{and} \quad \delta_1(p, x_1, \texttt{true}) = \{p'\}, \quad \text{and} \\ \kappa_1(y_1) = \{p\} \end{cases}$$

And $\mathcal{A}_2 = \langle \Sigma, \{x_2, y_2\}, \{q, q'\}, \{q\}, \delta_2, \{q'\}, \kappa_2 \rangle$ with

$$\begin{cases} \delta_2(q, x_2, \texttt{true}) = \{q'\} \quad \text{and} \quad \delta(q', y_2, (y_2 \neq x_2)) = \{q\}, \quad \text{and} \\ \kappa_2(x_2) = \kappa_2(y_2) = \{q\}. \end{cases}$$

We notice that while making the first loop over the state p of $\mathcal{A}_1$, the variable x_1 of the guard $(y_1 \neq x_1)$ is free and its value is guessed. Then the variable y_1 is refreshed in p, and at each loop the input letter should be different than the value of the variable x_1 already guessed. More precisely, the behaviour of $\mathcal{A}_1$ on an input word is as follows. Being in the initial state p, either:

- Makes the transition $p \to p'$ by reading the input symbol and bounding the variable x_1 to it, then enters the state p'. Or,
- Makes the transition $p \to p$ by:
  (1) Reading the input symbol and bounding the variable y_1 to it.
  (2) Guessing a symbol in Σ that is different than the input symbol (i.e. the value of x_1) and bounds the variable y_1 to the guessed symbol, then enters the state p.
  (3) From the state p, refresh the variable x_1, that is, it is no longer bound to the input symbol. Then, start again.

We illustrate the run of $\mathcal{A}_1$ on the word $w = abbc$, starting from the initial configuration $(\emptyset, p)$ as follows:

$$(\emptyset, p) \xrightarrow{a} \big(\{x_1 \mapsto c\}, p\big) \xrightarrow{b} \big(\{x_1 \mapsto c\}, p\big) \xrightarrow{b} \big(\{x_1 \mapsto c\}, p\big) \xrightarrow{c} \big(\{x_1 \mapsto c\}, p'\big)$$

Hence, the language $L(\mathcal{A}_1)$ consists of all the words in $\Sigma^\star$ in which the last letter is different than all the other letters. By following similar reasoning, we get $L(\mathcal{A}_2) = \{w_1 w_1' \cdots w_n w_n' \mid w_i, w_i' \in \Sigma, n \geq 1, \text{ and } w_i \neq w_i', \forall i \in [n]\}$.

4. Properties of parametrized automata

Closure properties are important for the modular development of services. PAs enjoy the same closure properties as finite automata except for complementation (see Theorem 5). Moreover checking whether a service satisfies a policy can often be reduced to show that the intersection of the service PA with a PA specifying the forbidden executions is empty. We give an example in Subsection 4.1. Since the class of PAs is closed under intersection, (Theorem 5), and Nonemptiness is decidable as shown in Subsection 4.1, we can perform this verification. On the other hand if the policy is expressed as a language of *authorized* traces and if this language is recognized by a PA, say $\mathcal{Q}$, then checking whether a service $\mathcal{M}$ respects the policy amounts to check the containment $L(\mathcal{M}) \subseteq L(\mathcal{Q})$. However, the containment for PAs is undecidable, as stated in Theorem 6 below. In practical cases it is often sufficient (in order to get this inclusion) to show the existence of a simulation preorder between $\mathcal{M}$ and $\mathcal{Q}$: this problem is decidable, as we prove it in the following (Theorem 26).

Therefore we study the closure properties of PAs and the decidability and complexity of classical decision problems: Nonemptiness (given $\mathcal{A}$, is $L(\mathcal{A}) \neq \emptyset$?), Membership (given a word w and $\mathcal{A}$, is $w \in L(\mathcal{A})$?), Universality (given $\mathcal{A}$, is $L(\mathcal{A}) = \Sigma^*$?), and Containment (given $\mathcal{A}_1$ and $\mathcal{A}_2$, is $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$?).

We have that:

Theorem 5. *PAs are closed under union, concatenation, Kleene operator and intersection. They are not closed under complementation.*

Proof. Let $\mathcal{A}_1 = \langle \Sigma_1, \mathcal{X}_1, Q_1, q_0^1, \delta_1, F_1, \kappa_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma_2, \mathcal{X}_2, Q_2, q_0^2, \delta_2, F_2, \kappa_2 \rangle$ be two PAs. Up to variable renaming it is sufficient to consider the closure under the above operations for two PAs that do not share variables.

The closure under Kleene operation and concatenation holds since PAs have ε-transitions. More precisely, the Kleene closure $\mathcal{A}_1^\star$ amounts to adding an (unguarded) ε-transition between the accepting states and initial states of $\mathcal{A}_1$. And the concatenation $\mathcal{A}_1 \cdot \mathcal{A}_2$ amounts to adding an (unguarded) ε-transition between the accepting states of $\mathcal{A}_1$ and the initial states of $\mathcal{A}_2$.

The closure under intersection follows from the fact that the intersection of two PAs $\mathcal{A}_1$ and $\mathcal{A}_2$ denoted by $\mathcal{A}_1 \cap \mathcal{A}_2$ can be defined as follows:

$$\mathcal{A}_1 \cap \mathcal{A}_2 = \left\langle \Sigma_1 \cup \Sigma_2, \mathcal{X}_1 \cup \mathcal{X}_2, Q_1 \times Q_2, q_0^1 \times q_0^2, \delta, F_1 \times F_2, \kappa \right\rangle,$$

where δ and κ are defined by:

$$\begin{cases} (q_1', q_2') \in \delta((q_1, q_2), (\alpha_1, (\alpha_1 = \alpha_2) \wedge g_1 \wedge g_2)) \quad \text{iff} \quad \begin{array}{l} q_1' \in \delta_1(q_1, \alpha_1, g_1) \text{ and} \\ q_2' \in \delta_2(q_2, \alpha_2, g_2). \end{array} \\ (q_1, q_2) \in \kappa(x) \quad \text{iff} \quad q_1 \in \kappa_1(x) \text{ or } q_2 \in \kappa_2(x). \end{cases}$$

The proof that $L(\mathcal{A}_1) \cap L(\mathcal{A}_2) = L(\mathcal{A}_1 \cap \mathcal{A}_2)$ is straightforward. □

For the main decision procedure we have that:

Theorem 6. *For PAs, Membership is NP-complete, Universality and Containment are undecidable.*

Proof. Let $\mathcal{A}$ be a PA and $w = w_1 \cdots w_n$ a word in $\Sigma^\star$.

For the upper bound of the membership, a non deterministic polynomial algorithm guesses a path in $\mathcal{A}$ of length $|w|$ such that the final state is accepting; and a series of substitutions $\sigma_1, \ldots, \sigma_{|w|}$, where $\sigma_i : \mathcal{X} \to \{w_j, 1 \leq j \leq |w|\}$, then checks whether the corresponding run on w is possible. The lower bound, i.e. the NP-hardness, follows from the fact that the membership problem for PAs without guards, i.e. fresh-variable automata, is NP-complete (Belkhir et al., 2013, Theorem 3).

The undecidability of Containment and Universality is a consequence of the undecidability of these problems for subclasses of PAs (Neven et al., 2004). □

It is worth mentionning that regular languages can be complemented within the class of PAs:

Proposition 7. *The complement of a regular language is PA-recognizable. That is, given a FA F there exists a PA $\mathcal{A}$ such that $L(\mathcal{A}) = \Sigma^\star \setminus L(F)$.*

Proof. The construction of $\mathcal{A}$ is similar to the one for FAs (over a finite alphabet). We assume that F is deterministic. Firstly, we make the completion of F, i.e. we construct an equivalent PA so that for each state q of F and for each letter $l \in \Sigma$ there is a unique transition outgoing from q that reads l. Secondly, we swap the accepting and non-accepting states.

Formally, assume $F = \langle \Sigma, Q, p_0, \delta, F \rangle$, with $Q = \{q_1, \ldots, q_n\}$, and define $\mathcal{A} = \langle \Sigma, \mathcal{X}, Q', p_0, \delta', F', \kappa \rangle$ by

$$\begin{cases} \mathcal{X} = \{x_1, \ldots, x_n\} \\ Q' = Q \cup \{q_{i1}, q_{i2}, i = 1, \ldots, n\} \\ F' = (Q \setminus F) \cup \{q_{i1}, q_{i2}, i = 1, \ldots, n\} \\ \delta' = \delta \cup \{q_i \xrightarrow{x_i, \bigwedge_j (x_i \neq a_j)} q_{i1} \mid \text{for all } a_j \text{ s.t. } q_i \xrightarrow{a_j} q_{i'} \in \delta\} \cup \{q_{i2} \xrightarrow{x_i} q_{i2}\} \\ \kappa(x_i) = \{q_{i2}\} \end{cases}$$

Notice that F rejects a word w iff $\mathcal{A}$ accepts w. □

We show in the next subsection that Nonemptiness for PAs is PSPACE-complete.

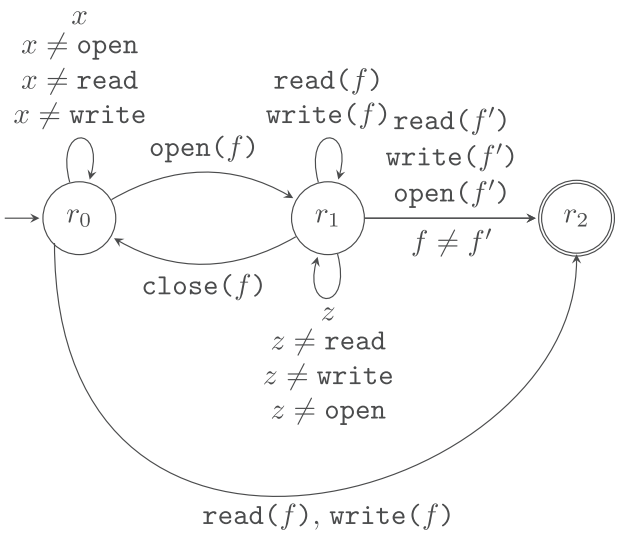


Fig. 3. A usage policy for opening and reading files.

4.1. Nonemptiness is in PSPACE

We motivate the study of PA Nonemptiness problem through an example of a service policy for reading and writing files, which is a variant of the example in Degano et al. (2012) and Reger et al. (2013). The PA of Fig. 3 defines a usage policy for opening and reading files by specifying (i.e. accepting) the forbidden executions. It states that (i) a file named f must be open before being used, where f is a variable, and (ii) the number of files which are open at the same time is at most one. More precisely once a file f is open the PA enters state r_1 allowing the file f to be read and written. Besides, any attempt to read or write in a file whose name is different than f leads the PA to the offending state r_2. On the other hand, starting from r_0 any attempt to read or write in a file that is not open leads the PA to the offending state r_2. We recall that a transition labelled by a term, say `open(f)`, abbreviates two successive transitions labelled by the root symbol and its arguments, here `open` and f, respectively. In this PA, the variables x and f are refreshed at r_0, the variable z is refreshed at r_1, and the variables f' is not refreshed in any state. As mentioned above, checking whether a service specified by PA M (with all states accepting, to be simple) respects the policy amounts to check the emptiness of $M \cap P$.

We recall that Nonemptiness is NL-complete for the subclass of fresh-variable automata (Belkhir et al., 2013) and it is NP-complete for the subclass of register automata (Sakamoto and Ikeda, 2000). Firstly we show that Nonemptiness is in PSPACE. Given a PA $\mathcal{A}$, we shall show that $\mathcal{A}$ recognizes a non-empty language over Σ iff $\mathcal{A}$ recognizes a non-empty language over a *finite* set of letters. For this purpose, and in order to relate the two runs of $\mathcal{A}$ (the one over an infinite alphabet and the one over a finite alphabet) we introduce a *coherence* relation between substitutions.

Definition 8. Let C be a finite subset of Σ. The coherence relation $\bowtie_C \subseteq \zeta \times \zeta$ between substitutions is defined by $\bar{\sigma} \bowtie_C \sigma$ iff the three following conditions hold:

(1) $dom(\bar{\sigma}) = dom(\sigma)$,
(2) If $\bar{\sigma}(x) \in C$ or $\sigma(x) \in C$ then $\bar{\sigma}(x) = \sigma(x)$, for any variable $x \in dom(\sigma)$, and
(3) for any variables $x, y \in dom(\sigma)$, $\bar{\sigma}(x) = \bar{\sigma}(y)$ iff $\sigma(x) = \sigma(y)$.

The claims in the following lemma are not hard to prove.

Lemma 9. *Let $C \subseteq \Sigma$ be a finite set of letters, $\bar{\sigma}$ and σ two substitutions, x, and a a letter in C. The following hold.*

(1) *If* $\bar{\sigma} \bowtie_C \sigma$ *then* $|codom(\bar{\sigma})| = |codom(\sigma)|$.
(2) *If* $\bar{\sigma} \bowtie_C \sigma$ *and* $D \subseteq Dom(\sigma)$ *then* $\bar{\sigma}_{|D} \bowtie_C \sigma_{|D}$.
(3) *If* $(\bar{\sigma}_1 \uplus \bar{\sigma}_2) \bowtie (\sigma_1 \uplus \sigma_2)$ *with* $dom(\bar{\sigma}_i) = dom(\sigma_i)$, *then* $\bar{\sigma}_i \bowtie \sigma_i$, *for* $i = 1, 2$.
(4) *If* $\bar{\sigma} \bowtie_C \sigma$ *and* γ *is a substitution with* $dom(\gamma) \cap dom(\sigma) = \emptyset$ *and* $codom(\gamma) \subseteq C$, *then* $\bar{\sigma} \uplus \gamma \bowtie_C \sigma \uplus \gamma$.
(5) *If* $\bar{\sigma} \bowtie_C \sigma$ *with* $\bar{\sigma}(y) = \bar{a}$ *and* $\sigma(y) = a$ *for some variable* y, *and* $x \notin dom(\sigma)$ *then* $\bar{\sigma} \uplus \{x \mapsto \bar{a}\} \bowtie_C \sigma \uplus \{x \mapsto a\}$.
(6) *If* $\bar{\sigma} \bowtie_C \sigma$ *and* $\bar{a} \notin C \cup codom(\bar{\sigma})$ *and* $a \notin C \cup codom(\sigma)$ *and* $x \notin dom(\sigma)$ *then* $\bar{\sigma} \uplus \{x \mapsto \bar{a}\} \bowtie_C \sigma \uplus \{x \mapsto a\}$.
(7) *Let* g *be a guard such that* $\Sigma_g \subseteq C$. *If* $\sigma \bowtie_C \bar{\sigma}$, *then* $\sigma \models g$ *iff* $\bar{\sigma} \models g$.
(8) *Assume that* $\sigma \bowtie_C \bar{\sigma}$, *and let* g *be a guard such that* $\Sigma_g \subseteq C$. *Then,* $\sigma \vdash g$ *iff* $\bar{\sigma} \vdash g$.

In order to prove that only a finite number of constants is needed to cover all possible moves in the simulation game, we have to prove that if two substitutions are equivalent *wrt* the winning condition—i.e., if $\sigma_1 \bowtie_C \sigma_2$ where C will be the set of letters occurring in the transitions of the simulation game—then for each substitution γ_1, there exists a substitution γ_2 such that for any transition with a guard g such that $\sigma_1 \uplus \gamma_1 \models g$ one also has $\sigma_2 \uplus \gamma_2 \models g$. This substitution γ_2 can actually be computed explicitely from $\sigma_1, \sigma_2, \gamma_1$.

In what follows we let $S_1 \subseteq \Sigma$, $S_2 \subseteq \Sigma$ be two (possibly infinite) sets of letters with $|S_1 \setminus S_2| > |\mathcal{X}|$ and $|S_2 \setminus S_1| > |\mathcal{X}|$ and $S_1 \cap S_2 = C \neq \emptyset$.

Definition 10. We define the functions

$$\Theta_C^{S_1,S_2}, \boldsymbol{\Theta}_C^{S_1,S_2} : \xi_{\mathcal{X},S_1} \times \xi_{\mathcal{X},S_1} \times \xi_{\mathcal{X},S_2} \to \xi_{\mathcal{X},S_2}$$

as follows. Let $\sigma_1, \gamma_1 \in \xi_{\mathcal{X},S_1}$, $\sigma_2 \in \xi_{\mathcal{X},S_2}$. Then, $\Theta_C^{S_1,S_2}(\sigma_1, \gamma_1, \sigma_2)$ is defined only when $|dom(\gamma_1)| = 1$ and $dom(\gamma_1) \cap dom(\sigma_1) = \emptyset$ and $\sigma_1 \bowtie_C \sigma_2$ by:

$$\Theta_C^{S_1,S_2}(\sigma_1, \gamma_1, \sigma_2) = \begin{cases} \gamma_1 & \text{if } \gamma_1(x) \in C \\ \{x \mapsto \sigma_2(y)\} & \text{if } \gamma_1(x) \in codom(\sigma_1) \setminus C \text{ and } \sigma_1(y) = \gamma_1(x),\ y \in \mathcal{X} \\ \{x \mapsto get(S_2 \setminus codom(\sigma_2))\} & \text{if } \gamma_1(x) \in S_1 \setminus (C \cup codom(\sigma_1)) \end{cases}$$

where $dom(\gamma_1) = \{x\}$. And $\boldsymbol{\Theta}_C^{S_1,S_2}(\sigma_1, \gamma_1, \sigma_2)$ is defined only when $dom(\gamma_1) \cap dom(\sigma_1) = \emptyset$ by:

$$\boldsymbol{\Theta}_C^{S_1,S_2}(\sigma_1, \gamma_1, \sigma_2) = \begin{cases} \Theta_C^{S_1,S_2}(\sigma_1, \gamma_1, \sigma_2) & \text{if } |\gamma_1| = 1 \\ \gamma_2' \uplus \boldsymbol{\Theta}_C^{S_1,S_2}(\sigma_1 \uplus \gamma_1', \gamma_1'', \sigma_2 \uplus \gamma_2') & \text{if } |\gamma_1| \geq 2,\ \gamma_1 = \gamma_1' \uplus \gamma_1'' \text{ and } |\gamma_1'| = 1, \text{ where } \gamma_2' = \Theta_C^{S_1,S_2}(\sigma_1, \gamma_1', \sigma_2) \end{cases}$$

The function $\boldsymbol{\Theta}$ will be used in the proof of Lemma 12 and in the proof of the decidability of the simulation preorder in Section 5.2. Let us now prove that this construction is correct.

Lemma 11. *Let* $\sigma, \gamma_1 \in \xi_{\mathcal{X},S_1}$ *and* $\sigma' \in \xi_{\mathcal{X},S_2}$ *be substitutions with* $dom(\sigma) \cap dom(\gamma_1) = \emptyset$ *and* $\sigma \bowtie_C \sigma'$. *We have that*

$$(\sigma \uplus \gamma_1) \bowtie_C \left(\sigma' \uplus \boldsymbol{\Theta}_C^{S_1,S_2}(\sigma, \gamma_1, \sigma')\right) \tag{1}$$

Proof. By induction on $|dom(\gamma_1)|$.

Base Case. If $|dom(\gamma_1)| = 1$ then assume $dom(\gamma_1) = \{x\}$ and let $\gamma_2 = \boldsymbol{\Theta}_C^{S_1,S_2}(\sigma, \gamma_1, \sigma')$. We distinguish three cases depending on $\gamma_1(x)$.

- If $\gamma_1(x) \in C$ then it follows from the definition of $\Theta_C^{S_1,S_2}$ that $\gamma_2 = \gamma_1$. From Item (4) of Lemma 9 we get $\sigma \uplus \gamma_1 \bowtie_C \sigma' \uplus \gamma_1$.

- If $\gamma_1(x) \in codom(\sigma) \setminus C$ then in this case we recall that $\gamma_2 = \{x \mapsto \sigma'(y)\}$ where $\sigma(y) = \gamma_1(x)$ for some variable $y \in \mathcal{X}$. The fact that $(\sigma \uplus \{x \mapsto \gamma_1(x)\}) \bowtie_C \sigma' \uplus \{x \mapsto \sigma'(y)\}$ follows from Item (5) of Lemma 9.
- Otherwise, i.e. if $\gamma_1(x) \in S_1 \setminus (C \cup codom(\sigma))$ then the fact that $(\sigma \uplus \{x \mapsto \gamma_1(x)\}) \bowtie_C (\sigma' \uplus \{x \mapsto get(S_2 \setminus codom(\sigma'))\})$ follows from Item (6) of Lemma 9.

Induction Case. Assume $\gamma_1 = \gamma_1' \uplus \gamma_1''$ with $|\gamma_1'| = 1$. Let

$$\begin{cases} \gamma_2' = \Theta_C^{S_1,S_2}(\sigma, \gamma_1', \sigma'), & \text{and} \\ \gamma_2'' = \boldsymbol{\Theta}_C^{S_1,S_2}(\sigma \uplus \gamma_1', \gamma_1'', \sigma' \uplus \gamma_2') & \text{and} \\ \gamma_2 = \boldsymbol{\Theta}_C^{S_1,S_2}(\sigma, \gamma_1, \sigma') = \gamma_2' \uplus \gamma_2'' \end{cases}$$

From the induction hypothesis it follows that

$$\begin{cases} \sigma \uplus \gamma_1' \bowtie_C \sigma' \uplus \gamma_2' \\ (\sigma \uplus \gamma_1') \uplus \gamma_1'' \bowtie_C (\sigma' \uplus \gamma_2') \uplus \gamma_2'' \end{cases}$$

Therefore

$$\sigma \uplus \gamma_1 \bowtie_C \sigma' \uplus \gamma_2 \qquad \square$$

Lemma 12. *Let $\mathcal{A}$ be a PA over Σ with k variables and m letters $\Sigma_{\mathcal{A}} = \{c_1, \ldots, c_m\}$. Let $\boldsymbol{\Sigma} = \{a_1, \ldots, a_k, c_1, \ldots, c_m\}$. Then, $\mathcal{A}$ recognizes a non-empty language over $\Sigma^\star$ if, and only if, it recognizes a non-empty language over $\boldsymbol{\Sigma}^\star$.*

Proof. Let $C = \{c_1, \ldots, c_n\}$. We show that there is a run $(\sigma_0, q_0) \to \ldots \to (\sigma_n, q_n)$ over $\Sigma^\star$ in $\mathcal{A}$ iff there is a run $(\sigma_0', q_0) \to \ldots \to (\sigma_n', q_n)$ over $\boldsymbol{\Sigma}^\star$ in $\mathcal{A}$ such that $\sigma_i \bowtie_C \sigma_i'$ for all $i = 0, \ldots, n$. The proof is by induction on n in both directions. The base case $n = 0$ holds trivially since $\sigma_0 = \sigma_0' = \emptyset$. Assume that the claim holds up to n and let us prove it for $n+1$.

$\Rightarrow$) Assume there is a transition $q_n \overset{\alpha_n, g_n}{\to} q_{n+1}$ in $\mathcal{A}$ where $\alpha_n \in \Sigma \cup \mathcal{X}$ and g_n is a guard. From the induction hypothesis we have that $\sigma_n \bowtie_C \sigma_n'$. It follows from Item (7) of Lemma 9 that $\sigma_n(g_n)$ holds iff $\sigma_n'(g_n)$ holds. Thus, the transition in $\mathcal{A}$ over $\boldsymbol{\Sigma}$ is possible. We describe next this transition. From Definition 4 of the run of PAs, there exists a substitution $\gamma_n : \mathcal{V}(\sigma_n(\alpha_n)) \cup \mathcal{V}(\sigma_n(g_n)) \to \Sigma$ such that $(\gamma_n \uplus \sigma_n)(g_n)$ holds. Hence, we must find a substitution $\gamma_n' : \mathcal{V}(\sigma_n'(\alpha_n)) \cup \mathcal{V}(\sigma_n'(g_n)) \to \boldsymbol{\Sigma}$ such that $(\gamma_n' \uplus \sigma_n')(g_n)$ holds. We define γ_n' by $\gamma_n' = \boldsymbol{\Theta}_C^{\Sigma,\boldsymbol{\Sigma}}(\sigma_n, \gamma_n, \sigma_n')$. From Lemma 11 we have that $(\sigma_n \uplus \gamma_n) \bowtie_C (\sigma_n' \uplus \gamma_n')$. Hence from Item (7) of Lemma 9 it follows that $(\gamma_n' \uplus \sigma_n')(g_n)$ holds. It remains to show that $\sigma_{n+1} \bowtie_C \sigma_{n+1}'$. But $\sigma_{n+1} = (\sigma_n \uplus \gamma_n)_{|D}$ and $\sigma_{n+1}' = (\sigma_n' \uplus \gamma_n')_{|D}$, for some set $D \subseteq \mathcal{X}$. From Item (2) of Lemma 9 it follows that $\sigma_{n+1} \bowtie_C \sigma_{n+1}'$.

$\Leftarrow$) Same proof but we call the function $\boldsymbol{\Theta}_C^{\Sigma,\Sigma}(\sigma_n', \gamma_n', \sigma_n)$ instead of $\boldsymbol{\Theta}_C^{\Sigma,\boldsymbol{\Sigma}}(\sigma_n, \gamma_n, \sigma_n')$. $\square$

Definition 13. (Restricted configuration) A *restricted configuration* of a PA $\mathcal{A}$ with k variables $x_1, \ldots, x_k$ and m constants $c_1, \ldots, c_m$ is a tuple $(q, \alpha_1, \ldots, \alpha_k)$ where $\alpha_1, \ldots, \alpha_k \in \{a_1, \ldots, a_k, c_1, \ldots, c_m\}$.

Note that the number of different restricted configurations is exponentially bounded in the size of $\mathcal{A}$. From Lemma 12 it follows that:

Lemma 14. *Assume $\mathcal{A}$ is a PA that recognizes a non-empty language. Then there is an accepting run $\boldsymbol{q}_1 \to \ldots \to \boldsymbol{q}_l$ such that $i \neq j$ implies $\boldsymbol{q}_i \neq \boldsymbol{q}_j$, and each $\boldsymbol{q}_i$ is a restricted configuration.*

As a corollary, we obtain that if a PA recognizes a non-empty language L it has an accepting run consisting of restricted configurations that each appear at most once. Hence its length is less than the number of configurations, which can be encoded in binary in space linear in the size of $\mathcal{A}$. We thus obtain:

Theorem 15. *The Nonemptiness problem for PAs is in PSPACE.*

4.2. Nonemptiness is PSPACE-hard

To show that Nonemptiness of PAs is PSPACE-hard, we reduce the reachability problem for bounded one-counter automata (known to be PSPACE-hard) to the Nonemptiness problem of PAs. In the rest of this subsection, we first introduce bounded one-counter automata, and then proceed to PAs.

Definition 16 *(Bounded one-counter automata).* (See Fearnley and Jurdzinski, 2013.) A *bounded one-counter automaton* (Boca) is a tuple (Q, b, Δ, q_0) where Q is a finite set of states, $b \in \mathbb{N}$ is a global counter bound, q_0 is the initial state, and $\Delta \subset Q \times [-b, b] \times Q$ is the transition relation.

A Boca configuration is a tuple (q, p) with $q \in Q$ and $0 \leq p \leq b$. Given $\tau = (q, p, q') \in \Delta$ and two configurations $c_1 = (q_1, p_1)$ and $c_2 = (q_2, p_2)$ we denote $c_1 \rightarrow^{\tau} c_2$ if $q_1 = q$, $q_2 = q'$, $p_2 = p_1 + p$.

Note that mandating that c_2 is a configuration implies two implicit inequality testing $0 \leq p_1 + p \leq b$. The reachability problem for Boca consists in determining whether there exists a sequence of transitions starting from the configuration $(q_0, 0)$ and ending in a configuration (q, p). This problem has been shown to be PSPACE-complete in Fearnley and Jurdzinski (2013). We reduce it to PA's Nonemptiness as follows:

- Assuming $2^{k-1} \leq b \leq 2^k$, we build a PA with $k+1$ variables $x_1, \ldots, x_k, r$, where each x_i contains the ith bit in the binary representation of the counter p, and r stands for a register that will hold the current carry of a bit-per-bit binary addition;
- Addition of a positive constant to the counter is encoded by a sequence of $k+1$ half-adders. Each half-adder reads x_i, and depending on the value of r and the ith bit of the number to add, refreshes x_i and r, and sets (using a guard) the correct new value for these two variables. A constant number of states is necessary at each half-addition step, and thus the encoding is linear in $k = \lceil \log_2 b \rceil$, and thus in the size of the input Boca. Note that the result is greater than $2^{\lceil \log_2 b \rceil}$ if, and only if, the register r contains 1 in the end;
- Bitwise 2-complement of a register is similarly performed bit-per-bit, using a number of states linear in k;
- Substraction of a positive integer is encoded by two bitwise 2-complement computations on the counter around a positive integer addition. If the final counter is negative the intermediate addition step will yield a number greater than 2^k;
- Finally, we test after each addition of a positive integer that the counter is smaller or equal to b by adding $2^k - b$ to it, and then substracting this same number;
- The final configuration (q_f, p_f) is encoded by a transition from states of the PA encoding q_f to its unique final state having as guard the equality test of the variables $x_1, \ldots, x_k$ with the bits in the binary encoding of p_f.

The details are in the companion report to this paper. The PSPACE-hardness of the reachability problem for Boca and Theorem 15 imply:

Theorem 17. *The Nonemptiness problem for PAs is PSPACE-complete.*

Note that we need only two letters in $\Sigma_{\mathcal{A}}$ (i.e. $\Sigma_{\mathcal{A}} = \{0, 1\}$) in the encoding we have employed to prove the hardness result.

5. Service synthesis with parametrized automata

We define and study the simulation preorder for PAs, an extension of the simulation preorder for FAs. Then we show how to synthesis a mediator (i.e. a PA) allowing the communication between a

client and the community of available services. To simplify the presentation, we shall only consider in this section PAs without ε-transitions and in which there is a unique initial state and all the states are accepting. The proof of the decidability of simulation for PAs with ε-transitions is discussed at the end of Subsection 5.2.

5.1. Simulation preorder for PAs, and symbolic games

Simulation for PAs is a relation defined over pairs of configurations instead of pairs of states as it is the case for FAs.

Definition 18 *(Simulation preorder).* Let $\mathcal{A}_1 = \langle \Sigma, \mathcal{X}_1, Q_1, q_0^1, \delta_1, F_1, \kappa_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, \mathcal{X}_2, Q_2, q_0^2, \delta_2, F_2, \kappa_2 \rangle$ be two PAs where $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$. A simulation of $\mathcal{A}_1$ by $\mathcal{A}_2$ is a relation

$$\trianglelefteq \subseteq (\zeta_{\mathcal{X}_1,\Sigma} \times Q_1) \times (\zeta_{\mathcal{X}_2,\Sigma} \times Q_2)$$

such that:

1. $(\emptyset, q_0^1) \trianglelefteq (\emptyset, q_0^2)$, and
2. if $(\sigma_1, q_1) \trianglelefteq (\sigma_2, q_2)$ and if $(\sigma_1, q_1) \xrightarrow{a} (\sigma_1', q_1')$ for $a \in \Sigma$ then there exists a state $q_2' \in Q_2$ and a substitution σ_2' such that $(\sigma_2, q_2) \xrightarrow{a} (\sigma_2', q_2')$ and $(\sigma_1', q_1') \trianglelefteq (\sigma_2', q_2')$.

In order to show the decidability of simulation for PAs we shall provide a game-theoretic formulation of simulation in terms of *symbolic simulation games*. Roughly speaking, the arena of a symbolic game is a PA in which each state is controlled by one of the two players, `Eloise` or `Abelard`. From a state under his control, the player chooses an outgoing transition and instantiates the (possible) free variable that labels this transition and all the free variables in the constraint of this transition. Firstly, we introduce symbolic games in Definition 19 and their concretisation in Definition 20. Then, we show in Definition 21 how to formulate the simulation preorder for PAs as a symbolic game.

Definition 19 *(Symbolic games).* A symbolic game is a pair $(\mathcal{A}, \lambda)$ where $\mathcal{A} = \langle \Sigma, \mathcal{X}, Q, q_0, \delta, F, \kappa \rangle$ is a PA and $\lambda : Q \to \{\mathtt{E}, \mathtt{A}\}$ is the labelling function. The states labelled by `E` (resp. `A`) correspond to `Eloise` (resp. `Abelard`) states.

The concretisation of a symbolic game amounts to instantiate its variables from a (possibly infinite) set of letters. The resulting game is a two-players game in the sense of Definition 1. The formal definition follows.

Definition 20 *(Concretisation of symbolic games).* Let $(\mathcal{A}, \lambda)$ be a symbolic game where $\mathcal{A} = \langle \Sigma, \mathcal{X}, Q, q_0, \delta, F, \kappa \rangle$. Let $S \subseteq \Sigma$ be a set of letters. The *concretisation* of the symbolic game $(\mathcal{A}, \lambda)$ with letters in S, denoted by $\mathcal{G}(\mathcal{A}, \lambda, S)$, is the two players game $\langle \mathrm{Pos}_{\mathtt{E}}, \mathrm{Pos}_{\mathtt{A}}, M, p^\star \rangle$ where the initial position is $p^\star = (\emptyset, q_0)$ and the set of positions $\mathrm{Pos} = \mathrm{Pos}_{\mathtt{A}} \cup \mathrm{Pos}_{\mathtt{E}}$ is the set of positions containing $p^\star$ and reachable from the moves M:

$$M = \left\{ (\sigma, q) \to (\sigma', q') \text{ where } (\sigma, q) \xrightarrow{a} (\sigma', q') \text{ for all } a \in \Sigma \right\}$$

Finally,

$$\begin{cases} \mathrm{Pos}_{\mathtt{E}} & = \{(\sigma, q) \in \mathrm{Pos} \text{ where } \sigma \in \zeta_{\mathcal{X},S} \text{ and } \lambda(q) = \mathtt{E}\} \\ \mathrm{Pos}_{\mathtt{A}} & = \{(\sigma, q) \in \mathrm{Pos} \text{ where } \sigma \in \zeta_{\mathcal{X},S} \text{ and } \lambda(q) = \mathtt{A}\} \end{cases}$$

Definition 21 *(Symbolic games for simulation).* Let $\mathcal{A}_1 = \langle \Sigma, \mathcal{X}_1, Q_1, q_0^1, \delta_1, F_1, \kappa_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, \mathcal{X}_2, Q_2, q_0^2, \delta_2, F_2, \kappa_2 \rangle$ be two PAs and let $S \subseteq \Sigma$ be a set of letters. The *symbolic simulation game* of $\mathcal{A}_1$ by $\mathcal{A}_2$ is the symbolic game $(\mathcal{M}, \lambda)$ where $\mathcal{M} = \langle \Sigma, \mathcal{X}, Q, q_0, \delta, F, \kappa \rangle$ is defined by:

$$
\begin{aligned}
Q \subseteq \quad & Q_1 \times Q_2 \cup Q_1 \times Q_2 \times (\Sigma_{\mathcal{A}_1} \cup \mathcal{X}_1) \\
q_0 = {} & q_0^1 \times q_0^2 \\
\delta = {} & \Delta_0 \cup \Delta_1, \quad \text{where} \begin{cases} \Delta_0 = \{(q_1, q_2) \xrightarrow{\alpha_1, g_1} (q_1', q_2, \alpha_1), & \text{for all } q_1 \xrightarrow{\alpha_1, g_1} q_1' \in \delta_1\} \\ \Delta_1 = \{(q_1, q_2, \alpha_1) \xrightarrow{\alpha_2, g_2 \wedge (\alpha_1 = \alpha_2)} (q_1, q_2'), & \text{for all } q_2 \xrightarrow{\alpha_2, g_2} q_2' \in \delta_2\} \end{cases} \\
\kappa = {} & \kappa_1 \times \kappa_2
\end{aligned}
$$

and the labelling function λ is defined by $\lambda((q_1, q_2)) = \mathtt{A}$ and $\lambda(q_1, q_2, \alpha) = \mathtt{E}$, for every (q_1, q_2), $(q_1, q_2, \alpha) \in Q$.

The simulation game of $\mathcal{A}_1$ by $\mathcal{A}_2$ over letters in S, denoted by $\mathcal{G}_S(\mathcal{A}_1, \mathcal{A}_2)$, is the concrete game $\mathcal{G}(\mathcal{M}, \lambda, S)$.

Finally, any infinite play in $\mathcal{G}(\mathcal{M}, \lambda, S)$ is winning for `Eloise`, and any finite play is losing for the player who cannot move.

The simulation problem for PAs is the following: given two PAs $\mathcal{A}_1$ and $\mathcal{A}_2$, is $\mathcal{A}_1 \trianglelefteq \mathcal{A}_2$? This amounts to asking whether player `Eloise` has a winning strategy in the concrete game $\mathcal{G}_\Sigma(\mathcal{A}_1, \mathcal{A}_2)$.

The following lemma states an immediate property of the symbolic games.

Lemma 22. *Let $(\mathcal{M}, \lambda)$ be a symbolic game and $S \subset \Sigma$. If S is finite then the concrete game $\mathcal{G}_S(\mathcal{M}, \lambda)$ is finite too.*

5.2. Decidability of the simulation problem

We show that the simulation problem is decidable by showing that solving symbolic games is decidable. The idea is that the problem of solving a symbolic game can be reduced to solving the same game in which the two players instantiate the variables from a *finite* set of letters, see Proposition 25. In order to relate these two games we need to adapt the notion of coherence between substitutions given in Definition 8 to the coherence between game positions. The definition of the coherence between game positions, still denoted by $\bowtie_C$, follows.

Definition 23. Let $\mathcal{M}$ be a PA and q be a state of $\mathcal{M}$. Let S_1, S_2 be subsets of Σ where $C = S_1 \cap S_2 \neq \emptyset$. Let $(q, \sigma)_{\mathtt{P}}$ (resp. $(q, \gamma)_{\mathtt{P}}$) be a position of player $\mathtt{P} \in \{\mathtt{E}, \mathtt{A}\}$ in the two-players game $\mathcal{G}_{S_1}(\mathcal{M}, \lambda)$ (resp. $\mathcal{G}_{S_2}(\mathcal{M}, \lambda)$). Define $(q, \sigma)_{\mathtt{P}} \bowtie_C (q, \gamma)_{\mathtt{P}}$ iff $\sigma \bowtie_C \gamma$.

Let $(\mathcal{M}, \lambda)$ be a symbolic game where $\mathcal{M} = \langle \Sigma, \mathcal{X}, Q, q_0, \delta, F, \kappa \rangle$ is a PA. Let $k = |\mathcal{X}|$. We define C_0 to be the finite set of letters:

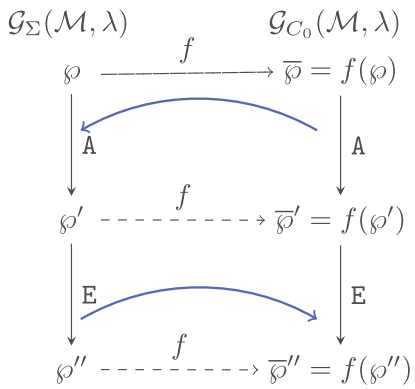
$$C_0 = \Sigma_{\mathcal{A}} \uplus \{c_1, \ldots, c_k\} \tag{2}$$

Let us take the abbreviations $\mathcal{G}_\Sigma = \mathcal{G}_\Sigma(\mathcal{M}, \lambda)$ and $\mathcal{G}_{C_0} = \mathcal{G}_{C_0}(\mathcal{M}, \lambda)$. Now we are ready to show that the games $\mathcal{G}_\Sigma$ and $\mathcal{G}_{C_0}$ are equivalent. For the direction "$\Rightarrow$" we show that out of a winning strategy of `Eloise` in $\mathcal{G}_\Sigma(\mathcal{M}, \lambda)$ we construct a winning strategy for her in $\mathcal{G}_{C_0}(\mathcal{M}, \lambda)$.

For this purpose, we show that each move of `Abelard` in $\mathcal{G}_{C_0}(\mathcal{M}, \lambda)$ can be mapped to an `Abelard` move in $\mathcal{G}_\Sigma(\mathcal{M}, \lambda)$, and that `Eloise` response in $\mathcal{G}_\Sigma(\mathcal{M}, \lambda)$ can be actually mapped to an `Eloise` move in $\mathcal{G}_{C_0}(\mathcal{M}, \lambda)$. Formally, we need to define a function f

$$f : \mathrm{Pos}\big(\mathcal{G}_\Sigma(\mathcal{M}, \lambda)\big) \to \mathrm{Pos}\big(\mathcal{G}_{C_0}(\mathcal{M}, \lambda)\big)$$

in order to make possible this mapping as shown in the Diagram below. It follows that this is sufficient to argue that if there is an infinite play in $\mathcal{G}_\Sigma(\mathcal{M}, \lambda)$ then we can construct an infinite play in $\mathcal{G}_{C_0}(\mathcal{M}, \lambda)$ as well. We show in Lemma 24 that it is possible to construct the function f.



The proof of the direction ($\Leftarrow$) is similar to the one of ($\Rightarrow$) by following the same construction.

Lemma 24. *Let $(\mathcal{M},\lambda)$ be a symbolic game. Let $\mathcal{G}_\Sigma(\mathcal{M},\lambda)$ (resp. $\mathcal{G}_{C_0}(\mathcal{M},\lambda)$) be the concrete game in which the variables are instantiated from the infinite set Σ (resp. finite set C_0 defined in Eq. (2)). Let $\wp^\star$ and $\overline{\wp}^\star$ be their starting position respectively. Then, there is a function*

$$f : \mathrm{Pos}(\mathcal{G}_\Sigma(\mathcal{M},\lambda)) \to \mathrm{Pos}(\mathcal{G}_{C_0}(\mathcal{M},\lambda))$$

with $f(\wp^\star) = \overline{\wp}^\star$ and $\wp \bowtie_C f(\wp)$ for all $\wp \in \mathrm{Pos}(\mathcal{G}_\Sigma(\mathcal{M},\lambda))$, such that the following hold:

i) *for all $\overline{\wp} \in \mathrm{Pos}_{\mathtt{A}}(\mathcal{G}_{C_0}(\mathcal{M},\lambda))$, if $\overline{\wp} \to \overline{\wp}'$ is a move of `Abelard` in $\mathcal{G}_{C_0}(\mathcal{M},\lambda)$ and $f(\wp) = \overline{\wp}$ for some position $\wp$ in $\mathcal{G}_\Sigma(\mathcal{M},\lambda)$ then there exists a position $\wp'$ in $\mathcal{G}_\Sigma(\mathcal{M},\lambda)$ such that the move $\wp \to \wp'$ is possible in $\mathcal{G}_\Sigma(\mathcal{M},\lambda)$ and $f(\wp') = \overline{\wp}'$. And,*
ii) *for all $\wp \in \mathrm{Pos}_{\mathtt{E}}(\mathcal{G}_\Sigma(\mathcal{M},\lambda))$, if $\wp \to \wp'$ is a move of `Eloise` in $\mathcal{G}_\Sigma(\mathcal{M},\lambda)$ then the move $f(\wp) \to f(\wp')$ is possible in $\mathcal{G}_{C_0}(\mathcal{M},\lambda)$.*

Proof. The proof is by induction on n, the number of the moves made in $\mathcal{G}_{C_0}(\mathcal{M},\lambda)$ plus the number of moves made in $\mathcal{G}_\Sigma(\mathcal{M},\lambda)$. The base case, i.e. when $n = 0$, trivially holds since the starting position of $\mathcal{G}_{C_0}(\mathcal{M},\lambda)$ and of $\mathcal{G}_\Sigma(\mathcal{M},\lambda)$ is $((\emptyset, q_0^1), (\emptyset, q_0^2))_{\mathtt{A}}$.

For the induction case let $\wp_n \in \mathrm{Pos}(\mathcal{G}_\Sigma(\mathcal{M},\lambda))$ and $\bar{\wp}_n \in \mathrm{Pos}(\mathcal{G}_{C_0}(\mathcal{M},\lambda))$ where $f(\wp_n) = \bar{\wp}_n$. We discuss two cases depending whether $\bar{\wp}_n$ and $\wp_n$ are both `Abelard` positions or they are both `Eloise` positions.

i) If $\wp_n \in \mathrm{Pos}_{\mathtt{A}}(\mathcal{G}_\Sigma(\mathcal{M},\lambda))$ and $\bar{\wp}_n \in \mathrm{Pos}_{\mathtt{A}}(\mathcal{G}_{C_0}(\mathcal{M},\lambda))$ then consider an `Abelard` move $\bar{m} = \bar{\wp}_n \to \bar{\wp}_{n+1}$ in $\mathcal{G}_{C_0}(\mathcal{M},\lambda)$. In this case $\bar{m}$ is of the form:

$$\bar{m} = \underbrace{(\bar{\sigma}, q)_{\mathtt{A}}}_{\bar{\wp}_n} \to \underbrace{((\bar{\sigma} \uplus \bar{\gamma})_{|D}, q')_{\mathtt{E}}}_{\bar{\wp}_{n+1}}$$

where $q' \in \delta_1(q, \alpha, g)$

and $D = Dom(\bar{\sigma} \uplus \bar{\gamma}) \setminus \kappa^{-1}(q')$

and $\bar{\sigma} \uplus \bar{\gamma} \vdash g$

and $\bar{\gamma} : \mathcal{V}(\bar{\sigma}(g)) \setminus \mathcal{V}(\bar{\sigma}(\alpha)) \to C_0$

From the induction hypothesis we have $\wp_n \bowtie_C f(\wp_n)$, that is, $\wp_n \bowtie_C \bar{\wp}_n$. Hence $\wp_n = (\sigma, q)_{\mathtt{A}}$, for a substitution σ where $\bar{\sigma} \bowtie_C \sigma$. Thus `Abelard` move m in $\mathcal{G}_\Sigma(\mathcal{M},\lambda)$ is defined by:

$$m = \underbrace{(\sigma, q)_{\mathtt{A}}}_{\wp_n} \to \underbrace{((\sigma \uplus \gamma)_{|D}, q')_{\mathtt{E}}}_{\wp_{n+1}}$$

where $\gamma : \mathcal{V}(\sigma(g)) \setminus \mathcal{V}(\alpha) \to \Sigma$ is defined by

$$\gamma = \boldsymbol{\Theta}_C^{C_0,\Sigma}(\bar{\sigma}, \bar{\gamma}, \sigma). \tag{3}$$

Notice that since $\bar{\sigma} \bowtie_C \sigma$ then $dom(\bar{\gamma}) = dom(\gamma)$. We let $f(\wp_{n+1}) = \bar{\wp}_{n+1}$ Furthermore, we must show that

$$\sigma \uplus \gamma \vdash g \tag{4}$$

and that $\wp_{n+1} \bowtie_C f(\wp_{n+1})$, i.e. to show that $\wp_{n+1} \bowtie_C \bar{\wp}_{n+1}$. That is, we must show that:

$$(\sigma \uplus \gamma)_{|D} \bowtie_C (\bar{\sigma} \uplus \bar{\gamma})_{|D} \tag{5}$$

From the definition of γ in Eq. (3) and from Lemma 11) we get:

$$(\sigma \uplus \gamma) \bowtie_C (\bar{\sigma} \uplus \bar{\gamma}) \tag{6}$$

On the one hand, since we already have $\bar{\sigma}_1 \uplus \bar{\gamma} \vdash g_1$, then it follows from Item (8) of Lemma 9 that $\sigma_1 \uplus \gamma \vdash g_1$. Thus Eq. (4) is proved. On the other hand, Eq. (5) follows from Eq. (6) by applying Item (2) of Lemma 9.

ii) If $\wp_n \in \mathrm{Pos}_{\mathrm{E}}(\mathcal{G}_\Sigma(\mathcal{M}, \lambda))$ and $\bar{\wp}_n \in \mathrm{Pos}_{\mathrm{E}}(\mathcal{G}_{C_0}(\mathcal{M}, \lambda))$ then consider an `Eloise` move $m = \wp_n \to \wp_{n+1}$ in $\mathcal{G}_\Sigma(\mathcal{M}, \lambda)$.
In this case m of the form:

$$m = \underbrace{(\sigma, q)_{\mathrm{E}}}_{\wp_n} \to \underbrace{((\sigma \uplus \gamma)_{|D}, q')_{\mathrm{A}}}_{\wp_{n+1}}$$

where $q' \in \delta(q, \beta, g)$

and $D = Dom(\sigma \uplus \gamma) \setminus \kappa^{-1}(q')$

and $\gamma \models \sigma(g)$

and $\gamma : \mathcal{V}(\sigma(\beta)) \cup \mathcal{V}(\sigma(g)) \to \Sigma$

From the induction hypothesis we have that $\wp_n \bowtie f(\wp_n)$, that is, $\wp_n \bowtie_C \bar{\wp}_n$, therefore $\bar{\wp}_n = (\bar{\sigma}, q)_{\mathrm{E}}$, for a substitution $\bar{\sigma}$ such that $\sigma \bowtie_C \bar{\sigma}$.
The corresponding move $\bar{m}$ in $\mathcal{G}_{C_0}(\mathcal{M}, \lambda)$ is defined by:

$$\bar{m} = \underbrace{(\bar{\sigma}, q)_{\mathrm{E}}}_{\bar{\wp}_n} \to \underbrace{((\bar{\sigma} \uplus \bar{\gamma})_{|D}, q')_{\mathrm{A}}}_{\bar{\wp}_{n+1}}$$

where $\bar{\gamma} \models \bar{\sigma}(g)$

and $\bar{\gamma} : \mathcal{V}(\bar{\sigma}(\beta)) \cup \mathcal{V}(\bar{\sigma}(g)) \to C_0$

where $\bar{\gamma}$ is defined by

$$\bar{\gamma} = \boldsymbol{\Theta}_C^{\Sigma,C_0}(\sigma, \gamma, \bar{\sigma})$$

From Eq. (1) we get

$$\sigma \uplus \gamma \bowtie_C \bar{\sigma} \uplus \bar{\gamma}$$

From Item (2) of Lemma 9 it follows that

$$(\sigma \uplus \gamma)_{|D} \bowtie_C (\bar{\sigma} \uplus \bar{\gamma})_{|D}$$

Therefore, $\wp_{n+1} \bowtie_C \bar{\wp}_{n+1}$.

This ends the proof of the lemma. □

Therefore,

Proposition 25. *Let $(\mathcal{M}, \lambda)$ be a symbolic game and let C_0 be the finite set of letters defined in Eq. (2). Then,* `Eloise` *has a winning strategy in $\mathcal{G}_\Sigma(\mathcal{M}, \lambda)$ iff she has a winning strategy in $\mathcal{G}_{C_0}(\mathcal{M}, \lambda)$.*

It follows from Lemma 22 and Proposition 25 that the simulation problem for PAs is decidable. We argue next that this problem is in EXPTIME.

Theorem 26. *The simulation problem for PAs is decidable in EXPTIME.*

Proof. Let $\mathcal{A}_1 = \langle \Sigma, \mathcal{X}_1, Q_1, q_0^1, \delta_1, F_1, \kappa_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, \mathcal{X}_2, Q_2, q_0^2, \delta_2, F_2, \kappa_2 \rangle$ be two PAs. Since the size of the symbolic simulation game of $\mathcal{A}_1$ by $\mathcal{A}_2$ is $O(|Q_1| \times |Q_2| \times |\mathcal{X}_1 \cup \Sigma_{\mathcal{A}_1}|)$, see Definition 21, it is sufficient to argue that solving symbolic games is in APSPACE (alternating polynomial space). Let $(\mathcal{M}, \lambda)$ be a symbolic game and $S \subset \Sigma$ be a finite set of letters. We describe an alternating Turing machine that solves the concrete game $(\mathcal{M}, \lambda, S)$ using polynomial space. The alternating Turing machine stores two pieces of data:

(1) the current position, i.e. a pair composed of a state q of $\mathcal{M}$ and a substitution $\sigma : \mathcal{X}_1 \uplus \mathcal{X}_2 \to S$, and
(2) a counter that stores the number of positions that have been reached so far.

Besides, universal states correspond to `Abelard` positions and existential states correspond to `Eloise` positions. Notice that the number of different positions of $(\mathcal{M}, \lambda, S)$ is bounded exponentially in the size of $\mathcal{M}$. Let p_{max} be such a bound. The machine halts if a deadlock position is reached or the value of the counter is greater or equal to p_{max}. Thus `Eloise` wins iff the machine halts on an `Abelard` position or the value of the counter is greater or equal to p_{max}. Since the number of different positions of $(\mathcal{M}, \lambda, S)$ is bounded exponentially in the size of $\mathcal{M}$ then the size of the counter is $O(\log(p_{max}))$. Therefore the size of the data stored by the machine is polynomial w.r.t. the size of $\mathcal{M}$. □

Finally, we mention that the proof of the decidability of simulation for PAs with ε-transitions can be obtained easily by considering in Lemma 24 that a move by `Abelard` on an ε-transition in the game $\mathcal{G}_{C_0}(\mathcal{M}, \lambda)$ corresponds to a move on the same ε-transition by `Abelard` in the game $\mathcal{G}_\Sigma(\mathcal{M}, \lambda)$; and conversely, a move by `Eloise` on an ε-transition in the game $\mathcal{G}_\Sigma(\mathcal{M}, \lambda)$ corresponds to a move on the same transition by `Eloise` in the game $\mathcal{G}_{C_0}(\mathcal{M}, \lambda)$.

5.3. Synthesis of a mediator

In this subsection we show how to synthesize a mediator (as a PA) allowing the communication between a client C and a community of available services $S_1, \ldots, S_n$ by relying on a winning strategy for `Eloise` in the simulation game $\mathcal{G}_{C_0}(C, S_1 \otimes \ldots \otimes S_n)$, where C_0 is the finite set of letters defined in Eq. (2). It is worth mentioning that it is possible to devise an algorithm that generates all possible mediators. To do this, we need to enumerate all possible winning strategies for `Eloise` in the concrete (simulation) game over a finite alphabet, e.g. by using the algorithm of Jurdzinski (2000), then we turn each winning strategy into a mediator.

Firstly, we define the asynchronous product of PAs which generalizes the asynchronous product of finite automata as given in e.g. Berardi et al. (2008).

Definition 27. Given n PAs $\mathcal{A}_i = \langle \Sigma_i, \mathcal{X}_i, Q_i, q_0^i, \delta_i, F_i, \kappa_i \rangle$, where $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$ for all $i \neq j$, their asynchronous product $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ is a PA: $\langle \Sigma, \mathcal{X}, Q, q_0, \delta, F, \kappa \rangle$, where

- $\Sigma = \bigcup_{i=1,\ldots,n} \Sigma_i$,
- $\mathcal{X} = \bigcup_{i=1,\ldots,n} \mathcal{X}_i$,

Algorithm 1: Composition synthesis of PAs.

input : A client PA $\mathcal{A}_0 = \langle \Sigma, \mathcal{X}, Q, q_0, \delta, F, \kappa \rangle$ and a community of services $\mathcal{A}_1, \ldots, \mathcal{A}_n$
output: A mediator $\mathcal{M}$ as a PA that delegates the actions of $\mathcal{A}_0$ to an appropriate service among the community of services

1 $(\mathcal{A}, \lambda) \leftarrow \texttt{Game}(\mathcal{A}_0, \{\mathcal{A}_1, \ldots \mathcal{A}_n\})$;
2 **Let** $\mathcal{A} = \langle \Sigma, \mathcal{X}, Q, q_0, \delta, F, \kappa \rangle$;
3 $k \leftarrow |\mathcal{X}|$;
4 $C_0 \leftarrow \Sigma_{\mathcal{A}} \cup \{c_1, \ldots, c_k\}$; /* As in Eq. (2) */
5 $G \leftarrow \mathcal{G}_{C_0}(\mathcal{A}, \lambda)$; /* The concrete game as defined in Definition 21 */
6 Compute a winning strategy

$$\rho : \quad \xi_{\mathcal{X}, C_0} \times Q \longrightarrow \xi_{\mathcal{X}, C_0} \times (\xi_{\mathcal{X}, C_0} \times Q)$$

for Eloise in G;
7 $\mathcal{G}_{\text{sym}} \leftarrow (\langle \Sigma, \mathcal{X}, Q, q_0, \delta_{|\rho}, F, \kappa \rangle, \lambda)$; /* with $\delta_{|\rho}$ as defined in Definition 28 */
8 $\mathcal{G}'_{\text{sym}} \leftarrow \mathcal{R}(\rho, \mathcal{G}_{\text{sym}})$; /* with the function $\mathcal{R}$ as defined in Definition 30 */
9 **Let** $\mathcal{G}'_{\text{sym}} = (\mathcal{M}, \lambda)$;

- $Q = Q_1 \times \cdots \times Q_n$,
- $q_0 = q_0^1 \times \cdots \times q_0^n$, $F = F_1 \times \cdots \times F_n$,
- δ is defined by: $\boldsymbol{q} \in \delta(\boldsymbol{p}, \alpha, g)$ iff for some i, $\pi_i(\boldsymbol{q}) \in \delta_i(\pi_i(\boldsymbol{p}), t, g)$, and for all $j \neq i$ we have that $\pi_j(\boldsymbol{q}) = \pi_j(\boldsymbol{p})$, for a label $\alpha \in \Sigma \cup \mathcal{X}$ and a guard g, where π_i denotes the projection along the ith-component, and
- κ is defined by: $\boldsymbol{p} \in \kappa(x)$ iff for some i, $\pi_i(\boldsymbol{p}) \in \kappa_i(x)$.

Secondly, we need to introduce the notion of restriction of the transition function of a symbolic game by a strategy for player Eloise. The idea is to remove from the symbolic game the transitions that Eloise does not follow when she plays according to a given strategy. The formal definition follows.

Definition 28. Let $(\mathcal{A}, \lambda)$ be a symbolic game for a PA $\mathcal{A} = \langle \Sigma, \mathcal{X}, Q, q_0, \delta, F, \kappa \rangle$. Let $S \subseteq \Sigma$ and ρ:

$$\rho : \quad \xi_{\mathcal{X}, S} \times Q \longrightarrow \xi_{\mathcal{X}, S} \times (\xi_{\mathcal{X}, S} \times Q)$$

be a strategy for Eloise in $\mathcal{G}_S(\mathcal{A}, \lambda)$. The restriction of the transition function δ by ρ, denoted by $\delta_{|\rho}$ is defined by:

$$q \xrightarrow{\alpha, g} q' \in \delta_{|\rho} \quad \text{iff} \quad \exists \sigma, \sigma', \gamma \in \xi_{\mathcal{X}, S} \text{ s.t. } \rho(\sigma, q) = (\gamma, (\sigma', q')).$$

Finally, let $(\mathcal{A}, \lambda)$ be a symbolic game, C_0 be the finite set of letters defined in Eq. (2) and ρ be a winning strategy for Eloise in $\mathcal{G}_{C_0}(\mathcal{A}, \lambda)$. Out for these data we shall derive another symbolic game that realizes the strategy ρ. This is achieved by turning the strategy ρ into guards. Roughly speaking, the guards will encode the strategy ρ by telling Eloise which move to make. We describe in Definition 29 how to turn a substitution into guards.

Definition 29. Let σ be a substitution and let $S \subseteq \Sigma$. Let $\mathcal{X}_S = \{x \in dom(\sigma) \mid \sigma(x) \in S\}$ and assume that $\sigma = \sigma_1 \uplus \sigma_2$ where $dom(\sigma_1) = \mathcal{X}_S$. The function $\Omega(\sigma, S)$ is defined by:

$$\Omega(\sigma, S) \stackrel{def}{=} \Omega_1(\sigma_1, S) \wedge \Omega_2(\sigma_2, S)$$

where:

$$\begin{cases} \Omega_1(\sigma_1, S) = \bigwedge_{x_i \in dom(\sigma_1)} (x_i = \sigma_1(x_i)) \quad \text{and,} \\ \Omega_2(\sigma_2, S) = \bigwedge_{\substack{z \in dom(\sigma_2) \\ a \in S}} (z \neq a) \wedge \bigwedge_{\substack{x, y \in dom(\sigma_2) \\ \sigma_2(x) = \sigma_2(y)}} (x = y) \wedge \bigwedge_{\substack{u, t \in dom(\sigma_2) \\ \sigma_2(u) \neq \sigma_2(t)}} (u \neq t) \end{cases}$$

We show in Definition 30 how to realize a strategy in a symbolic game.

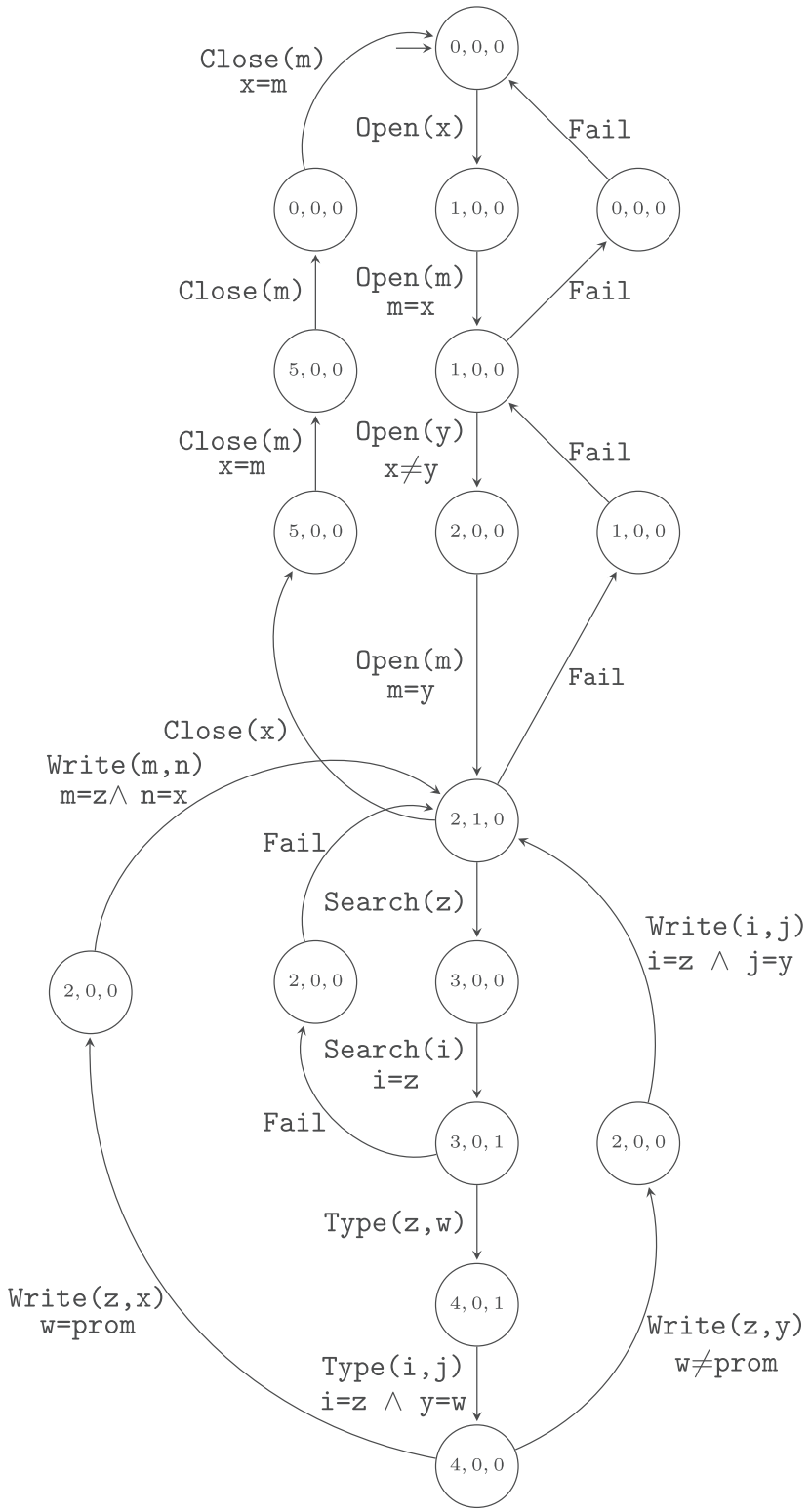


Fig. 4. A synthesized mediator for the PROM example.

Definition 30. Let $(\mathcal{A}, \lambda)$ be a symbolic game where $\mathcal{A} = \langle \Sigma, \mathcal{X}, Q, q_0, \delta, F, \kappa \rangle$ is a PA. Let C_0 be the finite set of letters defined in Eq. (2) and ρ be a strategy for `Eloise` in $\mathcal{G}_{C_0}(\mathcal{A}, \lambda)$. The *realisation* of ρ in $(\mathcal{A}, \lambda)$, denoted by $\mathcal{R}(\rho, (\mathcal{A}, \lambda))$, is the symbolic game $(\langle \Sigma, \mathcal{X}, Q, q_0, \delta', F, \kappa \rangle, \lambda)$ where:

$$\delta' = \bigcup_{\sigma \in \xi_{\mathcal{X}, C_0}} \left\{ q \xrightarrow{x, g \wedge \Omega(\pi_1(\rho(\sigma, q)), \Sigma_{\mathcal{A}})} q' \;\middle|\; q \xrightarrow{x, g} q' \in \delta \text{ and } \lambda(q) = \mathtt{E} \right\}$$

$$\cup \left\{ q \xrightarrow{x,g} q' \;\middle|\; q \xrightarrow{x,g} q' \in \delta \text{ and } \lambda(q) = \mathtt{A} \right\}$$

In Algorithm 1, the function `Game` computes a symbolic simulation of its first argument by the asynchronous product of the PAs in the set of available services given as a second argument as defined in Definition 21. We also use the keyword **Let** to define new variables by pattern-matching.

Step 1 computes $(\mathcal{A}, \lambda)$ the symbolic simulation game of $\mathcal{A}_0$ by $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ as given in Definition 21. Steps 2–6 compute a winning strategy ρ for `Eloise` in the concrete simulation game $\mathcal{G}_{C_0}(\mathcal{A}, \lambda)$ where the variables are instantiated from the finite set of letters C_0 defined in Eq. (2). Step 7 uses the strategy ρ to restrict `Eloise` transitions in the symbolic game $(\mathcal{A}, \lambda)$ yielding the symbolic game $\mathcal{G}_{\text{sym}}$ as given in Definition 28. Step 8 computes the symbolic game $\mathcal{G}'_{\text{sym}}$ which is the realization of the strategy ρ in the symbolic game $\mathcal{G}_{\text{sym}}$ as given in Definition 30. Finally, in Step 9 the desired mediator is the PA $\mathcal{M}$ which is the arena of the symbolic game $\mathcal{G}'_{\text{sym}}$.

5.4. Application to the PROM example

Now we are ready to apply the results developed so far to solve the composition problem for the PROM example. In this example, a winning strategy for `Eloise` can be computed in the game $\mathcal{G}_{\Sigma}(\mathtt{CLIENT}, \mathtt{FILE} \otimes \mathtt{SEARCH})$, and thus the client requests can be satisfied in all cases. By applying the mediator synthesis procedure described in Subsection 5.3 to the PROM example, we get the mediator depicted in Fig. 4. We notice that states of the mediator are abbreviated as a tuple (i, j, k) instead of $((p_i, (q_j, r_k)), \alpha)$, where p_i is a state of `CLIENT`, q_j is a state of `FILE`, r_k is a state of `SEARCH` and α is a label.

Finally, we mention that it is possible to modify the PA of Fig. 3 to get a usage policy specifying that at most two files can be open at the same time. It can be shown that this usage policy is respected by the mediator of the PROM example depicted in Fig. 4.

6. Conclusion

We introduced an extension of automata that provides us with both a natural specification style as shown by examples and a composition synthesis algorithm for parametrized services. To our knowledge PAs is one of the largest class of automata on infinite alphabet to admit a decidable simulation algorithm. In the future we plan to consider security policy enforcement on services in the spirit of Basin et al. (2013), Degano et al. (2012), Bartoletti et al. (2009), Martín et al. (2012), Ranise (2012). A first step in this direction will be to extend the results of De Giacomo et al. (2013) in our setting with an infinite alphabet.

References

Akroun, L., Benatallah, B., Nourine, L., Toumani, F., 2013. On decidability of simulation in data-centric business protocols. In: BPMW'13, vol. 132, pp. 352–363.

Alonso, G., Casati, F., Kuno, H., Machiraju, V., 2004. Web Services: Concepts, Architectures and Applications. Springer-Verlag.

Balbiani, P., Cheikh, F., Feuillade, G., 2010. Controller/orchestrator synthesis via filtration. Electron. Notes Theor. Comput. Sci. 262, 33–48.

Bartoletti, M., Degano, P., Ferrari, G.L., Zunino, R., 2009. Model checking usage policies. In: TGC 2008. In: Lect. Notes Comput. Sci., vol. 5474, pp. 19–35.

Basin, D.A., Jugé, V., Klaedtke, F., Zalinescu, E., 2013. Enforceable security policies revisited. ACM Trans. Inf. Syst. Secur. 16 (1), 3.

Belkhir, W., Chevalier, Y., Rusinowitch, M., 2013. Fresh-variable automata: application to service composition. In: 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. SYNASC. IEEE Computer Society, pp. 153–160.

Berardi, D., Calvanese, D., De Giacomo, G., Hull, R., Mecella, M., 2005. Automatic composition of transition-based semantic web services with messaging. In: Proc. 31st Int. Conf. Very Large Data Bases. VLDB 2005, pp. 613–624.

Berardi, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M., 2003. Automatic composition of E-services that export their behavior. In: ICSOC, pp. 43–58.

Berardi, D., Cheikh, F., Giacomo, G.D., Patrizi, F., 2008. Automatic service composition via simulation. Int. J. Found. Comput. Sci. 19 (2), 429–451.

Castagna, G., Gesbert, N., Padovani, L., 2007. A theory of contracts for web services. In: PLAN-X 2007, Programming Language Technologies for XML. An ACM SIGPLAN Workshop colocated with POPL 2007, Nice, France, January 20, pp. 37–48.

Castagna, G., Gesbert, N., Padovani, L., 2009. A theory of contracts for web services. ACM Trans. Program. Lang. Syst. 31 (5), 19:1–19:61.
Cheikh, F., 2009. Composition de services: algorithmes et complexité. Ph.D. thesis, Université Paul Sabatier, Thèse de doctorat.
De Giacomo, G., Patrizi, F., Sardiña, S., 2013. Automatic behavior composition synthesis. Artif. Intell. 196, 106–142.
Degano, P., Ferrari, G.L., Mezzetti, G., 2012. Nominal automata for resource usage control. In: CIAA, pp. 125–137.
Fearnley, J., Jurdzinski, M., 2013. Reachability in two-clock timed automata is PSPACE-complete. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M.Z., Peleg, D. (Eds.), ICALP (2). In: Lect. Notes Comput. Sci., vol. 7966. Springer, pp. 212–223.
Feuillade, G., Pinchinat, S., 2007. Modal specifications for the control theory of discrete event systems. Discrete Event Dyn. Syst. 17 (2), 211–232.
Finkel, A., Schnoebelen, P., 2001. Well-structured transition systems everywhere! Theor. Comput. Sci. 256 (1–2), 63–92.
Hamadi, R., Benatallah, B., 2003. A Petri net-based model for web service composition. In: Database Technologies 2003, Proceedings of the 14th Australasian Database Conference. ADC 2003, Adelaide, South Australia, February. In: CRPIT. Aust. Comput. Soc., vol. 17, pp. 191–200.
Hassen, R.R., Nourine, L., Toumani, F., 2008. Protocol-based web service composition. In: ICSOC. In: Lect. Notes Comput. Sci., vol. 5364, pp. 38–53.
Hoffmann, J., 2008. Towards efficient belief update for planning-based web service composition. In: 18th European Conference on Artificial Intelligence. ECAI 2008, Patras, Greece, July 21–25, 2008. In: Proc. Front. Artif. Intell. Appl., vol. 178, pp. 558–562.
Hoffmann, J., Bertoli, P., Pistore, M., 2007. Web service composition as planning, revisited: in between background theories and initial state uncertainty. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence. July 22–26, 2007, Vancouver, British Columbia, Canada. AAAI Press, pp. 1013–1018.
Hull, R., Su, J., 2005. Tools for composite web services: a short overview. SIGMOD Rec. 34 (2), 86–95.
Jancar, P., 1995. Undecidability of bisimilarity for Petri nets and some related problems. Theor. Comput. Sci. 148 (2), 281–301.
Jurdzinski, M., 2000. Small progress measures for solving parity games. In: STACS, pp. 290–301.
Kaminski, M., Zeitlin, D., 2010. Finite-memory automata with non-deterministic reassignment. Int. J. Found. Comput. Sci. 21 (5), 741–760.
Lustig, Y., Vardi, M.Y., 2009. Synthesis from component libraries. In: de Alfaro, L. (Ed.), FOSSACS. In: Lect. Notes Comput. Sci., vol. 5504. Springer, pp. 395–409.
Martín, J.A., Martinelli, F., Pimentel, E., 2012. Synthesis of secure adaptors. J. Log. Algebr. Program. 81 (2), 99–126.
Narayanan, S., McIlraith, S., 2002. Simulation, verification and automated composition of web services. In: Proceedings of the Eleventh International World Wide Web Conference. WWW-11, Honolulu, Hawaii, USA, May 7–11, pp. 77–88.
Neven, F., Schwentick, T., Vianu, V., 2004. Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Log. 5 (3), 403–435.
Nourine, L., Toumani, F., 2012. Formal approaches for synthesis of web service business protocols. In: Web Services and Formal Methods – 9th International Workshop. WS-FM 2012, Tallinn, Estonia, September 6–7. In: Revis. Sel. Pap. Lect. Notes Comput. Sci., vol. 7843. Springer, pp. 1–15.
Pathak, J., Basu, S., Honavar, V., 2007. Assembling composite web services from autonomous components. In: Emerging Artificial Intelligence Applications in Computer Engineering. In: Front. Artif. Intell. Appl., vol. 160. IOS Press, pp. 394–405.
Pistore, M., Marconi, A., Bertoli, P., Traverso, P., 2005. Automated composition of web services by planning at the knowledge level. In: IJCAI. USA, pp. 1252–1259.
Ranise, S., 2012. On the verification of security-aware E-services. J. Stat. Comput. 47 (9), 1066–1088.
Reger, G., Barringer, H., Rydeheard, D.E., 2013. A pattern-based approach to parametric specification mining. In: ASE. IEEE, pp. 658–663.
Reisig, W., 2008. Towards a theory of services. In: UNISCON'08, pp. 271–281.
Sakamoto, H., Ikeda, D., 2000. Intractability of decision problems for finite-memory automata. Theor. Comput. Sci. 231 (2), 297–308.
Waldinger, R.J., 2001. Web agents cooperating deductively. In: Proceedings of the First International Workshop on Formal Approaches to Agent-Based Systems-Revised Papers. FAABS'00. Springer-Verlag, London, UK, pp. 250–262.