Parametrized automata simulation and application to service composition

Walid Belkhir[1]
Y. Chevalier[2] , M. Rusinowitch[1]

[1]Loria, CASSIS
[2]Université Paul Sabatier & IRIT Toulouse

ICE Workshop, 2015

Automata over infinite alphabets:

- Web services: exchanging data over infinite domain
- XML documents: nodes carrying data over infinite domain
- Program verification: variables, time

Challenges:

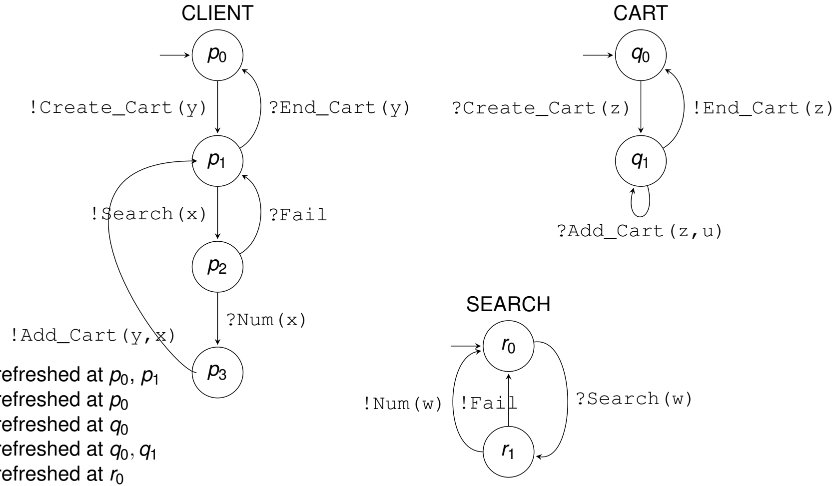
- Extending the formalism of finite automata to infinite domain
- Simplicity and readability (for non-experts)
- Closure properties (intersection, union, Kleene operator, concatenation, . . .)
- Decidability of main problems (membership, universality, emptiness, inclusion, (bi)simulation, . . .)
- Applications: e.g. Web Services, protocols

Plan

Plan

Motivating example

Service composition: simulation of CLIENT by CART⊗SEARCH



`x` is refreshed at p_0, p_1
`y` is refreshed at p_0
`z` is refreshed at q_0
`u` is refreshed at q_0, q_1
`w` is refreshed at r_0

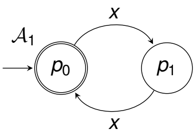
Parametrized automata (PAs)
Key ideas:

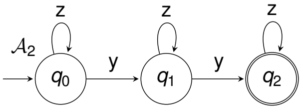
- Extension of finite automata to infinite alphabet (finite words, $|\Sigma| = \infty$)
- Transitions are labeled with variables or letters
  - The variables range over Σ
- Transitions are guarded with equalities/disequalities
- Variables are refreshed in some states

Parametrized automata (PAs)

Key ideas:

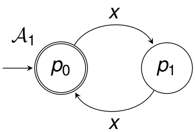
- Extension of finite automata to infinite alphabet (finite words, $|\Sigma| = \infty$)
- Transitions are labeled with variables or letters
  - The variables range over Σ
- Transitions are guarded with equalities/disequalities
- Variables are refreshed in some states

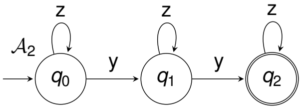




- $Fresh(x) = \{p_0\}$ and $Fresh(z) = \{q_0, q_1, q_2\}$

- Extension of finite automata to infinite alphabet (finite words, $|\Sigma| = \infty$)
- Transitions are labeled with variables or letters
  - The variables range over Σ
- Transitions are guarded with equalities/disequalities
- Variables are refreshed in some states



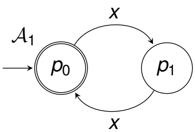


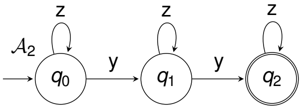
- $Fresh(x) = \{p_0\}$ and $Fresh(z) = \{q_0, q_1, q_2\}$
- $L(\mathcal{A}_1) = \{a_0 a_0 a_1 a_1 \ldots a_n a_n \mid a_i \in \Sigma\}$

Parametrized automata (PAs)

Key ideas:

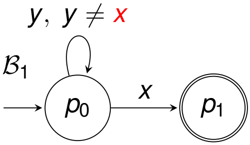
- Extension of finite automata to infinite alphabet (finite words, $|\Sigma| = \infty$)
- Transitions are labeled with variables or letters
  - The variables range over Σ
- Transitions are guarded with equalities/disequalities
- Variables are refreshed in some states

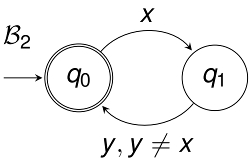




- $Fresh(x) = \{p_0\}$ and $Fresh(z) = \{q_0, q_1, q_2\}$
- $L(\mathcal{A}_1) = \{a_0 a_0 a_1 a_1 \ldots a_n a_n \mid a_i \in \Sigma\}$
- $L(\mathcal{A}_2) =$ all words in which a letter appears at least twice

Parametrized automata
Example

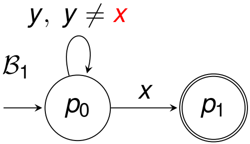




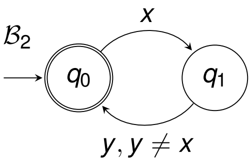
$$Fresh(y) = \{p_0\}$$

$$Fresh(x) = Fresh(y) = \{q_0\}$$

Parametrized automata
Example

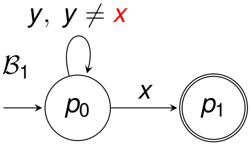


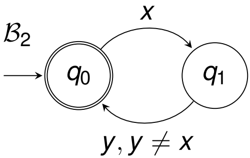
$Fresh(y) = \{p_0\}$



$Fresh(x) = Fresh(y) = \{q_0\}$

- $L(\mathcal{B}_1)$ = all the words in which the last letter *differs* from all the other letters





$Fresh(y) = \{p_0\}$ $\quad$ $Fresh(x) = Fresh(y) = \{q_0\}$

- $L(\mathcal{B}_1) =$ all the words in which the last letter *differs* from all the other letters
- $L(\mathcal{B}_2) = \{a_1 a'_1 \ldots a_n a'_n \mid a_i \neq a'_i, a_i \in \Sigma\}$

Parametrized automata

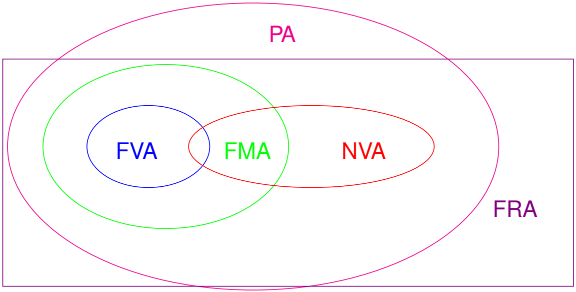
Definition of PAs

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

- Σ is a 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,
- $\delta : Q \times (\Sigma_{\mathcal{A}} \cup \mathcal{X}) \times \mathbb{G} \rightarrow 2^Q$ is a transition function where $\Sigma_{\mathcal{A}}$ is a finite subset of Σ,
- $F \subseteq Q$ is a set of accepting states,
- $\kappa : \mathcal{X} \rightarrow 2^Q$ is the refreshing function that associates to every variable the (possibly empty) set of states where it is refreshed.

▷ $\mathbb{G} ::= (\bigwedge_i \alpha_i = \beta_i) \wedge (\bigwedge_j \alpha_j \neq \beta_j)$, $\qquad \alpha_i, \beta_i, \alpha_j, \beta_j \in \Sigma_{\mathcal{A}} \cup \mathcal{X}$

Comparison with other models



- FVA: Fresh-variable automata (Belkhir et. al)
- FMA: Finite memory automata (Kaminsky et. al)
- NVA: Nondeterminist variable automata (Kupferman et. al)
- PA: Parametrized automata (Belkhir et. al)
- FRA: Fresh-register automata (N. Tzevelekos)

Plan

Closure properties and decision procedures in service composition

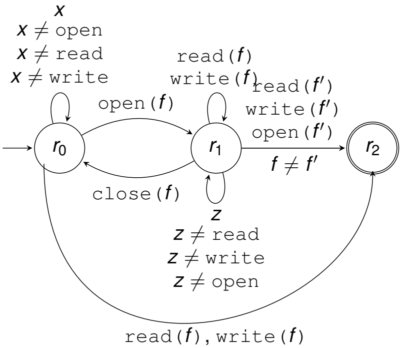
- Checking whether a service satisfies a policy can be reduced to checking that the intersection of the service PA with a PA specifying the forbidden executions is empty.
  - Closure of PAs under intersection
  - Checking Nonemptyness of PAs

Closure properties and decision procedures in service composition

- Checking whether a service satisfies a policy can be reduced to checking that the intersection of the service PA with a PA specifying the forbidden executions is empty.
  - Closure of PAs under intersection
  - Checking Nonemptyness of PAs
- If the policy is expressed as a language of authorized traces, then checking whether a service $\mathcal{M}$ respects the policy $\mathcal{Q}$ amounts to check the containment $L(\mathcal{M}) \subseteq L(\mathcal{Q})$.
  - Decidability of containment
  - If containment is undecidable, it can be underapproximated by simulation preorder.

A usage policy for opening and reading files

- A file named f must be open before being used, where f is a variable,
- the number of files which are open at the same time is at most one.



Closure properties of PAs

Theorem

PAs are closed under intersection, union, concatenation and Kleene operator but not under complementation

- Union, intersection: Straightforward
- Concatenation & Kleene operator: Introduction of unguarded ε-transitions does not increase the expressivity of PAs
- Complementation: the language of all words in which a letter appears at least twice can not be complemented

Decision procedures for PAs (1/3)

Theorem

For PAs

- Membership (i.e. $w \in L(\mathcal{A})$?) is NP-complete,
- Universality (i.e. $L(\mathcal{A}) = \Sigma^\star$?) is undecidable,
- Inclusion (i.e. $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$?) is undecidable, and
- Nonemptiness (i.e. $L(\mathcal{A}) \neq \emptyset$?) is PSPACE-Complete.

Decision procedures for PAs (1/3)

Theorem

For PAs

- Membership (i.e. $w \in L(\mathcal{A})$?) is NP-complete,
- Universality (i.e. $L(\mathcal{A}) = \Sigma^\star$?) is undecidable,
- Inclusion (i.e. $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$?) is undecidable, and
- Nonemptiness (i.e. $L(\mathcal{A}) \neq \emptyset$?) is PSPACE-Complete.

- Membership: guessing an accepting path. Reduction of Hamiltonian cycle
- Universality & Inclusion: undecidable for the strict subclass of Finite Memory Automata (FMA)

Decision procedures for PAs (2/3)

Lemma

Nonemptiness (i.e. $L(\mathcal{A}) \neq \emptyset$?) is PSPACE.

- Reduction to the Nonemptiness of PAs in which the variables range over a finite set of letters.

Decision procedures for PAs (2/3)

Lemma

Nonemptiness (i.e. $L(\mathcal{A}) \neq \emptyset$?) is PSPACE.

- Reduction to the Nonemptiness of PAs in which the variables range over a finite set of letters.
- A Turing machine stores two pieces of data:

Decision procedures for PAs (2/3)

Lemma

Nonemptiness (i.e. $L(\mathcal{A}) \neq \emptyset$?) is PSPACE.

- Reduction to the Nonemptiness of PAs in which the variables range over a finite set of letters.
- A Turing machine stores two pieces of data:
  - the current configuration (q, σ), where q is a state, σ is a substitution,

Decision procedures for PAs (2/3)

Lemma

Nonemptiness (i.e. $L(\mathcal{A}) \neq \emptyset$?) is PSPACE.

- Reduction to the Nonemptiness of PAs in which the variables range over a finite set of letters.
- A Turing machine stores two pieces of data:
  - the current configuration (q, σ), where q is a state, σ is a substitution,
  - a counter for the number of configurations visited so far.

Decision procedures for PAs (2/3)

Lemma

Nonemptiness (i.e. $L(\mathcal{A}) \neq \emptyset$?) is PSPACE.

- Reduction to the Nonemptiness of PAs in which the variables range over a finite set of letters.
- A Turing machine stores two pieces of data:
  - the current configuration (q, σ), where q is a state, σ is a substitution,
  - a counter for the number of configurations visited so far.
- Since the size of this data is polynomial, a Turing machine running in polynomial space can solve the Nonemptiness problem:

Decision procedures for PAs (2/3)

Lemma

Nonemptiness (i.e. $L(\mathcal{A}) \neq \emptyset$?) is PSPACE.

- Reduction to the Nonemptiness of PAs in which the variables range over a finite set of letters.
- A Turing machine stores two pieces of data:
  - the current configuration (q, σ), where q is a state, σ is a substitution,
  - a counter for the number of configurations visited so far.
- Since the size of this data is polynomial, a Turing machine running in polynomial space can solve the Nonemptiness problem:
  - exponential number of configurations.

Lemma

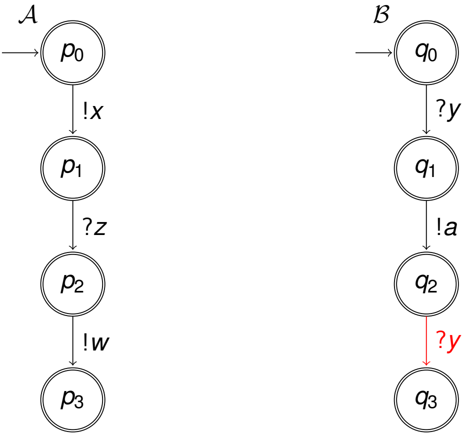
Nonemptiness (i.e. $L(\mathcal{A}) \neq \emptyset$?) is PSPACE-hard.

- PAs can encode bounded one-counter automata (`Boca`):
  - PAs over the finite alphabet $\Sigma = \{0, 1\}$
  - variables play the role of registers (finite number of variables)
  - PAs can encode addition and subtraction
- Reachability of `Boca` is PSPACE-complete:
  - reduce the reachability of `Boca` to the Nonemptiness of PAs

Plan

Communicating PAs and communicating simulation

$!x = !a_0, !a_1, !a_2, \ldots$ where $a_i \in \Sigma$



Decidability of the (communicating) simulation (1/5)

Theorem

The (communicating) simulation for PAs is in EXPTIME.

Key points of the proof:

1. Game theoretic characterization of the simulation($\preceq$) (`Eloise` (service) vs `Abelard` (client))
   - $\mathcal{A}_1 \preceq \mathcal{A}_2$ iff `Eloise` has a winning strategy in $\mathcal{G}_\Sigma(\mathcal{A}_1, \mathcal{A}_2)$
   - Positions of the game are pairs of configurations $((q_1, \sigma_1), (q_2, \sigma_2))$ where q_i is a state of $\mathcal{A}_i$ and $\sigma_i : \mathcal{X} \to \Sigma$ is a substitution
   - Infinite plays are winning for `Eloise`

Theorem

The (communicating) simulation for PAs is in EXPTIME.

Key points of the proof:

2. Constructing an equivalent game $\mathcal{G}_{\mathcal{C}_0}(\mathcal{A}_1, \mathcal{A}_2)$ in which the players instantiate the variables from the finite set of letters $\mathcal{C}_0$:

$$\mathcal{C}_0 = \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2} \cup \{c_1, \ldots, c_n\}$$

where

$$\begin{cases} \Sigma_{\mathcal{A}_i} & = \text{ finite set of letters appearing in the PA } \mathcal{A}_i, \\ |\mathcal{X}_1 \cup \mathcal{X}_2| & = n \end{cases}$$

Decidability of the (communicating) simulation (3/5)

How to prove that the games $\mathcal{G}_\Sigma$ and $\mathcal{G}_{\mathcal{C}_0}$ are equivalent?

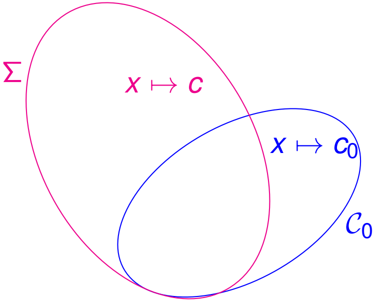
$\mathcal{G}_\Sigma \implies \mathcal{G}_{\mathcal{C}_0}$:

$$
\begin{tikzcd}
\mathcal{G}_\Sigma & \mathcal{G}_{\mathcal{C}_0} \\
\wp \arrow[r, dashed, "f"] \arrow[d, "\mathtt{A}"] & \wp_0 = f(\wp) \arrow[d, "\mathtt{A}"] \\
\wp' \arrow[r, dashed, "f"] \arrow[d, "\mathtt{E}"] & \wp'_0 = f(\wp') \arrow[d, "\mathtt{E}"] \\
\wp'' \arrow[r, dashed, "f"] & \wp''_0 = f(\wp'')
\end{tikzcd}
$$

Proof of direction $\mathcal{G}_\Sigma \Rightarrow \mathcal{G}_{\mathcal{C}_0}$

Instantiation strategy:

1. $c \in \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2}$

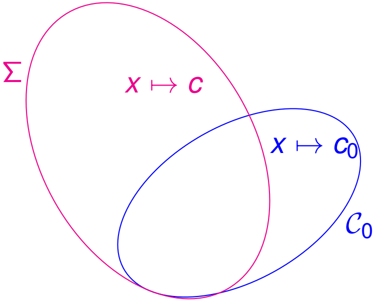


$\Sigma \cap \mathcal{C}_0 = \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2}$

Proof of direction $\mathcal{G}_\Sigma \Rightarrow \mathcal{G}_{\mathcal{C}_0}$

Instantiation strategy:

1. $c \in \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2}$
   - $c_0 = c$

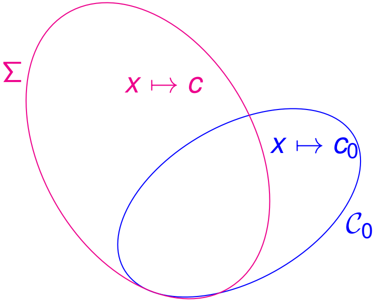


$$\Sigma \cap \mathcal{C}_0 = \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2}$$

Proof of direction $\mathcal{G}_\Sigma \Rightarrow \mathcal{G}_{\mathcal{C}_0}$

Instantiation strategy:

1. $c \in \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2}$
   - $c_0 = c$
2. $c \in \Sigma \setminus \mathcal{C}_0$ and c is new in the current configuration

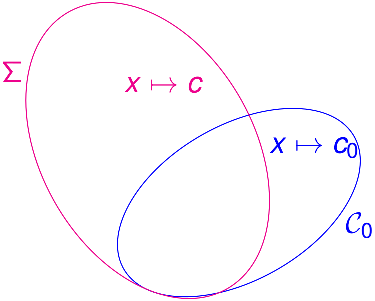


$\Sigma \cap \mathcal{C}_0 = \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2}$

Proof of direction $\mathcal{G}_\Sigma \Rightarrow \mathcal{G}_{\mathcal{C}_0}$

Instantiation strategy:

1. $c \in \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2}$
   - $c_0 = c$
2. $c \in \Sigma \setminus \mathcal{C}_0$ and c is new in the current configuration
   - $c_0 \in \mathcal{C}_0 \setminus (\Sigma \cap \mathcal{C}_0)$ must be new

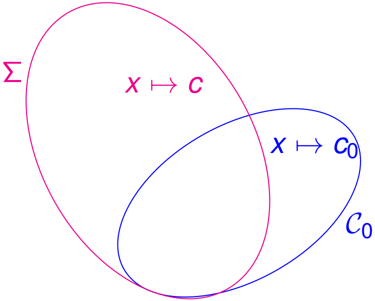


$\Sigma \cap \mathcal{C}_0 = \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2}$

Proof of direction $\mathcal{G}_\Sigma \Rightarrow \mathcal{G}_{\mathcal{C}_0}$

Instantiation strategy:

1. $c \in \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2}$
   - $c_0 = c$
2. $c \in \Sigma \setminus \mathcal{C}_0$ and c is new in the current configuration
   - $c_0 \in \mathcal{C}_0 \setminus (\Sigma \cap \mathcal{C}_0)$ must be new
3. $c \in \Sigma \setminus \mathcal{C}_0$ and c appears in the current configuration

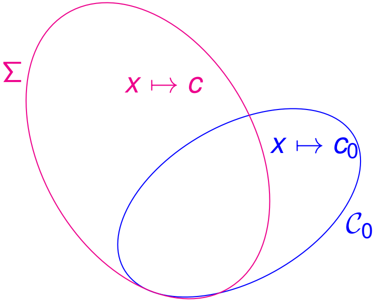


$\Sigma \cap \mathcal{C}_0 = \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2}$

Proof of direction $\mathcal{G}_\Sigma \Rightarrow \mathcal{G}_{\mathcal{C}_0}$

Instantiation strategy:

1. $c \in \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2}$
   - $c_0 = c$
2. $c \in \Sigma \setminus \mathcal{C}_0$ and c is new in the current configuration
   - $c_0 \in \mathcal{C}_0 \setminus (\Sigma \cap \mathcal{C}_0)$ must be new
3. $c \in \Sigma \setminus \mathcal{C}_0$ and c appears in the current configuration
   - $c_0 =$ back to the previous choice



$\Sigma \cap \mathcal{C}_0 = \Sigma_{\mathcal{A}_1} \cup \Sigma_{\mathcal{A}_2}$

lemma

Simulation for PAs over finite alphabet is in EXPTIME.

We use an alternating Turing machine running in polynomial space:

- We need to store two pieces of data:
  - the current game position $((q_1, \sigma_1), (q_2, \sigma_2))$, where q_i is a state, σ_i is a substitution,
  - a counter for the number of positions visited so far.

lemma

Simulation for PAs over finite alphabet is in EXPTIME.

We use an alternating Turing machine running in polynomial space:

- We need to store two pieces of data:
  - the current game position $((q_1, \sigma_1), (q_2, \sigma_2))$, where q_i is a state, σ_i is a substitution,
  - a counter for the number of positions visited so far.
- Since the size of this data is polynomial, an ATM can solve the simulation game:
  - existential states $\sim$ player `Eloise`
  - universal states $\sim$ player `Abelard`

lemma

Simulation for PAs over finite alphabet is in EXPTIME.

We use an alternating Turing machine running in polynomial space:

- We need to store two pieces of data:
  - the current game position $((q_1, \sigma_1), (q_2, \sigma_2))$, where q_i is a state, σ_i is a substitution,
  - a counter for the number of positions visited so far.
- Since the size of this data is polynomial, an ATM can solve the simulation game:
  - existential states $\sim$ player `Eloise`
  - universal states $\sim$ player `Abelard`
- APSPACE=EXPTIME

Decision procedures: summary

	FVAs	PAs
Membership	NP-Complete	NP-Complete
Nonemptiness	NL-Complete	PSPACE-Complete
Universality	Decidable	Undecidable
Inclusion	?	Undecidable
(bi)Simulation	EXPTIME	EXPTIME-complete

FVAs = PAs without guards (all guards are true)

References

[SYNASC'13] W. Belkhir, Y. Chevalier, and M. Rusinowitch. Fresh-Variable Automata: Application to Service Composition.

[JSC'14] W. Belkhir, Y. Chevalier, and M. Rusinowitch. Parametrized automata simulation and application to service composition.

[AiML'14] W. Belkhir, G. Rossi and M. Rusinowitch, A Parametrized Propositional Dynamic Logic with Application to Service Synthesis.

Plan

Current work and perspectives

- Model checking of PAs with parametrized μ-calculus
- Service synthesis under policy enforcement as parametrized μ-calculus model-checking
- Service synthesis under timed constraints with timed PAs