Parametrized Automata Simulation and Application to Service Composition

Walid Belkhir[1], Yannick Chevalier[2], and Michael Rusinowitch[1]

[1] INRIA Nancy–Grand Est & LORIA
walid.belkhir@inria.fr,rusi@loria.fr,particle.mania@gmail.com
[2] Université Paul Sabatier & IRIT Toulouse ychevali@irit.fr

1 Introduction

This paper summarizes several results that have been published in [3]. 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. Service composition is required when none of the existing services can fulfill some client's 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.

Service composition has been studied in many works e.g. [9,4]. The related problem of system synthesis from libraries of reusable components has been thoroughly investigated too [8,5].

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. [6,9]): given a client and a community of available services, compute a mediator, that can be viewed as a special service for enabling communication between the client and the available services in such a way that each client request is forwarded to an appropriate service.

As usual (e.g. [4]), this problem is reduced to the computation of 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 is known 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 abstracts 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 ([1]). Starting from the approach initiated in [2] 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 [2] we get a 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 labeled 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: their value is reset, and they can be bound later to an arbitrary letter. Refreshing mechanism is particularly useful when computation starts a new sessions or to simulate calls to functions with local variables.

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.

2 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.

2.1 A motivating example

In Fig. 1 we have an e-commerce website 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 [4] the problem reduces to finding 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 labeled by a term, say `write(m,n)`, abbreviates successive transitions labeled 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 [2] since they do not have guards.

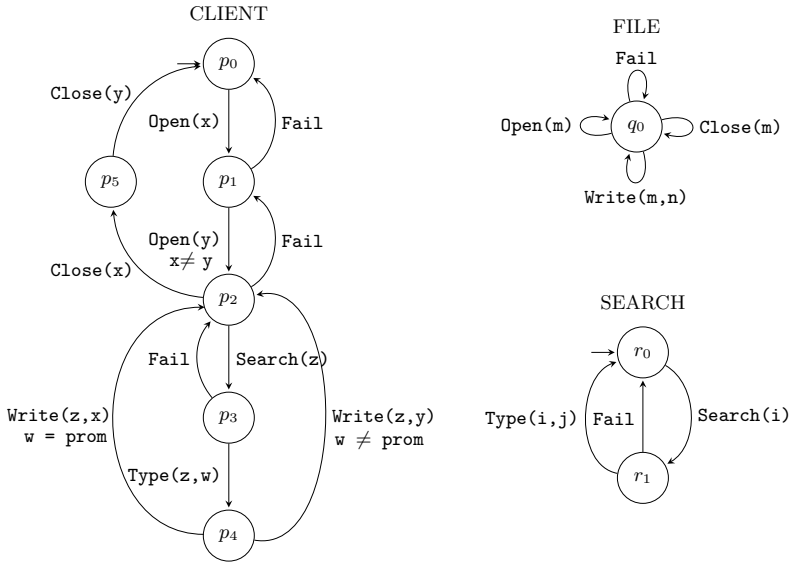

Fig. 1: PROM example.

Before introducing formally the class of PAs, let us first explain the main ideas behind them. The transitions of a PA are labeled with letters or variables ranging over an infinite set of letters. These transitions can also be labeled 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 1. *The set $\mathbb{G}$ of guards over $\Sigma \cup \mathcal{X}$ where Σ is an infinite set of letters, and $\mathcal{X}$ is a finite set of variables, 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.

The formal definition of PAs follows.

Definition 2. *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, $Q_0 \subseteq Q$ 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.*

A 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.

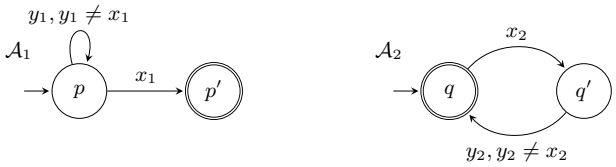

Fig. 2: Two PAs $\mathcal{A}_1$ and $\mathcal{A}_2$ where the variable y_1 is refreshed in the state p, and the variables x_2, y_2 are refreshed in the state q.

Example 1. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the PAs depicted above in Figure 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.

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 binding the variable y_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 binds the variable y_1 to the guessed symbol, then enters the state p.
  3. From the state p, refresh the variable y_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:

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

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 $L(\mathcal{A}_2) = \{w_1 w_1' \cdots w_n w_n' \mid w_i, w_i' \in \Sigma,\ n \geq 1,\ \text{and } w_i \neq w_i',\ \forall i \in [n]\}$.

2.2 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:

Theorem 1. *[3] PAs are closed under union, concatenation, Kleene operator and intersection. They are not closed under complementation.*

For the main decision procedures we have that:

Theorem 2. *[3] For PAs, Membership is NP-complete, Universality and Containment are undecidable. Nonemptiness is PSPACE-complete*

To argue that Nonemptiness is PSPACE, given a PA $\mathcal{A}$, it is sufficient to 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.

To show that the Nonemptiness of PAs is PSPACE-hard, we reduce the reachability problem for bounded one-counter automata [7] (known to be PSPACE-hard) to the Nonemptiness problem of PAs [3].

Theorem 3. *The simulation problem for PAs is decidable in EXPTIME.*

In order to show the decidability of simulation for PAs we 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. 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.

References

1. L. Akroun, B. Benatallah, L. Nourine, and F. Toumani. On decidability of simulation in data-centric business protocols. In *BPMW'13, v.132*, pages 352–363.
2. W. Belkhir, Y. Chevalier, and M. Rusinowitch. Fresh-variable automata: Application to service composition. In *SYNASC'13*, pages 153–160. IEEE C.S.
3. W. Belkhir, Y. Chevalier, and M. Rusinowitch. Parametrized automata simulation and application to service composition. *Journal of Symbolic Computation*, 69:40–60, 2014.
4. D. Berardi, F. Cheikh, G. D. Giacomo, and F. Patrizi. Automatic service composition via simulation. *Int. J. Found. Comput. Sci.*, 19(2):429–451, 2008.
5. S. Bliudze and J. Sifakis. Synthesizing glue operators from glue constraints for the construction of component-based systems. In *Software Composition*, pages 51–67. Springer, 2011.
6. F. Cheikh. *Composition de services: algorithmes et complexité.* PhD thesis, Université Paul Sabatier, Thèse de doctorat. http://thesesups.ups-tlse.fr/712/1/Cheikh_Fahima.pdf, 2009.
7. C. Haase, J. Ouaknine, and J. Worell. On the relationship between reachability problems in timed and counter automata. In *Proc. of* Reachability Problems, 2012.
8. Y. Lustig and M. Y. Vardi. Synthesis from component libraries. In *FOSSACS'09*, pages 395–409, 2009.
9. L. Nourine and F. Toumani. Formal approaches for synthesis of web service business protocols. In *WS-FM'12*, volume 7843 of *LNCS*, pages 1–15. Springer, 2012.