Parametric Verification of Protocols

Wojciech Penczek and Michał Knapik

a joint work with Artur Męski

Institute of Computer Sciences, PAS, Warsaw, Poland
Siedlce Universisty, Poland.

WG2.2 Meeting
Munich, Germany, September 2014

Outline

Introduction: Parametric Model Checking

MODEL CHECKING:

- fixed model $\mathcal{M}$
- fixed property ϕ

check: $\mathcal{M} \models \phi$?

PARAMETRIC MODEL CHECKING:

allow free parameters in $\mathcal{M}$ or ϕ:

synthesise all parameter valuations v s.t. $\mathcal{M} \models_v \phi$

How to do it in an efficient automatic way, if possible at all ?

Some Related Work

- Parameter synthesis for parametric version of LTL, Alur et al., 2001,
- Verification of feature CTL, Schnoebellen 2002,
- Parametric model checking for TCTL, Bruyere et al. 2008,
- Parametric model checking for MITL, Giampaolo et al. 2010,
- Model checking of parametric timed automata, Many papers.

Related Work (of our group: Knapik, Meski, Szreter, Penczek)

- SMT-based Parameter Synthesis for L/U Automata, PNSE 2012,
- Bounded Model Checking for Parametric Timed Automata, ToPNoC 2012,
- Group Synthesis for Parametric Epistemic-Temporal Logic, AAMAS 2012,
- Action Synthesis for Branching Time Logic: Theory and Applications, ACSD 2014,
- Parameter Synthesis for Timed Kripke Structures, Fundam. Inform. 2014.

The most recent results

- Theory of fixed-point based synthesis for pmARCTL,
- Open-source tool SPATULA:
  - BDD-based symbolic engine,
  - Brute-force BDD engine for benchmarking comparison,
  - C-like model description language,
- Experimental evaluation:
  - Promising results on scalable benchmarks.

Models: Labeled Transition Systems (MTS) (1)

MTS: Kripke models with the action-labeled transitions

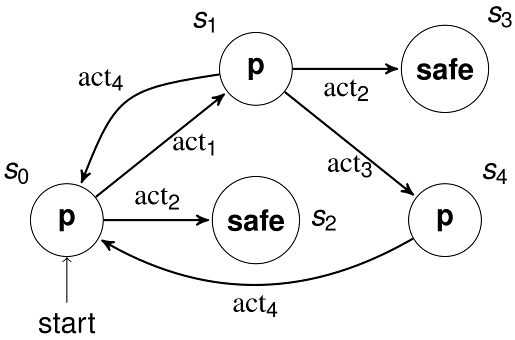
A 5-tuple $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{T}, \mathcal{L})$ is a MTS, where:

- $\mathcal{S}$ – a set of states,
- $s^0 \in \mathcal{S}$ – the initial state,
- $\mathcal{A}$ – **a set of actions**,
- $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ – **a labeled transition relation**,
- $\mathcal{PV}$ – a set of the propositional variables,
- $\mathcal{L} : \mathcal{S} \to 2^{\mathcal{PV}}$ – a labeling function.

A path π in $\mathcal{M}$ is a sequence $s_0 a_0 s_1 a_1 \ldots$ of states and actions such that $(s_i, a_i, s_{i+1}) \in \mathcal{T}$.

$\pi_i = s_i$, $\mid \pi \mid$ is the number of the states of π

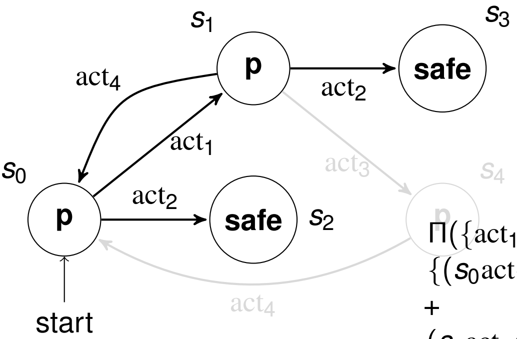
Models: Labeled Transition Systems (2)



$B \subseteq \mathcal{A}$ – a set of allowed actions

- $\Pi(B, s)$ – the maximal paths over B, starting from s
- $\Pi^\omega(B, s)$ – the infinite paths over B, starting from s

Models: Labeled Transition Systems (2)

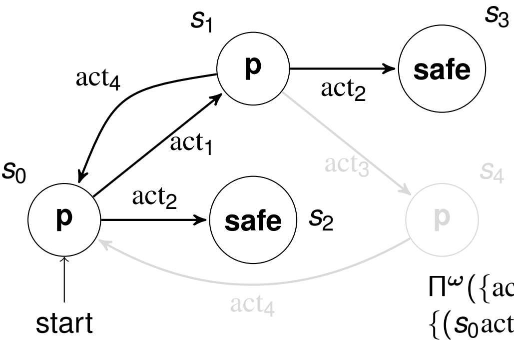


$$\Pi(\{\mathrm{act}_1, \mathrm{act}_2, \mathrm{act}_4\}, \mathrm{start}) =$$
$$\{(s_0 \mathrm{act}_1 s_1 \mathrm{act}_4)^\omega$$
$$+$$
$$(s_0 \mathrm{act}_1 s_1 \mathrm{act}_4)^* s_0 \mathrm{act}_1 s_1 \mathrm{act}_2 s_3$$
$$+$$
$$(s_0 \mathrm{act}_1 s_1 \mathrm{act}_4)^* s_0 \mathrm{act}_2 s_2\}$$

$B \subseteq \mathcal{A}$ – a set of allowed actions

- $\Pi(B, s)$ – the maximal paths over B, starting from s
- $\Pi^\omega(B, s)$ – the infinite paths over B, starting from s

Models: Labeled Transition Systems (2)



$\Pi^{\omega}(\{\mathrm{act}_1, \mathrm{act}_2, \mathrm{act}_4\}, \mathrm{start}) = \{(s_0 \mathrm{act}_1 s_1 \mathrm{act}_4)^{\omega}\}$

$B \subseteq \mathcal{A}$ – a set of allowed actions

- $\Pi(B, s)$ – the maximal paths over B, starting from s
- $\Pi^{\omega}(B, s)$ – the infinite paths over B, starting from s

pmARCTL: Syntax (1)

ActSets – a set of the non-empty subsets of $\mathcal{A}$,
ActVars – a set of the action set (group) variables

pmARCTL: the formulae ϕ generated by the BNF grammar:

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid E_{\alpha}X\phi \mid E_{\alpha}G\phi \mid E^{\omega}_{\alpha}G\phi \mid E_{\alpha}(\phi\ U\ \phi)$$

$p \in \mathcal{PV}$, $\alpha \in$ ActSets $\cup$ ActVars

- E_{α} - *there exists a maximal path over* α,
- E^{ω}_{α} - *there exists an infinite path over* α,
- X, G, U - *neXt, Globally, Until*

pmARCTL: CTL with action set and group variable subscripts

pmARCTL: Syntax (2)

Derived modalities:

1. $E^{\omega}_{\alpha}(\phi\ U\psi) \stackrel{def}{=} E_{\alpha}\big(\phi\ U(\psi \wedge E^{\omega}_{\alpha} G\ \textbf{true})\big)$
2. $E^{r}_{\alpha} F\phi \stackrel{def}{=} E^{r}_{\alpha}(\textbf{true}\ U\phi)$
3. $A^{r}_{\alpha} G\phi \stackrel{def}{=} \neg E^{r}_{\alpha} F\neg\phi$
4. $A^{r}_{\alpha}(\phi\ U\psi) \stackrel{def}{=} \neg\big(E^{r}_{\alpha}(\neg\psi U\neg(\phi \vee \psi)) \vee E^{r}_{\alpha} G\neg\psi\big)$
5. $A^{r}_{\alpha} F\phi \stackrel{def}{=} \neg E^{r}_{\alpha} G\neg\phi$

$\phi, \psi \in$ pmARCTL, $\alpha \in$ ActSets $\cup$ ActVars, $r \in \{\boldsymbol{\omega}, \boldsymbol{\epsilon}\}$

- A_{α} – *for All maximal paths over* α,
- A^{ω}_{α} – *for All infinite paths over* α,
- *F – in Future*.

Examples of pmARCTL formulae expressivness:

1. $E^{\omega}_{Y} G$**true** – detection of the infinite paths,
2. $A_Y G E_Y X$**true** – detection of a lack of Y-deadlocks,
3. $A_Y G($**p** $\wedge$ EF_Z**safe**$)$ – nested variables,
4. $E_{\{forward,left\}}($**free**$U E^{\omega}_{Y} G$**safe**$)$ – mixed formula: actions and variables,
5. $A_Y G \phi$ – how to modify a model so that ϕ holds.

pmARCTL: Semantics (1)

The formulae interpreted over models w.r.t. the variables valuations

- υ : ActVars $\rightarrow$ ActSets

(slight notational abuse: if $\alpha \in$ ActSets, then $\upsilon(\alpha) = \alpha$)

Semantics:

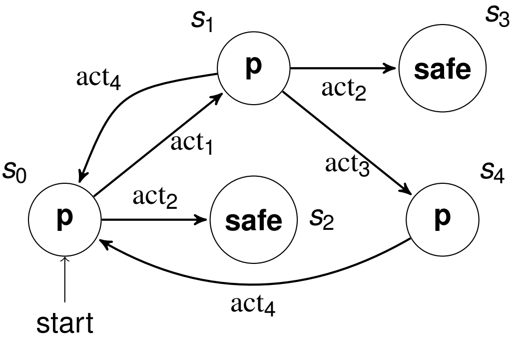
- $s \models_\upsilon p$ iff $p \in \mathcal{L}(s)$,
- $s \models_\upsilon \neg\phi$ iff $s \not\models_\upsilon \phi$,
- $s \models_\upsilon \phi \vee \psi$ iff $s \models_\upsilon \phi$ or $s \models_\upsilon \psi$,
- $s \models_\upsilon E_\alpha X\phi$ iff there is $\pi \in \Pi(\upsilon(\alpha), s)$ s.t. $|\pi| > 1$ and $\pi_1 \models_\upsilon \phi$,

pmARCTL: Semantics (2)

- $s \models_\upsilon E^\omega_\alpha G\phi$ iff there is $\pi \in \Pi^\omega(\upsilon(\alpha), s)$ s.t. $\pi_i \models_\upsilon \phi$ for all $i < |\pi|$,
- $s \models_\upsilon E_\alpha G\phi$ iff there is $\pi \in \Pi(\upsilon(\alpha), s)$ s.t. $\pi_i \models_\upsilon \phi$ for all $i < |\pi|$,
- $s \models_\upsilon E_\alpha(\phi\ U\psi)$ iff there is $\pi \in \Pi(\upsilon(\alpha), s)$ s.t. $\pi_i \models_\upsilon \psi$ for some $i < |\pi|$ and $\pi_j \models_\upsilon \phi$ for all $0 \leq j < i$.

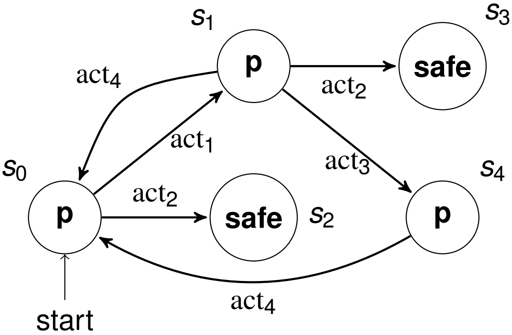
$p \in \mathcal{PV}$, $\phi, \psi \in$ pmARCTL, $\alpha \in$ ActSets $\cup$ ActVars.

Action Synthesis: the Problem in the Nutshell



$A_{\color{red}Y} G(\mathbf{p} \wedge E_{\color{blue}Z} F\,\mathbf{safe})$: *for each* *Y-reachable state* ***p*** *holds and* ***safe*** *is* *Z-reachable*

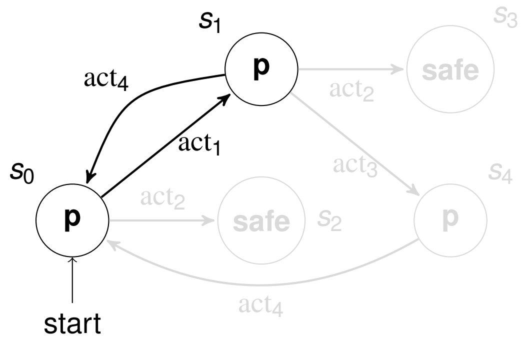
Action Synthesis: the Problem in the Nutshell



$A_Y G(\mathbf{p} \wedge E_Z F\mathbf{safe})$: *for each* Y*-reachable state* ***p*** *holds and* ***safe*** *is* Z*-reachable*

$$\text{start} \models A_{\{\text{act}_1,\, \text{act}_4\}} G(\mathbf{p} \wedge E_{\{\text{act}_2\}} F\mathbf{safe})$$

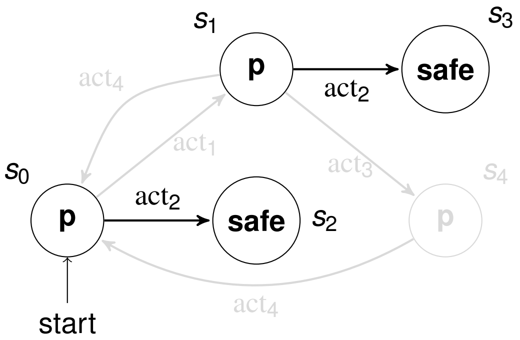
Action Synthesis: the Problem in the Nutshell



$A_Y G(\mathbf{p} \wedge E_Z F\mathbf{safe})$: *for each Y-reachable state **p** holds and **safe** is Z-reachable*

$$\text{start} \models A_{\underline{\{\mathrm{act}_1,\, \mathrm{act}_4\}}} G(\mathbf{p} \wedge E_{\{\mathrm{act}_2\}} F\mathbf{safe})$$

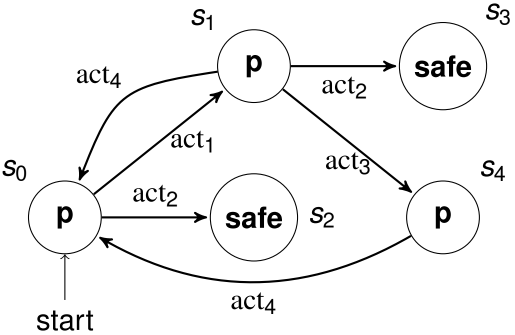
Action Synthesis: the Problem in the Nutshell



$A_Y G(\mathbf{p} \wedge E_Z F\mathbf{safe})$: *for each Y-reachable state **p** holds and **safe** is Z-reachable*

$$\text{start} \models A_{\{\text{act}_1,\, \text{act}_4\}} G(\mathbf{p} \wedge E_{\underline{\{\text{act}_2\}}} F\mathbf{safe})$$

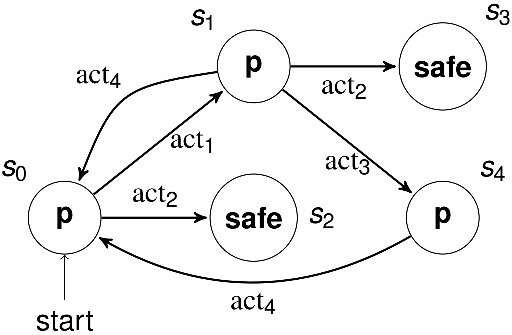
Action Synthesis: the Problem in the Nutshell



$A_Y G(\mathbf{p} \wedge E_Z F\,\mathbf{safe})$: *for each Y-reachable state* ***p*** *holds and* ***safe*** *is Z-reachable*

$$\text{start} \not\models A_{\{\mathrm{act}_1,\,\mathrm{act}_3\}} G(\mathbf{p} \wedge E_{\{\mathrm{act}_2\}} F\,\mathbf{safe})$$

Action Synthesis: the Problem in the Nutshell



$A_Y G(\mathbf{p} \wedge E_Z F\mathbf{safe})$: *for each* Y*-reachable state* ***p*** *holds and* ***safe*** *is* Z*-reachable*

Goal: find the valuations v for Y, Z s.t.:

$$\text{start} \models A_{v(Y)} G(\mathbf{p} \wedge E_{v(Z)} F\mathbf{safe})$$

Action Synthesis: Formal Definition

$\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{T}, \mathcal{L})$, $\phi \in$ pmARCTL
denote ActVals $\stackrel{def}{=}$ ActSets$^{\text{ActVars}}$

Goal: build $f_\phi : \mathcal{S} \to 2^{\text{ActVals}}$ s.t. for all $s \in \mathcal{S}$:

$$\upsilon \in f_\phi(s) \iff s \models_\upsilon \phi$$

($f_\phi(s)$ contains all valuations that make ϕ hold in s)

THEOREM
The problem of deciding whether $f_\phi(s) \neq \emptyset$ is NP-complete.

Evaluation: Peterson's Algorithm – Introduction

- 2 processes compete to access the critical section
- 3 shared memory bits available
- mutual exclusion holds
- no deadlock, non-blocking, no strict sequencing

Variable initialisation

$B_0 := False$; $B_1 := False$

Process 0

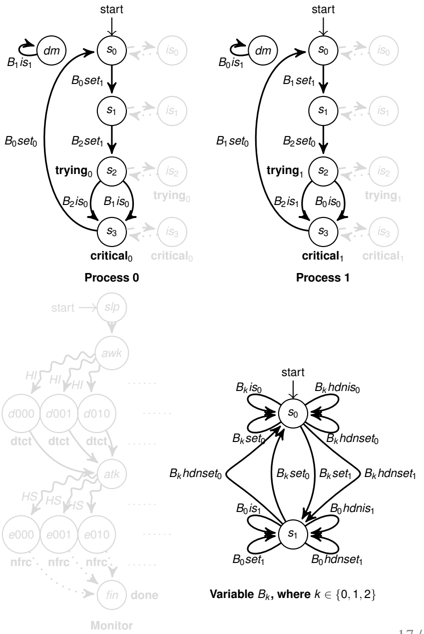
$B_0 := True$
$B_2 := True$
while $B_1 = True$ and $B_2 = True$ **do**
    pass {busy wait}
end while
{critical section}
$B_0 := False$

Process 1

$B_1 := True$
$B_2 := False$
while $B_0 = True$ and $B_2 = False$ **do**
    pass {busy wait}
end while
{critical section}
$B_1 := False$

Evaluation: Peterson's Algorithm – Modelling Malicious Monitor (1)

- normal operation:
  basic properties verified
- interrupt request:
  1. freeze processes
  2. inspect variables
  3. play with variables
  4. un-freeze
- resume operation



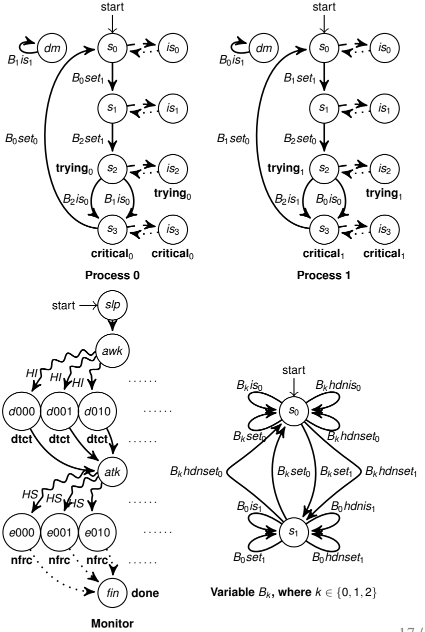
Evaluation: Peterson’s Algorithm – Modelling Malicious Monitor (1)

- normal operation:

  basic properties verified

- interrupt request:
  1. freeze processes
  2. inspect variables
  3. play with variables
  4. un-freeze

- resume operation



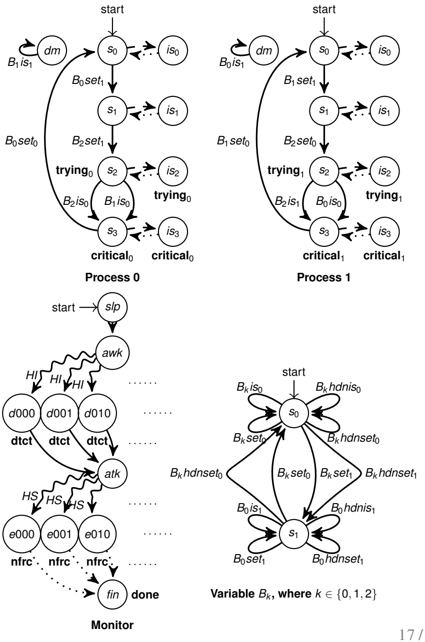
Evaluation: Peterson's Algorithm – Modelling Malicious Monitor (1)

- normal operation:

  basic properties verified

- interrupt request:
  1. freeze processes
  2. inspect variables
  3. play with variables
  4. un-freeze

- resume operation



Evaluation: Peterson's Algorithm – Modelling Malicious Monitor (2)

Question: *"can the monitor infer only by looking at the values of shared variables if any of two processes is attempting to enter or already entered the critical section?"*

$$\phi_{dtct} = E_{\mathcal{A}_{\text{norm}}} F A_{Y} G(\text{dtct} \Longrightarrow (\text{trying}_0 \vee \text{trying}_1 \vee \text{critical}_0 \vee \text{critical}_1)) \wedge E_{Y} F \text{dtct}$$

Answer: NO. PA is not susceptible to easy eavesdropping.

(0.04 sec. param. vs 1.06 sec. naïve)

Evaluation: Peterson's Algorithm – Modelling Malicious Monitor (2)

Question: *"can the monitor test and set shared variables s.t. after return from interrupt and a single step at least one of processes attempts to enter or already entered critical section?"*

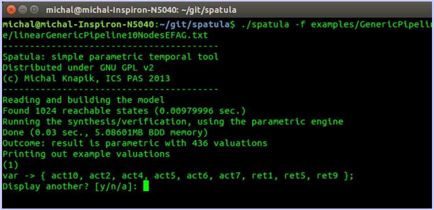
$$\phi_{\mathrm{nfrcAX}} = E_{\mathcal{A}_{\mathrm{norm}}} F A_{Y} G(\mathrm{nfrc} \implies A_{\{irqret\}} X A_{\mathcal{A}_{\mathrm{norm}}} X$$
$$(\mathrm{trying}_0 \vee \mathrm{trying}_1 \vee \mathrm{critical}_0 \vee \mathrm{critical}_1)) \wedge E_{Y} F \mathrm{done}.$$

make **Answer:** NO. PA is not susceptible to obvious disruption.

(0.07 sec. param. vs 87.41 sec. naïve)

SPATULA: BDD-based pmARCTL synthesis and verification

- input models: networks of automata
- C-like model description language
- GNU GPL license
- written in C/C++/CUDD




```
module Controller:
  trainsNo = k;
  faultyTrainNo = j;

  /* correct behaviour */
  bloom("s0");
  mark_with("s0", "initial");
  mark_with("s0", "green");
  bloom("s1");
  mark_with("s1", "red");
  ctr = 1;
  while(ctr <= trainsNo) {
     outlabel = "out" + ctr;
     inlabel  = "in" + ctr;
     join_with("s0", "s1", inlabel);
     join_with("s1", "s0", outlabel);
     ctr = ctr + 1;
  }

  /* faulty behaviour */
  inlabelF = "inF" + faultyTrainNo;
  outlabelF = "outF" + faultyTrainNo;
  join_with("s0", "s0", inlabelF);
  join_with("s1", "s1", outlabelF);
```

`https://michalknapik.github.io/spatula`

Conclusions and future work

- Definition of parametric extension of ARCTL,
- Complexity of pmARCTL model checking,
- A new symbolic approach to pmARCTL model checking,
- Experimental results show that our tool can deal with small and medium models reasonably fast, so it could be used by system designers.

- Future: Application to checking security of protocols.

Thank you