A Parametrized Propositional Dynamic Logic with Application to Service Synthesis

Walid Belkhir

INRIA Nancy–Grand Est & LORIA, France

Gisela Rossi

National University of Córdoba, Argentina

Michael Rusinowitch

INRIA Nancy–Grand Est & LORIA, France

Abstract

We extend propositional dynamic logic (PDL) with variables ranging over an infinite domain. This extension, called *parametrized* PDL or PPDL for short, is interpreted over parametrized transitions systems whose edges are labeled with letters or variables and whose states are labeled with non-parametrized propositions. We show that the satisfiability problem for PPDL is decidable.
We apply these results to the composition problem of web services in presence of constraints on the global ordering of the message-exchange events between the agents. We express the client specification and the available services as parametrized transitions systems and we express the behavioral constraints as a PPDL formula that the generated orchestrator must fulfill. It turns out that the model of such a formula represents the desired orchestrator.

Keywords: parametrised propositional dynamic logic, infinite domain, satisfiability, service synthesis, games.

1 Introduction

The synthesis problem has initially been introduced by Church [8] in the context of digital circuits and amounts to construct from a given specification an automaton satisfying this specification whenever it exists and return a negative answer otherwise. Wolper [23] has considered the synthesis problems for communicating processes. Controller synthesis in Ramadge and Wonham theory of discrete event processes [22] aims to generate supervisors that restrict the behavior of a plant so that a given specification is fulfilled. Synthesis in the context of Service Oriented Computing can be defined as the automated derivation of a specification of how to coordinate some available component services

to fulfill the client requests [7]. In several interesting cases this composition specification can be derived automatically and can be turned into a program that monitors the flow among the component services and the client.

PDL is a logic that was introduced in [12] to reason about programs and was successfully applied in several areas in computer science: program verification, agent-based system specification (e.g. [14]), planning, knowledge representation. It also admits strong connections with Description Logics [3], making it even more interesting. PDL combines two entities: formulas to be interpreted in the nodes of a Kripke structure, and programs to be interpreted by binary relations over the set of nodes of a Kripke structure. PDL is well-adapted to describe transition systems, and the model checking problem for PDL remains PTIME-complete even when the logic is extended by looping and repeat operators [15].

The satisfiability problem for PDL asks whether a given formula has a model and to construct it whenever it exists. One of the applications of the satisfiability problem is the automatic program synthesis that, roughly speaking, consists in the automatic generation of programs out of their specification. When service behaviors can be represented with finite-state transition systems then the composition synthesis problem for Web services can be reduced to PDL satisfiability [10]. In this approach the existence of an orchestrator, that is a transition system that delegates any requested action from the client to one of the available community of services, is expressed with a PDL formula. If this one is satisfiable then from any of its finite models (known to exist) we can extract a transition system that solves the synthesis problem.

However computational models based on finite-state transition systems over finite alphabets (i.e. over finite set of actions) are inefficient and even insufficient to accurately describe systems that need to deal with an arbitrary large amount of data. This observation has motivated many works to introduce and study models over infinite alphabets e.g. [20,19,11,6] since large datasets can be abstracted as infinite domains. On the other hand, these models have not been applied to service composition except in [6] where the composition problem for Web services is proved to be decidable and is reduced to compute a simulation preorder. Such simulation preorder can be turned into an orchestrator that suitably schedules the actions of the available community of services to fulfill the client requests.

Besides, we would like also to specify different interaction modes between the client and the available services, as well as additional constraints on the global ordering of the message-exchange events between the composite services and the client. Among the possible interaction modes between the agents we can mention orchestration, choreography [1,21], and distributed orchestration [2]. An advantage of PDL-based synthesis over the simulation-based synthesis is that, on the one hand, PDL provides a systematic approach for the description of interactions modes between the agents. And, on the other hand, it is possible in this framework to express behavioral constraints (as PDL formulas) that the synthesized composition must fulfill. Such constraints can not be expressed

and taken into account, at least in a straightforward way, in the simulation-based framework. Here are examples of such constraints: preventing data exchange between competing agents (Chinese wall security policies), preventing data update conflicts from different services (critical section paradigm), saving agents' data at the end of a session, closing every open file after usage.

An interesting potential application of PPDL is parametrized system verification by model-checking. Moreover, in some cases PPDL might be more advantageous than PDL even for specifying finite (non-parametrized) systems since PPDL formulas can be exponentially more succinct than their equivalent PDL formulas, leading to better readable formulas. The complexity of PPDL model-checking on parametrized systems is worth studying and requires a separate work.

Contributions. We introduce parametrized transition systems (PTS) and parametrized PDL (PPDL). The transitions of PTS are labeled by variables that can be assigned the read letter. A variable binding can be released at some states: in that case we say that the variable is *refreshed*. This mechanism is natural to express iteration processes, for instance when a service has to scan a list of item identifiers, or sessions. It is useful for real-world applications where service actions are parameterized by terms built with data taken from infinite alphabets (identifiers, codes, addresses ...). Besides, the states are labeled with (non-parametrized) propositional constants, i.e. the propositional constants being true at these states. On the other hand, standard PDL incorporates two entities: formulas and programs, where the programs are regular expressions built over a finite set of atomic actions using concatenation, union and Kleene star. For PPDL we consider *parametrized* regular expressions. In this setting we allow variables in the regular expressions. Such variables range over the infinite set of actions Σ. Besides, in order to free a variable after being bound to an action we shall introduce the *reset* operator $\mathsf{res}(.)$. For instance, the expression $(x; \mathsf{res}(x))^\star$, where $\mathsf{res}(x)$ denotes the resetting of the variable x, stands for all possible finite traces in $\Sigma^\star$ i.e. traces of the form $a_1 a_2 \dots a_n$, where $a_i \in \Sigma$ and $n \in \mathbb{N}$. The PPDL formula $\phi_1 = \forall x.[(x; \mathsf{res}(x))^\star]\mathtt{p}$, where $[-]$ stands for the *necessity* modal operator, $\mathtt{p}$ is a propositional constant and x is a variable, states that $\mathtt{p}$ holds globally, i.e. $\mathtt{p}$ holds in every state of the model. Then, we introduce satisfiability games for PPDL and prove its completeness, then we show the decidability of the satisfiability problem of PPDL. As an application, we show how to use these results to the synthesis of parametrized services.

Related works. Many extensions of PDL were developed e.g. with propositional assignments [4] with intersection operator [5], with converse operators [18], with context-free programs [16]. Model checking algorithms as well as their complexity for PDL with looping, repeat, test intersection, converse operators and context-free programs were developed in [15]. Model checking problems of PDL, and its extension with intersection, over various classes of infinite state systems (basic parallel processes, basic process algebra, pushdown systems, prefix-recognizable systems and Petri nets) were studied in [13]. The

nominal automata that are used for resource usage control in [11] subsumes our parametrized transition systems with refreshing. However model checking technique, rather than satisfiability and synthesis in our case, were considered in this work. Web Services Choreography Description Language (WS-CDL) [1] provides another approach in describing the global ordering of the message-exchange events between the communicating agents. Our proofs are inspired by [17]: they rely on a game-theoretic formulation of satisfiability together with the *focus* mechanism, rather than automata-theoretic techniques. It is shown in [17] how the focus technique solved satisability for the temporal logics `LTL` and `CTL`, and at the same time led to simple completeness proofs.

When service behaviors are represented with finite-state transition systems i.e. they are not data-aware, the composition synthesis problem was reduced to PDL satisfiability in [10]. The composition problem can also be reduced to computing a simulation preorder as in [7]. This approach cannot be extended to the data-centric `Colombo` model for services since 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 or empty database. A theory of contracts that formalizes the compatibility of a client to a service, and the safe replacement of a service with another service has been developed in [9]. Contracts ensures that every possible interaction between compatible clients and services can be completed successfully. Access control policies can be expressed by graphs. We believe that it is possible to translate such graphs into PDL and PPDL and to integrate them in synthesis problems.

Paper organization. The paper is organized as follows. Section 2 introduces parametrized transition systems (PTS) and parametrized PDL (PPDL). Section 3 introduces satisfiability games for PPDL. Section 4 proves the decidability of the satisfiability problem of PPDL. Section 5 applies these results to the synthesis of parametrized services.

Preliminaries. Let $\mathcal{X}$ be a finite set of variables, Σ an infinite set of atomic actions. A substitution 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 $\{a \in \Sigma \mid \exists x \in dom(\sigma) \text{ s.t. } \sigma(x) = a\}$. 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 $F_{|A'}$.

2 Parametrized PDL

In this section we define parametrized propositional dynamic logic (PPDL). Formulas of PPDL are interpreted over *parametrized transitions systems* whose edges are labeled with variables or atomic actions and whose states are labeled with atomic (non-parametrized) propositions. Firstly we introduce the syntax of PPDL and the main ideas behind it, then we introduce parametrized transitions systems and define their traces. Finally, the semantics of PPDL over parametrized transition systems is defined. But first let us illustrate our ideas through a practical example.

A motivating example. Figure 1 represents an e-commerce Web site allowing clients to search and to buy plane tickets with prior authentication. For each action the client performs, the services save the data in a file. The agents in this example are: CLIENT, AUTHENTICATION, FLIGHT, PAYMENT and FILE, they communicate with messages ranging over a possibly infinite set of terms. The problem is to check whether the services AUTHENTICATION, FLIGHT, PAYMENT and FILE can *collaborate* to satisfy the CLIENT requests in presence of certain global constraints expressed by PPDL formulas to be introduced in the following. Services collaborate by exchanging messages before answering a client request. This notion will be formalized by a so-called $\star$-*simulation* relation, which is a variant of the classical simulation preorder. 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. Besides, while composing these agents, there are some requirements that must be fulfilled. We impose that every open file has to be closed and that the flight data of the client have to be stored in an appropriate file. These requirements can be turned into a PPDL formula, Appendix B.2.

Syntax of PPDL. In standard PDL the programs are regular expressions built over a finite set of atomic actions using concatenation, union and Kleene star. For PPDL we consider *parametrized* regular expressions. In this setting we allow variables in the regular expressions. These variables range over the infinite set of actions. Besides, in order to free a variable after being bound to an action we shall introduce the *reset* operator $\mathsf{res}(.)$. For example the expression $x^\star$, where x is a variable, stands for all traces a^* in $\Sigma^\star$, where a is an atomic action in Σ. While the expression $(x; \mathsf{res}(x))^\star$ stands for all possible finite traces in $\Sigma^\star$, where $\mathsf{res}(x)$ denotes the resetting of the variable x, i.e. traces of the form $a_1 a_2 \dots a_n$, where $a_i \in \Sigma$ and $n \in \mathbb{N}$.

Let $\mathbb{P}$ be a finite set of propositional constants containing tt and ff, and Σ an infinite set of atomic actions (or atomic programs), and $\mathcal{X}$ a finite set of variables ranging over Σ. The syntax of PPDL formula is given by the following grammar:

$$\phi ::= [\alpha]\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \mathtt{p} \mid \forall x.\, \phi \mid \neg\phi$$
$$\alpha ::= a \mid x \mid \alpha; \alpha \mid \alpha \cup \alpha \mid \alpha^* \mid \mathsf{res}(x) \mid \phi?$$

where $a \in \Sigma$, $x \in \mathcal{X}$ and $\mathtt{p} \in \mathbb{P}$.

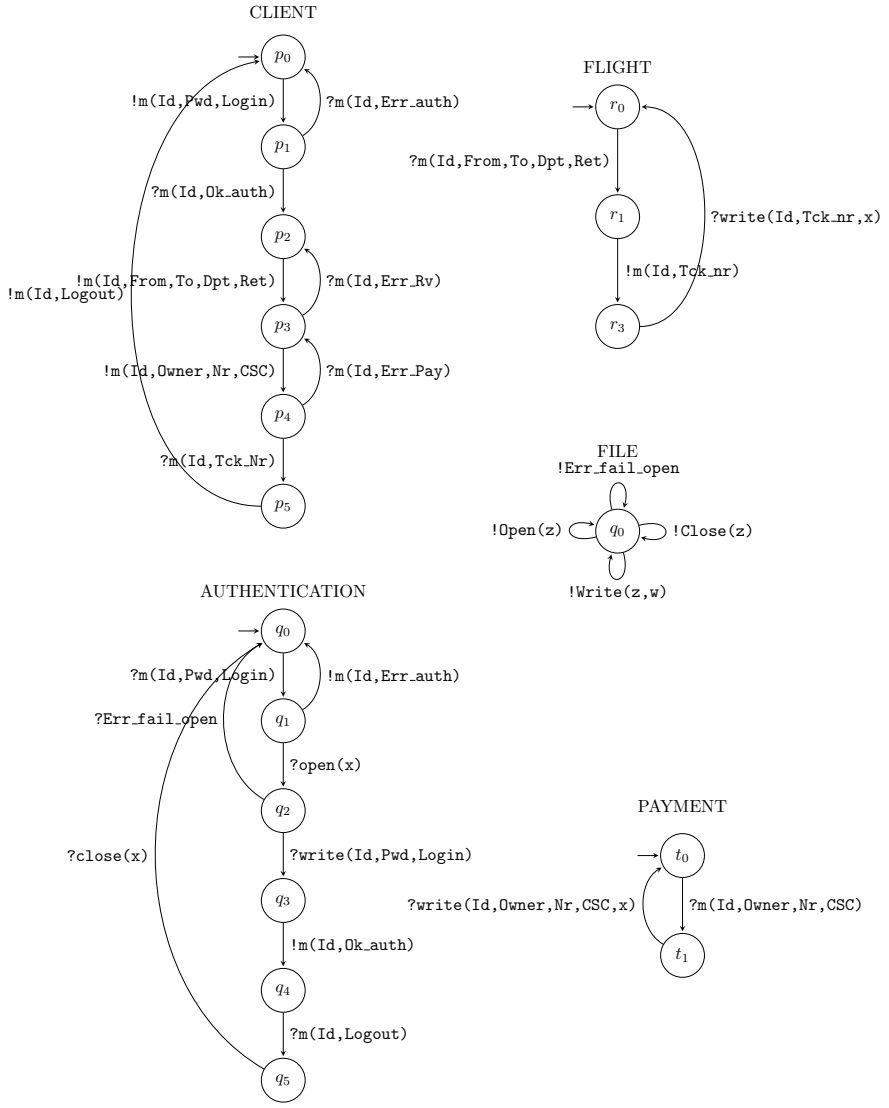

Fig. 1. Flight reservation example where "!" (resp. "?") stands for sending (resp. receiving) a message.

The diamond operator $\langle\cdot\rangle$ (resp. existential quantifier $\exists$) can be defined in terms of the box operator $[\cdot]$ (resp. universal quantifier $\forall$) in the standard way as follows: $\langle\alpha\rangle\phi \stackrel{def}{=} \neg(\neg[\alpha]\neg\phi)$ and $\exists x.\phi \stackrel{def}{=} \neg(\forall x.\neg\phi)$.

For a formula ϕ, we define the finite set of atomic actions appearing in ϕ, denoted by $\Sigma(\phi)$, inductively as follows: $\Sigma([\alpha]\psi) = \Sigma(\alpha) \cup \Sigma(\psi)$, $\Sigma(\psi_1 \vee \psi_2) = \Sigma(\psi_1 \wedge \psi_2) = \Sigma(\psi_1) \cup \Sigma(\psi_2)$, $\Sigma(\exists x.\psi) = \Sigma(\psi?) = \Sigma(\neg\psi) = \Sigma(\psi)$, $\Sigma(\alpha^\star) = \Sigma(\alpha)$, $\Sigma(\alpha_1;\alpha_2) = \Sigma(\alpha_1 \cup \alpha_2) = \Sigma(\alpha_1) \cup \Sigma(\alpha_2)$, $\Sigma(a) = \{a\}$ where

$a \in \Sigma$, and $\Sigma(x) = \Sigma(\mathsf{res}(x)) = \Sigma(\mathtt{p}) = \emptyset$ where $x \in \mathcal{X}$ and $\mathtt{p} \in \mathbb{P}$. For an occurrence x of a variable in a formula, we let $\tau(x) = \exists$ (resp. $\tau(x) = \forall$) if x is existentially (resp. universally) quantified. If λ is a formula or a program, we shall denote by $\mathtt{Res}(\lambda)$ the set of variables being reset in λ. Throughout this paper, formulas are presented in positive form (i.e. the negation only operates on propositional constants in $\mathbb{P}$). This is possible since De Morgan laws can be proved.

Parametrized transition systems. PPDL formulas are interpreted over *parametrized transition systems* (PTS). Before introducing them formally, let us first explain the main ideas behind them. The transitions of a parametrized transition system are labeled with actions or variables. We have a finite number of variables ranging over an infinite set of actions. On a transition labeled with a variable x and an input action l, if x is not bound then taking the transition amounts to binding x to l. On the other hand if x is already bound then the transition can be taken only if x is already bound to l. Since we would like to reuse variables, we add an additional mechanism which will free the variables depending on the states of the automaton. That is, some variables are refreshed in some states, i.e. variables can be freed in these states so that new actions can be assigned to them. The formal definition follows.

Definition 2.1 A parametrized transition system (PTS for short) is a tuple $\mathcal{M} = \langle \Sigma, \mathcal{X}, Q, q_0, \delta, \pi, \kappa \rangle$ where:

- Σ is a infinite set of actions, $\mathcal{X}$ is a finite set of variables,
- Q is a finite set of states, $q_0 \in Q$ is a the initial state,
- $\delta : Q \times (\Sigma_{\mathcal{A}} \cup \mathcal{X}) \to 2^Q$ is a transition function where $\Sigma_{\mathcal{A}}$ is a finite subset of Σ,
- $\pi : Q \to 2^{\mathbb{P}}$ assigns truth values to each propositional constant in $\mathbb{P}$ for each state, and
- $\kappa : \mathcal{X} \to 2^Q$ is the refreshing function that associates to every variable the (possibly empty) set of states where it is refreshed.

We shall denote by $\Sigma_{\mathcal{A}}$ the finite subset of actions from Σ appearing in the PTS $\mathcal{A}$. For a refreshing function $\kappa : \mathcal{X} \to 2^Q$, we define the function $\kappa^{-1} : Q \to 2^{\mathcal{X}}$ by $\kappa^{-1}(q) = \{x \mid q \in \kappa(x)\}$. A LTS is a tuple $(\Sigma, S, s_0, \Delta, \Pi)$ where S is a (possibly infinite) set of states, $s_0 \in S$, $\Delta : S \times \Sigma \to 2^S$ and $\Pi : S \to 2^{\mathbb{P}}$.
The formal definition of configurations and trace for PTSs.

Definition 2.2 Let $\mathcal{A} = \langle \Sigma, \mathcal{X}, Q, q_0, \delta, \pi, \kappa \rangle$ be a PTS. A *configuration* is a pair (q, γ) where $q \in Q$ and γ is a substitution. We define a transition relation over the configurations as follows: $(q_1, \gamma_1) \xrightarrow{a} (q_2, \gamma_2)$, where $a \in \Sigma$, iff there exists a substitution σ such that $dom(\sigma) \cap dom(\gamma_1) = \emptyset$ and there exists a label $\alpha \in \Sigma \cup \mathcal{X}$ such that $q_2 \in \delta(q_1, \alpha, g)$, $(\gamma_1 \uplus \sigma)(\alpha) = a$ and $\gamma_2 = (\gamma_1 \uplus \sigma)_{|D}$, with $D = Dom(\gamma_1 \uplus \sigma) \setminus \kappa^{-1}(q_2)$. A trace of a PTS is a sequence $a_1 a_2 \dots a_n$ such that there exist states q_i and substitutions σ_i, $i = 1, \dots, n$ such that $(q_0, \emptyset) \xrightarrow{a_1} (q_1, \sigma_1) \dots \xrightarrow{a_n} (q_n, \sigma_n)$.

Example 2.3 Let $\mathcal{A}$ and $\mathcal{A}'$ be the PTS depicted in Figure 2 where the variable x is refreshed in q_0 and the variable z is refreshed in q'_0, q'_1 and q'_2.

The behavior of $\mathcal{A}$ is as follows. Being in the initial state q_0:

- Makes the transition $q_0 \to q_1$ by making an action and bounding the variable x to it, then enters the state q_1,
- Makes the transition $q_1 \to q_0$ by making an action that equals to the value of x, then enters the state q_0,
- From the state q_0, refresh the variable x, that is, it is no longer bound to the input symbol. Then, start again.

We illustrate the run of $\mathcal{A}$ on the trace $w = aabb$, starting from the initial configuration $(\emptyset, q_0)$ as follows:

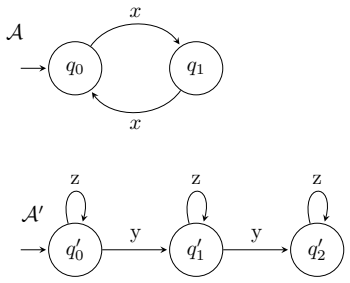

Fig. 2: Two PTS $\mathcal{A}$ and $\mathcal{A}'$ where the variable x is refreshed in q_0 and the variable z is refreshed in q'_0, q'_1 and q'_2.

$$(\emptyset, q_0) \xrightarrow{a} (\{x \mapsto a\}, q_1) \xrightarrow{a} (\emptyset, q_0) \xrightarrow{b} (\{x \mapsto b\}, q_1) \xrightarrow{b} (\emptyset, q_0) \xrightarrow{c} (\{x_1 \mapsto c\}, p').$$

We next define the instantiation of a PTS, it consists in instantiating its variables with all possible actions in Σ, yielding a system with possibly infinite number of states and transitions.

Definition 2.4 [Instantiation of a PTS] Let $\mathcal{M} = \langle \Sigma, \mathcal{X}, Q, q_0, \delta, \pi, \kappa \rangle$ be a PTS. The *instantiation* of $\mathcal{M}$, denoted by $\mathcal{C}(\mathcal{M})$, is the LTS $(\Sigma, S, s_0, \Delta, \Pi)$, where:

$$\begin{aligned} S &= Q \times \xi_{\mathcal{X},\Sigma}, \\ s_0 &= (q_0, \emptyset), \\ (q', \sigma') \in \Delta(a, (q, \sigma)) \text{ iff } (q, \sigma) &\xrightarrow{a} (q', \sigma'), \text{ and} \\ \Pi((q, \sigma)) &= \pi(q), \quad \text{for all } \sigma \in \xi_{\mathcal{X},\Sigma} \text{ and } q \in Q. \end{aligned}$$

Semantics of PPDL over parametrized transition systems. PPDL formulas are interpreted over *configurations* of parametrized transition systems, or equivalently over the states of the instantiation of PTSs. That is, given a structure $\mathcal{M} = \langle \Sigma, \mathcal{X}, Q, q_0, \delta, \pi, \kappa \rangle$. The interpretation of a formula ϕ over the LTS which is the instantiation of $\mathcal{M}$: $\mathcal{C}(\mathcal{M}) = (\Sigma, S, s_0, \Delta, \Pi)$, will be denoted by $[\![\phi]\!]_{\mathcal{M}}$, or simply $[\![\phi]\!]$, such that $[\![\phi]\!] \subseteq S$, if ϕ is a formula; and $[\![\alpha]\!] \subseteq S \times S$,

if α is a program, is defined as follows:

$$\begin{aligned}
[\![\mathsf{ff}]\!] &= \emptyset \\
[\![\mathtt{p}]\!] &= \Pi^{-1}(\mathtt{p}), \quad \text{where } \mathtt{p} \in \mathbb{P} \\
[\![\psi_1 \wedge \psi_2]\!] &= [\![\psi_1]\!] \cap [\![\psi_2]\!] \\
[\![\psi_1 \vee \psi_2]\!] &= [\![\psi_1]\!] \cup [\![\psi_2]\!] \\
[\![[\alpha]\psi]\!] &= \{s \mid \forall s' \text{ if } (s,s') \in [\![\alpha]\!] \text{ then } \\
&\qquad s' \in [\![\psi]\!]\}, \text{ if } \alpha \notin \mathcal{X} \\
[\![[x]\psi]\!] &= [\![\forall x.[x]\psi]\!], \text{ if } x \in \mathcal{X} \\
[\![\forall x.\psi]\!] &= \bigcup_{a \in \Sigma} [\![\psi[x := a]]\!],
\end{aligned}$$

$$\begin{aligned}
[\![\neg\psi]\!] &= S \setminus [\![\psi]\!] \\
[\![\alpha;\beta]\!] &= [\![\alpha]\!] \circ [\![\beta]\!] \\
[\![\alpha \cup \beta]\!] &= [\![\alpha]\!] \cup [\![\beta]\!] \\
[\![\alpha^\star]\!] &= [\![\alpha]\!]^\star = \bigcup_{n \geq 0} [\![\alpha]\!]^n \\
[\![\mathsf{res}(x)]\!] &= \mathtt{Id} \\
[\![a]\!] &= \{(s,s') | s' \in \Delta(a,s)\}, \\
&\qquad \text{if } a \in \Sigma,
\end{aligned}$$

where $\phi[x := a]$ stands for the application of the substitution $\{x \mapsto a\}$ to ϕ. It is inductively defined as follows:

$$\begin{aligned}
\mathtt{p}[x := a] &= \mathtt{p}, \quad \text{if } \mathtt{p} \in \mathbb{P} \\
(\psi_1 \wedge \psi_2)[x := a] &= \psi_1[x := a] \wedge \psi_2[x := a] \\
([\alpha]\psi)[x := a] &= ([\alpha[x := a]]\psi[x := a]) \\
(\forall y.\psi)[x := a] &= (\forall y.\psi[x := a]), \text{ if } x \neq y \\
(\neg\psi)[x := a] &= \neg(\psi[x := a]) \\
(\alpha \cup \beta)[x := a] &= (\alpha[x := a] \cup \beta[x := a]), \\
\alpha^\star[x := a] &= (\alpha[x := a])^\star \\
(\mathtt{res}(y))[x := a] &= \mathtt{res}(y),
\end{aligned}$$

$$(\alpha;\beta)[x := a] = \begin{cases} (\alpha[x := a];\beta), \text{ if } x \in \mathtt{Res}(\alpha) \\ (\alpha[x := a];\beta[x := a]), \text{ other.} \end{cases}$$

$$\beta[x := a] = \begin{cases} a \text{ if } \beta = x \\ \beta \text{ if } \beta \in \Sigma \cup \mathcal{X} \text{ and } \beta \neq x \end{cases}$$

A formula ϕ is *satisfiable* if there is a PTS $\mathcal{M}$ together with its instantiation $\mathcal{C}(\mathcal{M}) = (\Sigma, S, s_0, \Delta, \Pi)$ and a state $s \in S$, s.t. $s \in [\![\phi]\!]_{\mathcal{M}}$. The formula ϕ is said to be valid, denoted $\models \phi$, if it is true in every state of every parametrized transition system. Notice that $\not\models \phi$ iff $\neg\phi$ is satisfiable.

Example 2.5 Firstly, we give some PPDL formula to illustrate the combination of the quantifiers with the modalities. Let $\phi_1 = \forall x.[x]\phi'_1$, $\phi_2 = \exists x.[x]\phi'_2$, $\phi_3 = \forall x.\langle x\rangle\phi'_3$, and $\phi_4 = \exists x.\langle x\rangle\phi'_4$. where ϕ'_i are PPDL formula. The formula ϕ_1 holds in a state q iff for every instantiation of the variable x, say with an action $a \in \Sigma$, each transition outgoing from q and labeled with a yields a state where $\phi'_1[x := a]$ holds. The formula ϕ_2 holds in a state q iff there exists an instantiation of the variable x, say with an action $a \in \Sigma$, such that each transition outgoing from q and labeled with a yields a state where $\phi'_2[x := a]$ holds. The formula ϕ_3 holds in an state q iff for every instantiation of the variable x, say with an action $a \in \Sigma$, there exists a transition outgoing from q and labeled with a that yields a state where $\phi'_3[x := a]$ holds. The formula ϕ_4 holds in a state q iff there exists an instantiation of the variable x, say with an action $a \in \Sigma$, such that there exists a transition outgoing from q and labeled with a that yields a state where $\phi'_4[x := a]$ holds.

Secondly, we give some examples of useful properties. The property "there is a transition that is labeled with the action a and that can be reached from

the current state" can be expressed by the PPDL formula $\exists x.\langle(x; \mathsf{res}(x))^\star\rangle\langle a\rangle\mathtt{tt}$. The formula $\langle((\exists x.\langle x\rangle\mathtt{tt})?)^\star\rangle\mathtt{p}$ states that either $\mathtt{p}$ holds in the current state, or there exist transitions for which $\mathtt{p}$ holds in the outgoing states.

□

It is worth mentioning that if the alphabet is finite, the three properties above can be expressed by PDL formulas whose size depends on the size of the (finite) set of actions. However this is not the case if they are expressed by PPDL formulas (in which the variables range over a finite set of actions).

3 Satisfiability games

A satisfiability game $\mathcal{G}_S(\phi)$ on a PPDL formula ϕ where the variables are instantiated from the set of actions $S \subseteq \Sigma$, is an infinite duration game played between two players: player I (or Abelard) and player II (or Eloise). Player II is looking to prove that ϕ is satisfiable and player I is trying to prove that it is not satisfiable. Besides, the game positions are configurations (i.e. a pair composed of a state and a substitution). The name of the rules is indicated on the left of the inference rule; and the name of the player is indicated on the right.

$$\mathtt{R}_\wedge\colon \frac{\Big[(\phi_0 \wedge \phi_1, \sigma)\Big], \Gamma}{\Big[(\phi_i, \sigma)\Big], (\phi_{1-i}, \sigma), \Gamma}\ \mathrm{I} \qquad \mathtt{R}_\vee\colon \frac{\Big[(\phi_0 \vee \phi_1, \sigma)\Big], \Gamma}{\Big[(\phi_i, \sigma)\Big], \Gamma}\ \mathrm{II}$$

The rules of the formulas starting with modalities follow:

$$\mathtt{R}_1\colon \frac{\Big[(\langle\alpha_0 \cup \alpha_1\rangle\phi, \sigma)\Big], \Gamma}{\Big[(\langle\alpha_i\rangle\phi, \sigma)\Big], \Gamma}\ \mathrm{II} \qquad \mathtt{R}_2\colon \frac{\Big[([\alpha_0 \cup \alpha_1]\phi, \sigma)\Big], \Gamma}{\Big[([\alpha_i]\phi, \sigma)\Big], ([\alpha_{1-i}]\phi, \sigma), \Gamma}\ \mathrm{I}$$

$$\mathtt{R}_3\colon \frac{\Big[(\langle\alpha_0; \alpha_1\rangle\phi, \sigma)\Big], \Gamma}{\Big[(\langle\alpha_0\rangle\langle\alpha_1\rangle\phi, \sigma)\Big], \Gamma} \qquad \mathtt{R}_4\colon \frac{\Big[([\alpha_0; \alpha_1]\phi, \sigma)\Big], \Gamma}{\Big[([\alpha_0][\alpha_1]\phi, \sigma)\Big], \Gamma}$$

$$\mathtt{R}_5\colon \frac{\Big[(\langle\alpha^\star\rangle\phi, \sigma)\Big], \Gamma}{\Big[(\phi, \sigma) \vee (\langle\alpha\rangle\langle\alpha^\star\rangle\phi, \sigma)\Big], \Gamma} \qquad \mathtt{R}_6\colon \frac{\Big[([\alpha^\star]\phi, \sigma)\Big], \Gamma}{\Big[(\phi, \sigma) \wedge ([\alpha][\alpha^\star]\phi, \sigma)\Big], \Gamma}$$

The rules for the reset operator and the test operator follow:

$$\mathtt{R}^1_\mathtt{r}\colon \frac{\Big[([\mathsf{res}(x); \alpha]\phi, \sigma)\Big], \Gamma}{\Big[([\alpha]\phi, \sigma_{|Dom(\sigma)\setminus\{x\}}\Big], \Gamma} \qquad \mathtt{R}^2_\mathtt{r}\colon \frac{\Big[(\langle\mathsf{res}(x); \alpha\rangle\phi, \sigma)\Big], \Gamma}{\Big[(\langle\alpha\rangle\phi, \sigma_{|Dom(\sigma)\setminus\{x\}}\Big], \Gamma}$$

The rules for free variables appearing in a modality:

$$\mathtt{R}^1_x\colon \frac{\Big[([x]\phi,\sigma)\Big],\Gamma \quad \text{if } \tau(x)=\forall}{\Big[([x]\phi,\sigma\uplus\{x\mapsto a\}\Big],\Gamma}\ \mathrm{I} \qquad \mathtt{R}^2_x\colon \frac{\Big[([x]\phi,\sigma)\Big],\Gamma \quad \text{if } \tau(x)=\exists}{\Big[([x]\phi,\sigma\uplus\{x\mapsto a\}\Big],\Gamma}\ \mathrm{II}$$

$$\mathtt{R}^1_?\colon \frac{\Big[\langle\psi?\rangle\phi,\sigma)\Big],\Gamma}{(\psi,\sigma),\Big[(\phi,\sigma)\Big],\Gamma} \qquad \mathtt{R}^2_?\colon \frac{\Big[[\psi?]\phi,\sigma)\Big],\Gamma}{\Big[(\neg\psi,\sigma)\vee(\phi,\sigma)\Big],\Gamma}$$

The rules for the universally and existentially quantified formula follow:

$$\mathtt{R}_\forall\colon \frac{\Big[(\forall x\,\phi,\sigma)\Big],\Gamma}{\Big[(\phi,\sigma_{|Dom(\sigma)\setminus\{x\}}\uplus[x\mapsto a])\Big],\Gamma}\ \mathrm{I} \qquad \mathtt{R}_\exists\colon \frac{\Big[(\exists x\,\phi,\sigma)\Big],\Gamma}{\Big[(\phi,\sigma_{|Dom(\sigma)\setminus\{x\}}\uplus[x\mapsto a])\Big],\Gamma}\ \mathrm{II}$$

The successive applications of the above rules might yield a configuration in which all formulas are either propositional constants or of the form $(\langle\alpha\rangle\phi,\sigma)$ or $([\alpha]\phi,\sigma)$ where $\alpha\in\Sigma\cup\mathcal{X}$ and $\sigma(\alpha)\in\Sigma$.

$$\mathrm{X}_1\colon \frac{\Big[(\langle\alpha_1\rangle\phi_1,\sigma_1)\Big],\ldots,(\langle\alpha_n\rangle\phi_n,\sigma_n),([\beta_1]\psi_1,\gamma_1),\ldots,([\beta_m]\psi_m,\gamma_m),\ldots,p_1,\ldots,p_l}{\Big[(\phi_1,\sigma_1)\Big],(\psi_{j_1},\gamma_{j_1}),\ldots,(\psi_{j_q},\gamma_{j_q})}$$

where $\alpha_i,\beta_i\in\Sigma\cup\mathcal{X}$ and $\forall i=1,\ldots,q:\sigma_1(\alpha_1)=\gamma_{j_i}(\beta_{j_i}), j_i\in\{1,\ldots,m\}$ and $\sigma_l(\alpha_l),\gamma_{l'}(\beta_{l'})\in\Sigma$ for all l,l'.

$$\mathrm{X}_2\colon \frac{(\langle\alpha_1\rangle\phi_1,\sigma_1),\ldots,(\langle\alpha_n\rangle\phi_n,\sigma_n),\Big[([\beta_1]\psi_1,\gamma_1)\Big],\ldots,([\beta_m]\psi_m,\gamma_m),\ldots,p_1,\ldots,p_l}{(\phi_k,\sigma_k),\Big[(\psi_{j_1},\gamma_{j_1})\Big],\ldots,(\psi_{j_q},\gamma_{j_q})}\ \mathrm{I}$$

where $\alpha_i,\beta_i\in\Sigma\cup\mathcal{X}$ and $\forall i=1,\ldots,q:\sigma_k(\alpha_k)=\gamma_{j_i}(\beta_{j_i}), j_i\in\{1,\ldots,m\}$, and $\sigma_l(\alpha_l),\gamma_{l'}(\beta_{l'})\in\Sigma$ for all l,l'.

Moreover, there is a rule allowing player I to change his mind w.r.t. to the focus:

$$\mathrm{FC}\colon \frac{\Big[(\phi,\sigma)\Big],(\psi,\gamma),\Gamma}{(\phi,\sigma),\Big[(\psi,\gamma)\Big],\Gamma}\ \mathrm{I}$$

We notice that the main difference with the satisfiability games for standard PDL is that our games have (possibly) an infinite number of positions and

infinite branching. This is due to the fact that the size of our configurations is unbounded. Thus we modify accordingly the winning conditions of to deal with this new setting as follows.

Firstly, the winning conditions have to deal with the fact that a least fixed-point construct is fulfilled and there is no contradiction in the propositional constants. Player I wins the (possibly infinite) play $\pi = C_0, \ldots, C_n, \ldots$ iff

(i) $C_m = \Big[(q, \sigma)\Big], \Gamma$ and ($q = \mathsf{ff}$ or $\bar{q} \in \Gamma$), for some m, or

(ii) The formula $\langle \alpha^\star \rangle \phi$ appears infinitely often under the focus in π and player I has applied the focus rule (FC) only a finite number of times. That is, there exists an infinite sequence $i_1, i_2, \ldots$ of integers such that $C_{i_j} = \Big[(\langle \alpha^\star \rangle \phi, \sigma_{i_j})\Big], \Gamma_{i_j}$ for all $j = 1, 2, \ldots$ and there exists some i_k such that no focus rule is applied from the configurations C_m for all $m \geq i_k$.

Player II wins the (possibly infinite) play $\pi = C_0, \ldots, C_n, \ldots$ if

(iii) $C_n = \Big[(q_1, \sigma_1)\Big], \ldots, (q_k, \sigma_k)$ and $\{q_1, \ldots, q_k\}$ is satisfiable, where q_i are propositional constants, or

(iv) The formula $[\alpha^\star]\phi$ appears infinitely often in π under the focus, that is, there exists an infinite sequence $i_1, i_2, \ldots$ of integers such that $C_{i_j} = \Big[([\alpha^\star]\phi, \sigma_{i_j})\Big], \Gamma_{i_j}$ appears in π, or

(v) The formula ϕ appears infinitely often in π under the focus and Player I has applied the focus rule (FC) infinitely often.

One can argue that these winning conditions are mutually exclusive, hence:

Lemma 3.1 *The game $\mathcal{G}_S(\phi)$ has a unique winner, where $S \subseteq \Sigma$.*

Now we will prove that the game theoretic characterization of PPDL is sound and complete.

Theorem 3.2 *The following hold:*

(Soundness). If player II wins the game $\mathcal{G}(\Phi_0)$ then Φ_0 is satisfiable.

(Completeness). If Φ_0 is satisfiable then player II wins the game $\mathcal{G}(\Phi_0)$.

The proof of the completeness relies on the following Lemma:

Lemma 3.3 *We have that $\phi \wedge \langle \alpha \rangle \psi$ is satisfiable iff $\phi \wedge (\psi \vee \langle \alpha \rangle (\langle \alpha^\star \rangle \psi \vee \neg \phi))$ is satisfiable.*

4 Decidability of PPDL

The idea of the proof of the decidability of PPDL relies on reducing the satisfiability problem of a PPDL formula into the satisfiability problem of the same formula in which the actions range over a *finite* set of actions. It turns out that solving the resulting game is decidable since it is a finite 2-players game with Büchi winning conditions. The construction of this finite set of actions follows.

Definition 4.1 Let ϕ be a PPDL formula and let $\Sigma(\phi) = \{a_1, \ldots, a_n\}$ and $\mathcal{X} = \{x_1, \ldots, x_k\}$. Define $\mathcal{G}_C(\phi)$ to be the satisfiability game in which the variables are instantiated from the finite set of constants C:

$$C = \{a_1, \ldots, a_n, c_1, \ldots, c_k\} \tag{1}$$

The idea is that the game $\mathcal{G}_C(\phi)$ is used to simulate the game $\mathcal{G}_\Sigma(\phi)$. Before showing that, we need to introduce a variant of satisfiability games in which Player II can duplicate a formula under the focus a finite number of times.

Definition 4.2 Define $\mathcal{G}_S^{\mathcal{D}}$ to be satisfiability game in which we add the duplication rule for player II:

$$\text{DP: } \frac{\Big[(\phi,\sigma)\Big], \Gamma}{\Big[(\phi,\sigma)\Big], (\phi,\sigma), \Gamma}\ \text{II}$$

that has to be applied a finite number of times.

The main result of this section follows, as well as the structure of the proof:

Theorem 4.3 *Satisfiability for PPDL is decidable.*

On the one hand, Corollary 4.8 shows that $\mathcal{G}_\Sigma$ and $\mathcal{G}_\Sigma^{\mathcal{D}}$ are equivalent. On the other hand, Lemma 4.4 shows that $\mathcal{G}_\Sigma$ and $\mathcal{G}_C^{\mathcal{D}}$ are equivalent, and hence $\mathcal{G}_C$ and $\mathcal{G}_C^{\mathcal{D}}$ are equivalent as well. It follows that $\mathcal{G}_C$ and $\mathcal{G}_\Sigma$ are equivalent. The equivalence must be understood as player II wins in one game iff she wins in the other game.

Lemma 4.4 *The games $\mathcal{G}_\Sigma$ and $\mathcal{G}_\Sigma^{\mathcal{D}}$ are equivalent. That is, Player II wins in $\mathcal{G}_\Sigma(\phi)$ iff she wins in $\mathcal{G}_\Sigma^{\mathcal{D}}(\phi)$.*

In order to relate the configurations of $\mathcal{G}_C(\phi)$ to the ones of $\mathcal{G}_\Sigma(\phi)$, we define a relation $\lhd$ between the configurations of $\mathcal{G}_C(\phi)$ and the configurations of $\mathcal{G}_\Sigma(\phi)$. We shall denote by Ψ the (finite) set of formulas appearing in a configuration, i.e. $\Psi((\psi,\sigma)) = \{\psi\}$ and $\Psi([C_1], \ldots, C_n) = \bigcup_i \Psi(C_i)$. Firstly, we define the *coherence* relation between substitutions.

Definition 4.5 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:

(i) $dom(\bar{\sigma}) = dom(\sigma)$,

(ii) If $\bar{\sigma}(x) \in C$ then $\bar{\sigma}(x) = \sigma(x)$, and if $\sigma(x) \in C$, then $\bar{\sigma}(x) = \sigma(x)$, for any variable $x \in dom(\sigma)$, and

(iii) for any variables $x, y \in dom(\sigma)$, $\bar{\sigma}(x) = \bar{\sigma}(y)$ iff $\sigma(x) = \sigma(y)$.

In order to relate the configurations of $\mathcal{G}_\Sigma$ and $\mathcal{G}_C$, where C is the set of letters defined in Eq (1), we define next a binary relation, denoted by $\lhd$, between configurations.

Definition 4.6 Let ϕ be a PPDL formula with $\Sigma(\phi) = \{a_1, \ldots, a_n\}$. Let Γ (resp. $\widehat{\Gamma}$) be a list of configurations in $\mathcal{G}_\Sigma(\phi)$ (resp. $\mathcal{G}_C(\phi)$) of the form: $\Gamma = (\psi_1, \sigma_1), \ldots, (\psi_m, \sigma_m), \cdots$, and $\widehat{\Gamma} = (\widehat{\psi}_1, \widehat{\sigma}_1), \ldots, (\widehat{\psi}_m, \widehat{\sigma}_m)$, where ψ_i and $\widehat{\psi}_i$ are PPDL formulas, and σ_i and $\widehat{\sigma}_i$ are substitutions. Let f be a total surjective function from the set of configurations of $\mathcal{G}_\Sigma(\phi)$ to the set of configurations of $\mathcal{G}_C^{\mathcal{D}}(\phi)$. We define $\Gamma \lhd_f^{\Sigma,C} \widehat{\Gamma}$ iff the following hold:

(i) $\Psi(\Gamma) = \Psi(\widehat{\Gamma})$.

(ii) If $f((\psi_i, \sigma_i)) = (\widehat{\psi}_j, \widehat{\sigma}_j)$ then $\psi_i = \widehat{\psi}_j$ and $\sigma_i \bowtie_C \widehat{\sigma}_j$.

(iii) If $(\widehat{\psi}, \widehat{\sigma})$ is under the focus in $\widehat{\Gamma}$ and (ψ, σ) is under the focus in Γ, then $f((\psi, \sigma)) = (\widehat{\psi}, \widehat{\sigma})$.

Besides, we write $\mathcal{G}_\Sigma(\phi) \lhd \mathcal{G}_C(\phi)$ iff there exists a surjective function f such that $\left[\phi\right], \emptyset \lhd_f^{\Sigma,C} \left[\phi\right], \emptyset$.

In what follows we shall write $\lhd_f$ instead of $\lhd_f^{\Sigma,C}$ if there is no ambiguity. The following Lemma shows that the inference rules of section 3 preserve the relation $\lhd$:

Lemma 4.7 *Let ϕ be a PPDL formula with the finite set of actions $C = \{a_1, \ldots, a_n, c_1, \ldots, c_k\}$ as defined in Eq. (1). Let Γ (resp. $\widehat{\Gamma}$) be a list of configurations in $\mathcal{G}_\Sigma(\phi)$ (resp. $\mathcal{G}_C^{\mathcal{D}}(\phi)$). If $\Gamma \lhd_f \widehat{\Gamma}$ then*

(i) *for every Γ' such that $\Gamma \xrightarrow{I} \Gamma'$ is a move in $\mathcal{G}_\Sigma(\phi)$, there exists a total surjective function f' and a list of configurations $\widehat{\Gamma}'$ such that $\widehat{\Gamma} \xrightarrow{I} \widehat{\Gamma}'$ is a possible move in $\mathcal{G}_C^{\mathcal{D}}(\phi)$ and $\Gamma' \lhd_{f'} \widehat{\Gamma}'$, and*

(ii) *for every $\widehat{\Gamma}'$ such that $\widehat{\Gamma} \xrightarrow{II} \widehat{\Gamma}'$ is a move in $\mathcal{G}_C^{\mathcal{D}}(\phi)$, there exists a total surjective function f' and a list of configurations Γ' such that $\Gamma \xrightarrow{II} \Gamma'$ is a possible move in $\mathcal{G}_\Sigma(\phi)$ and $\Gamma' \lhd_{f'} \widehat{\Gamma}'$,*

Corollary 4.8 *Let ϕ be a PPDL formula. If $\mathcal{G}_\Sigma(\phi) \lhd \mathcal{G}_C(\phi)$ then Player II wins in $\mathcal{G}_\Sigma(\phi)$ iff she wins in $\mathcal{G}_C(\phi)$.*

5 Application to service composition

We can formulate service composition problem under policy constraints as a PPDL satisfiability problem as in [10]. The client and services are firstly specified by parametrized systems which can be translated into PPDL formulas.

Modeling services collaboration with $\star$-simulation. In previous works [7,6] a simulation relation is used to express that a combination of services jointly satisfies a client. In this setting a client request is met by an elementary action of a single service. A $\star$-simulation relation is a variant of the previous simulation relation that allows the services to communicate with each others before answering the client request. Its formal definition is in Annex B.

Composition synthesis as a PPDL satisfiability problem. Web-service composition via $\star$-simulation in presence of policy constraints can be expressed

as a satisfiability problem of a PPDL formula by following the same construction as [10] for PDL. Let $S = (S_1, ..., S_n)$ be a community of available services over the shared actions Σ. Each available service S_i is represented by a parametrized transition system TS_i. Let TS_0 be the client specification. The detailed construction of the formula is given in Annex B.

Extraction of a mediator. It is possible to extract a model, as a PTS, for a satisfiable PPDL formula out of a winning strategy ρ for player II in the game $\mathcal{G}_C(\phi)$. We firstly take the sub-game $(\mathcal{G}_C(\phi))_{|\rho}$ induced by the strategy ρ. Then, we turn the game $(\mathcal{G}_C(\phi))_{|\rho}$ into a symbolic game in which the moves are labeled with variables, i.e. the transitions labeled with variables correspond to the moves in which the rules X1 and X2 are applied; the remaining transitions are ε-transitions and correspond to the other rules (i.e. $R_\vee$, $R_\wedge$, R_1, R_2, etc).

6 Conclusion

We have introduced an extension of PDL, called PPDL, in which the actions can be letters or variables ranging over an infinite domain. We have proved that the satisfiability problem of PPDL is decidable when it is interpreted over the subclass of parametrized transition systems in which the variables can be refreshed. As an application, we have shown how to formulate services composition problem of parametrized services as a synthesis problem of PPDL.

References

[1] *Web service choreography description language (WS-CDL)*, www.w3.org/TR/ws-cdl-10.

[2] Avanesov, T., Y. Chevalier, M. A. Mekki, M. Rusinowitch and M. Turuani, *Distributed orchestration of Web services under security constraints*, in: *DPM'11*, pp. 235–252.

[3] Baader, F., D. Calvanese, D. McGuinness, D. Nardi and P. Patel-Schneider, "The Description Logic Handbook: Theory, Implementation and Applications," CUP, 2003.

[4] Balbiani, P., A. Herzig and N. Troquard, *Dynamic logic of propositional assignments: A well-behaved variant of PDL*, in: *LICS* (2013), pp. 143–152.

[5] Balbiani, P. and D. Vakarelov, *PDL with intersection of programs: A complete axiomatization*, Journal of Applied Non-Classical Logics **13** (2003), pp. 231–276.

[6] Belkhir, W., Y. Chevalier and M. Rusinowitch, *Fresh-variable automata: Application to service composition*, in: *15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC* (2013), pp. 153–160.

[7] Berardi, D., F. Cheikh, G. D. Giacomo and F. Patrizi, *Automatic service composition via simulation*, Int. J. Found. Comput. Sci. **19** (2008), pp. 429–451.

[8] Büchi, J. R. and L. H. Landweber, *Solving sequential conditions by finite-state strategies*, Transactions of the American Mathematical Society **138** (1969), pp. 295–311.

[9] Castagna, G., N. Gesbert and L. Padovani, *A theory of contracts for web services*, ACM Trans. Program. Lang. Syst. **31** (2009), pp. 19:1–19:61.

[10] Cheikh, F., G. D. Giacomo and M. Mecella, *Automatic web services composition in trustaware communities*, in: *SWS*, 2006, pp. 43–52.

[11] Degano, P., G. L. Ferrari and G. Mezzetti, *Nominal automata for resource usage control*, in: *CIAA*, 2012, pp. 125–137.

[12] Fischer, M. J. and R. E. Ladner, *Propositional dynamic logic of regular programs*, J. Comput. Syst. Sci. **18** (1979), pp. 194–211.

[13] Göller, S. and M. Lohrey, *Infinite state model-checking of propositional dynamic logics*, in: Z. Ésik, editor, *CSL*, Lecture Notes in Computer Science **4207** (2006), pp. 349–364.

[14] Herzig, A., E. Lorini, F. Moisan and N. Troquard, *A dynamic logic of normative systems*, in: *IJCAI 2011*, 2011, pp. 228–233.
[15] Lange, M., *Model checking propositional dynamic logic with all extras*, Journal of Applied Logic **4** (2006), pp. 39 – 49.
[16] Lange, M. and R. Somla, *Propositional dynamic logic of context-free programs and fixpoint logic with chop*, Inf. Process. Lett. **100** (2006), pp. 72–75.
[17] Lange, M. and C. Stirling, *Focus games for satisfiability and completeness of temporal logic*, in: *LICS*, 2001, pp. 357–365.
[18] Lutz, C., *PDL with intersection and converse is decidable.*, in: C.-H. L. Ong, editor, *CSL*, Lecture Notes in Computer Science **3634** (2005), pp. 413–427.
[19] Manuel, A. and R. Ramanujam, *Automata over infinite alphabets*, World Scientific Review **9** (2011), pp. 329–363.
[20] Neven, F., T. Schwentick and V. Vianu, *Finite state machines for strings over infinite alphabets*, ACM Trans. Comput. Log. **5** (2004), pp. 403–435.
[21] Peltz, C., *Web services orchestration and choreography*, Computer **36** (2003), pp. 46–52.
[22] Ramadge, P. J. and W. M. Wonham, *Supervisory control of a class of discrete event processes*, SIAM J. Control Optim. **25** (1987), pp. 206–230.
[23] Wolper, P., *Specification and synthesis of communicating processes using an Extended Temporal Logic*, in: *POPL* (1982), pp. 20–33.

Appendix

A Proofs for Section 4

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

Remark A.1 Let $C \subseteq \Sigma$ be a finite set of letters, $\bar{\sigma}$ and σ two substitutions, x a variable, and a a letter in C. The following hold.

(i) If $\bar{\sigma} \bowtie_C \sigma$ then $|codom(\bar{\sigma})| = |codom(\sigma)|$ and $\bar{\sigma}_{|D} \bowtie_C \sigma_{|D}$, where $D \subseteq Dom(\sigma)$.

(ii) Consequently, 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 $i = 1, 2$.

Lemma A.2 *[6] Let $C_1 \subseteq \Sigma$ and $C_2 \subseteq \Sigma$ be two sets of actions. Let $C = C_1 \cap C_2$ be s.t. $|C| > |\mathcal{X}|$. Let a_1 be an action in C_1 and $x \in \mathcal{X}$ and let $\sigma_i : \mathcal{X} \to C_i$, $i = 1, 2$ be two substitutions where $\sigma_1 \bowtie_C \sigma_2$. Then, there exists a function $\mathbf{\Theta}^{C_1,C_2}$ satisfying $\sigma_1 \uplus \{x \mapsto a_1\} \bowtie_C \mathbf{\Theta}(\sigma_1, x, a_1, \sigma)$.*

Lemma A.3 (i.e. Lemma 4.7) *Let ϕ be a PPDL formula with the finite set of actions $C = \{a_1, \ldots, a_n, c_1, \ldots, c_k\}$ as defined in Eq. (1). Let Γ (resp. $\widehat{\Gamma}$) be a list of configurations in $\mathcal{G}_\Sigma(\phi)$ (resp. $\mathcal{G}^{\mathcal{D}}_C(\phi)$). If $\Gamma \lhd_f \widehat{\Gamma}$ then*

(i) *for every Γ' such that $\Gamma \xrightarrow{I} \Gamma'$ is a move in $\mathcal{G}_\Sigma(\phi)$, there exists a total surjective function f' and a list of configurations $\widehat{\Gamma}'$ such that $\widehat{\Gamma} \xrightarrow{I} \widehat{\Gamma}'$ is a possible move in $\mathcal{G}^{\mathcal{D}}_C(\phi)$ and $\Gamma' \lhd_{f'} \widehat{\Gamma}'$, and*

(ii) *for every $\widehat{\Gamma}'$ such that $\widehat{\Gamma} \xrightarrow{II} \widehat{\Gamma}'$ is a move in $\mathcal{G}^{\mathcal{D}}_C(\phi)$, there exists a total surjective function f' and a list of configurations Γ' such that $\Gamma \xrightarrow{II} \Gamma'$ is a possible move in $\mathcal{G}_\Sigma(\phi)$ and $\Gamma' \lhd_{f'} \widehat{\Gamma}'$,*

Proof. Assume $\mathcal{X} = \{x_1, \ldots, x_k\}$. We discuss many cases depending on the applied rule.

(i) The applied rules here for player I can be $\mathtt{R}_\wedge$, $\mathtt{R}_2$, $\mathtt{R}_3$, $\mathtt{R}_4$, $\mathtt{R}_5$, $\mathtt{R}_6$, $\mathtt{R}^1_\intercal$, $\mathtt{R}^2_\intercal$, $\mathtt{R}^1_x$, $\mathtt{R}^1_?$, $\mathtt{R}^2_?$, $\mathtt{R}_\forall$, X_1, X_2 and FC. For the rule $\mathtt{R}_\wedge$, Let:

$$\begin{cases} \Gamma = \Big[(\phi_0 \wedge \phi_1, \sigma)\Big], \Upsilon & \text{and} \\ \Gamma' = \Big[(\phi_i, \sigma)\Big], (\phi_{1-i}, \sigma), \Upsilon & \text{and} \\ \widehat{\Gamma} = \Big[(\phi_0 \wedge \phi_1, \widehat{\sigma})\Big], \widehat{\Upsilon} \end{cases}$$

where $\sigma \bowtie_{\Sigma(\phi)} \widehat{\sigma}$. In this case we let

$$\begin{cases} \widehat{\Gamma}' \stackrel{def}{=} \Big[(\phi_i, \widehat{\sigma})\Big], (\phi_{1-i}, \widehat{\sigma}), \widehat{\Upsilon} \\ \sigma \stackrel{def}{=} \widehat{\sigma}, \text{ and} \\ f' \stackrel{def}{=} f_{|\Gamma} \cup \{\big((\phi_i, \sigma), (\phi_i, \widehat{\sigma})\big)\} \cup \{\big((\phi_{1-i}, \sigma), (\phi_{1-i}, \widehat{\sigma})\big)\} \end{cases}$$

Thus, $\Gamma \lhd_{f'} \widehat{\Gamma}'$.

The rules $\mathtt{R}_2$, $\mathtt{R}_3$, $\mathtt{R}_4$, $\mathtt{R}_5$, $\mathtt{R}_6$, $\mathtt{R}^1_?$ and $\mathtt{R}^2_?$ can be handled similarly.

For the rules $\mathtt{R}^i_{\mathtt{r}}$, $i = 1, 2$ the claim follows from the fact that if $\sigma \bowtie_{\Sigma(\phi)} \widehat{\sigma}$ then $\sigma_{|Dom(\sigma)\setminus\{x\}} \bowtie_{\Sigma(\phi)} \widehat{\sigma}_{|Dom(\widehat{\sigma})\setminus\{x\}}$, see Item 1 of remark A.1.

For the rule $\mathtt{R}_\forall$, assume that

$$\begin{cases} \Gamma &= \Big[(\forall x\phi, \sigma)\Big], \Lambda \quad \text{and} \\ \Gamma' &= \Big[(\phi, \sigma_{x\mapsto a})\Big], \Lambda \quad \text{and} \\ \widehat{\Gamma} &= \Big[(\forall x\phi, \widehat{\sigma})\Big], \widehat{\Lambda} \end{cases}$$

We distinguish two cases

Case 1. If $\nexists C \in \Lambda$ s.t. $f(C) = (\forall x\phi, \widehat{\sigma})$, then in this case the related move in $\widehat{\mathcal{G}}_{\mathcal{D}}$ is

$$\underbrace{\Big[(\forall x\phi, \widehat{\sigma})\Big], \widehat{\Lambda}}_{\widehat{\Gamma}} \xrightarrow{(\mathtt{R}_\forall)} \underbrace{\Big[(\phi, \widehat{\sigma}')\Big], \widehat{\Lambda}}_{\widehat{\Gamma}'}$$

where the substitution $\widehat{\sigma}'$ is defined by $\widehat{\sigma}' \stackrel{def}{=} \mathbf{\Theta}^{\Sigma,\Sigma_f}(\sigma, x, a, \widehat{\sigma})$.

Case 2. If $\exists C \in \Lambda$ s.t. $f(C) = (\forall x\phi, \widehat{\sigma})$, then in this case the related moves in $\widehat{\mathcal{G}}_{\mathcal{D}}$ are

$$\underbrace{\Big[(\forall x\phi, \widehat{\sigma})\Big], \widehat{\Lambda}}_{\widehat{\Gamma}} \xrightarrow{(\mathtt{DP})} \Big[(\forall x\phi, \widehat{\sigma})\Big], (\forall x\phi, \widehat{\sigma}), \widehat{\Lambda} \xrightarrow{(\mathtt{R}_\forall)} \underbrace{\Big[(\phi, \widehat{\sigma}')\Big], (\forall x\phi, \widehat{\sigma}), \widehat{\Lambda}}_{\widehat{\Gamma}'}$$

where $\widehat{\sigma}'$ is defined by $\widehat{\sigma}' \stackrel{def}{=} \mathbf{\Theta}^{\Sigma,\Sigma_f}(\sigma, x, a, \widehat{\sigma})$.

Notice that in both cases we have that $\widehat{\sigma}' \bowtie_{\Sigma(\phi)} \sigma$, Lemma A.2. Besides, in both cases the function f' is defined by $f' = f_{|Dom(f)\setminus\{(\forall x\phi,\sigma)\}}$, and $f'((\phi, \sigma_{x\mapsto a})) = (\phi, \widehat{\sigma}')$. Thus $\Gamma \lhd_{f'} \widehat{\Gamma}'$.

For the rule X1, assume that

$$\begin{cases} \Gamma &= \Big[(\langle\alpha_1\rangle\phi_1, \sigma_1)\Big], \ldots, (\langle\alpha_n\rangle\phi_n, \sigma_n), ([\beta_1]\psi_1, \gamma_1), \ldots, \\ &\quad ([\beta_m]\psi_m, \gamma_m), \ldots, p_1, \ldots, p_l \\ \text{and} & \\ \widehat{\Gamma} &= \Big[(\langle\widehat{\alpha}_1\rangle\widehat{\phi}_1, \widehat{\sigma}_1)\Big], \ldots, (\langle\widehat{\alpha}_k\rangle\widehat{\phi}_k, \widehat{\sigma}_k), ([\widehat{\beta}_1]\widehat{\psi}_1, \widehat{\gamma}_1), \ldots, \\ &\quad ([\widehat{\beta}_r]\widehat{\psi}_r, \widehat{\gamma}_r), \ldots, p_1, \ldots, p_{l'} \end{cases}$$

Hence,

$$\begin{cases} \Gamma' &= \Big[(\phi_1, \sigma_1)\Big], (\psi_{j_1}, \gamma_{j_1}), \ldots, (\psi_{j_q}, \gamma_{j_q}) \quad \text{and} \\ \widehat{\Gamma}' &= \Big[(\widehat{\phi}_1, \widehat{\sigma}_1)\Big], (\widehat{\psi}_{j'_1}, \widehat{\gamma}_{j'_1}), \ldots, (\widehat{\psi}_{j'_p}, \widehat{\gamma}_{j'_p}) \end{cases}$$

where, on the one hand, $\alpha_i, \beta_i \in \Sigma \cup \mathcal{X}$ and $\forall i = 1, \ldots, q : \sigma_1(a_1) = \gamma_{j_i}(b_{j_i}), j_i \in \{1, \ldots, m\}$ and, on the other hand, $\widehat{\alpha}_i, \widehat{\beta}_i \in \Sigma \cup \mathcal{X}$ and $\forall i = 1, \ldots, p : \widehat{\sigma}_1(\widehat{\alpha}_1) = \widehat{\gamma}_{j'_i}(\widehat{\beta}_{j'_i}), j'_i \in \{1, \ldots, r\}$. Finally, we let $f' \stackrel{def}{=} f_{|\Gamma'}$. Therefore, $\Gamma' \lhd_{f'} \widehat{\Gamma}'$.

For the rule FC, assume that

$$\begin{cases} \Gamma &= \Big[(\phi, \sigma)\Big], (\psi, \gamma), \Upsilon \quad \text{and} \\ \Gamma' &= \Big[(\psi, \gamma)\Big], (\phi, \sigma), \Upsilon \quad \text{and} \\ \widehat{\Gamma} &= \Big[(\widehat{\phi}, \widehat{\sigma})\Big], \widehat{\Upsilon} \end{cases}$$

The idea is to choose a formula $(\widehat{\psi}, \widehat{\gamma})$ from Υ such that $\widehat{\Gamma}' = \Big[(\widehat{\psi}, \widehat{\gamma})\Big], \widehat{\Upsilon}$. We define $(\widehat{\psi}, \widehat{\gamma}) := f((\psi, \gamma))$. Thus we have $\Gamma' \lhd_f \widehat{\Gamma}'$ since $\widehat{\psi} = \psi$ and $\widehat{\gamma} \bowtie \gamma$.

(ii) The possible rules for player II are $\mathtt{R}_\exists$, X_1 and $\mathtt{R}^2_x$. The rule $\mathtt{R}_\exists$ (resp. $\mathtt{R}^2_x$) is exactly like the rule $\mathtt{R}_\forall$ (resp. $\mathtt{R}^1_x$) apart that player II who moves instead of Player I.

□

B Proofs and definitions for Section 5

B.1 Modeling services collaboration with $\star$-simulation

Definition B.1 Let $\mathcal{A}_i = \mathcal{M} = \langle \Sigma, \mathcal{X}_i, Q^i, q^i_0, \delta_i, \pi_i, \kappa_i \rangle$, $i = 1, 2$ be two parametrized labeled transition systems. A $\star$-simulation is a relation $\trianglelefteq \subseteq (Q_1 \times \zeta_{\mathcal{X}_1,\Sigma}) \times (Q_2 \times \zeta_{\mathcal{X}_2,\Sigma})$ such that:

- $(q^1_0, \emptyset) \trianglelefteq (q^2_0, \emptyset)$.
- If $(q_1, \sigma_1) \trianglelefteq (q_2, \sigma_2)$ and if $(q_1, \sigma_1) \stackrel{a}{\rightarrow} (q'_1, \sigma'_1)$ for some action $a \in \Sigma$, then there exist states $q^0_2, \ldots, q^n, p^0_2, \ldots, p^m_2 \in Q_2$ and substitutions $\sigma^0_2, \ldots, \sigma^n, \gamma^0_2, \ldots, \gamma^m_2$ and actions $a_0, \ldots, a_{n-1}, b_0, \ldots, b_{m-1} \in \Sigma$ such that

$$\begin{aligned} &(q_2, \sigma_2) \stackrel{a_0}{\rightarrow} (q^0_2, \sigma^0_2) \stackrel{a_0}{\rightarrow} (q^1_2, \sigma^1_2) \stackrel{\star}{\rightarrow} \ldots \stackrel{a_{n-1}}{\rightarrow} (q^n_2, \sigma^n_2) \stackrel{a_{n-1}}{\rightarrow} (q^n_2, \sigma^n_2) \\ &\stackrel{a}{\rightarrow} (q'_2, \sigma'_2) \\ &\stackrel{b_0}{\rightarrow} (p^0_2, \gamma^0_2) \stackrel{b_0}{\rightarrow} (p^1_2, \gamma^1_2) \stackrel{\star}{\rightarrow} \ldots \stackrel{b_{m-1}}{\rightarrow} (p^m_2, \gamma^m_2) \stackrel{b_{m-1}}{\rightarrow} (p^m_2, \gamma^m_2) \end{aligned}$$

and $(\sigma'_1, q'_1) \trianglelefteq (p^m_2, \gamma^m_2)$.

B.2 Composition synthesis as a PPDL satisfiability problem

Let $S = (S_1, ..., S_n)$ be a community of available services over the shared actions Σ. Each variable services S_i is represented by a parametrized transition system $TS_i = \langle \Sigma, \mathcal{X}_i, S_i, s_{i0}, \delta_i, \pi_i, \kappa_i \rangle$ defined as above. Let $TS_0 = \langle \Sigma, \mathcal{X}_0, S_0, s_{00}, \delta_0, \pi_0, \kappa_0 \rangle$ be the client specification.

Then we build a PPDL formula Φ to be checked for satisfiability as follows. As propositional constants, we have:

- One propositional constant s for each $i \in \{0, 1, ..., n\}$ and each state s of TS_i, which intuitively denotes that TS_i is in a final state.
- Propositional constants $exec_{ix}$, for $i \in \{0, 1, ..., n\}$ and $x \in \mathcal{A}$, denoting that x will be executed next by the available service S_i.
- One propositional constant *undef* denoting that we are in an "illegal" situation, where the orchestrator program can be left undefined.

For representing the transitions of each available service S_i, we construct a formula Φ_i as the conjunction of:

- $\forall x(s \rightarrow \bigwedge_{(s',x)\in\varepsilon}(\langle x\rangle s') \wedge [x](\bigvee_{(s',x)\in\varepsilon} s'))$. Where $\varepsilon = \{(s', x)|(s, x, s') \in \delta_i\}$, for each s of S_i and $x \in \mathcal{A}$.
- $\forall x(s \wedge exec_{ix} \rightarrow [x]\mathbf{false})$, for each s of S_i such that for no g and s' we have that $(s, g, x, s') \in \delta_i$.
- $\forall x(s \wedge exec_{ix} \rightarrow [x]s)$ for each s of S_i and $x \in \mathcal{A}$.

In addition, we have the formula Φ_{add} obtained as the conjunction of:

- $s \rightarrow \neg s'$ for all pairs of states s, s' of S_i, and for $i \in \{0, 1, ..., n\}$.
- $F_i \leftrightarrow \bigvee_{s\in F_i} s$, for $i \in \{0, 1, ..., n\}$.
- $\forall x(undef \rightarrow [x]undef)$, for $x \in \mathcal{A}$.
- $\forall x(\neg undef \rightarrow \langle x\rangle\mathbf{true} \rightarrow \bigvee_{i\in\{1,...,n\}} exec_{ix})$, for $x \in \mathcal{A}$.
- $\forall x(exec_{ix} \rightarrow \neg exec_{jx})$, for each $i, j \in \{1, ..., n\}, i \neq j$ and each $a \in \mathcal{A}$
- $F_0 \rightarrow \bigvee_{i\in\{1,...,n\}} F_i$.

The requirements that every open file has to be closed and that the flight data of the client have to be stored in an appropriate file can be respectively expressed by the two PPDL formula ψ_1 and ψ_1 as follows:

$$\begin{cases} \psi_1 & = \forall x \forall y[(\mathsf{res}(x); x)^\star; \mathtt{Open(x)}]\big(\exists z\langle(\mathsf{res}(z); z)^\star\rangle\langle\mathtt{Close(x)}\rangle\mathtt{tt}\big), \text{ and} \\ \psi_2 & = \forall x \forall y[(\mathsf{res}(x); x)^\star; \mathtt{!m(Id,Owner,Nbr,CSC}] \\ & \qquad\qquad \big(\exists f\ \exists z\langle(\mathsf{res}(z); z)^\star\rangle\langle\mathtt{Write(m(Id,Owner,Nbr,CSC,\ f)}\rangle\mathtt{tt}\big) \end{cases}$$

Finally, we describe Φ as

$$Init \wedge \forall z.\,[\mathbf{u}](\Phi_0 \wedge \bigwedge_{i\in\{1,...,n\}} \Phi_i \wedge \Phi_{add}) \wedge \psi_1 \wedge \psi_2,$$

where *Init* stands for $s_{00} \wedge s_{10} \wedge ... \wedge s_{n0}$ and represents the initial state of all services S_i (including the target) and $(\mathbf{u} = (z; \mathsf{res}(z))^*)$, which is used to force $(\Phi_0 \wedge \bigwedge_{i\in\{1,...,n\}} \Phi_i \wedge \Phi_{add})$ to be true in every point of the model.