Action Synthesis for Branching Time Logic: Theory and Applications

Michał Knapik, Artur Męski, Wojciech Penczek

Institute of Computer Sciences, PAS, Warsaw, Poland

LIPN Paris XIII, 2 July 2015

Outline

Introduction: Parametric Model Checking

MODEL CHECKING:

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

test: $\mathcal{M} \models \phi$?
alter $\mathcal{M}$ or ϕ if false

PARAMETRIC MODEL CHECKING:

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

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

needs something better than brute-force enumeration over all v

Introduction: this Work's Contribution

- fixpoint-based **synthesis** for CTL-like parametric logic
- open-source tool **SPATULA**:
  - BDD-based **symbolic engine**
  - brute-force BDD engine for **benchmarking comparison**
  - C-like model **description language**
- experimental evaluation
  - some applications to **security analysis**
  - **promising results** on scalable benchmarks

pmARCTL: Mixed Transition Systems (1)

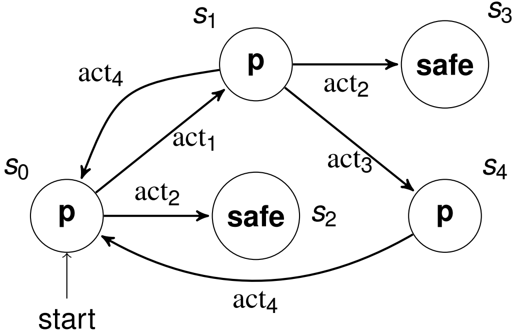
$\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{T}, \mathcal{L})$, where:

- $\mathcal{S}$ – states
- $s^0 \in \mathcal{S}$ – the initial state
- $\mathcal{A}$ – actions
- $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ – transition relation
- $\mathcal{L} : \mathcal{S} \to 2^{\mathcal{PV}}$ – labeling by propositions from $\mathcal{PV}$

is a MTS.

(MTS: Kripke structures with action-labeled transitions)

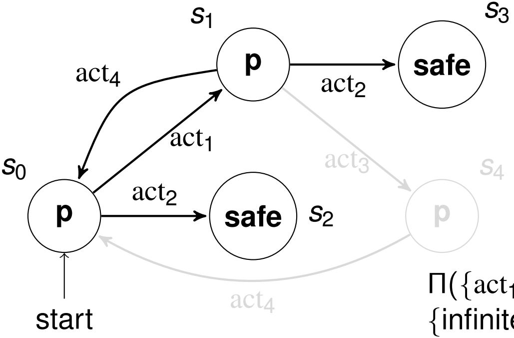
pmARCTL: **Mixed Transition Systems (2)**



$\chi \subseteq \mathcal{A}$ – allowed actions

- $\Pi(\chi, s)$ – *maximal* paths over χ, starting from s
- $\Pi^{\omega}(\chi, s)$ – *maximal infinite* paths over χ, starting from s

pmARCTL: **Mixed Transition Systems (2)**

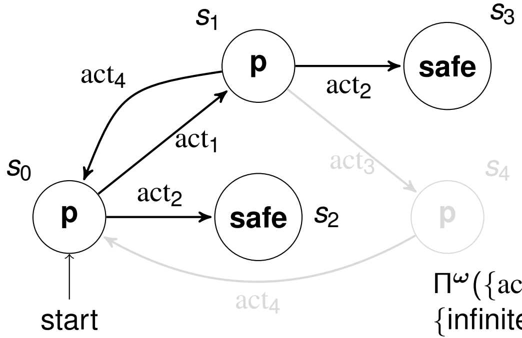


$\Pi(\{\mathrm{act}_1, \mathrm{act}_2, \mathrm{act}_4\}, \mathrm{start}) =$
$\{$infinite **p** – loop
+ **p** – loops ending in **safe**$\}$

$\chi \subseteq \mathcal{A}$ – allowed actions

- $\Pi(\chi, s)$ – *maximal* paths over χ, starting from s
- $\Pi^{\omega}(\chi, s)$ – *maximal infinite* paths over χ, starting from s

pmARCTL: **Mixed Transition Systems (2)**



$\Pi^{\omega}(\{\mathrm{act}_1, \mathrm{act}_2, \mathrm{act}_4\}, \mathrm{start}) =$
$\{$infinite **p** – loop$\}$

$\chi \subseteq \mathcal{A}$ – allowed actions

- $\Pi(\chi, s)$ – *maximal* paths over χ, starting from s
- $\Pi^{\omega}(\chi, s)$ – *maximal infinite* paths over χ, starting from s

pmARCTL: **Syntax (1)**

ActSets – subsets of $\mathcal{A}$, ActVars – group variables

pmARCTL: ϕ – formulae generated by BNF grammar:

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

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

- E_α - *Exists a path over* α
- X, G, G^{ω}, U - *neXt, Globally, Globally and infinitely, Until*

(pmARCTL: CTL with actions/variable subscripts)

pmARCTL**: Syntax (2)**

Derived modalities:

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

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

- A_α – *for All paths over* α, *F* – *in Future*, $^{\omega}$ – infinite paths

pmARCTL: Syntax (3)

Some examples:

1. $A_Y G E_Y X \mathbf{true}$ – (lack of) deadlock detection
2. $A_Y G(\mathbf{p} \wedge EF_Z \mathbf{safe})$
3. $E_{\{forward, left\}} \mathbf{free} U E_Y G^{\omega} \mathbf{safe}$ – mixed formula

pmARCTL: Semantics (1)

Formulae interpreted w.r.t action 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 exists $\pi \in \Pi(\upsilon(\alpha), s)$ s. t. $|\pi| > 1$ and $\pi_1 \models_\upsilon \phi$

. . .continued on next slide

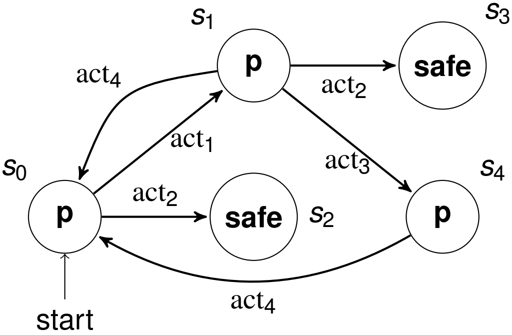
pmARCTL: **Semantics (2)**

. . .continued

- $s \models_\upsilon E_\alpha G^\omega \phi$ iff exists $\pi \in \Pi^\omega(\upsilon(\alpha), s)$ s. t. $\pi_i \models_\upsilon \phi$ for all $i \in \mathbb{N}$
- $s \models_\upsilon E_\alpha G \phi$ iff exists $\pi \in \Pi(\upsilon(\alpha), s)$ s. t. $\pi_i \models_\upsilon \phi$ for all $i < |\pi|$
- $s \models_\upsilon E_\alpha(\phi\ U \psi)$ iff exists $\pi \in \Pi(\upsilon(\alpha), s)$ s. t. $\pi_i \models_\upsilon \psi$ for some $i < |\pi|$ and $\pi_j \models_\upsilon \phi$ for all $0 \leq j < i$

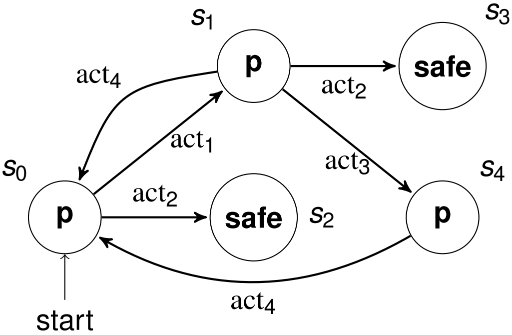
$p \in \mathcal{PV}$, $\phi, \psi \in$ pmARCTL, $\alpha \in$ ActSets $\cup$ ActVars.

Action Synthesis: the Problem in a Nutshell



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

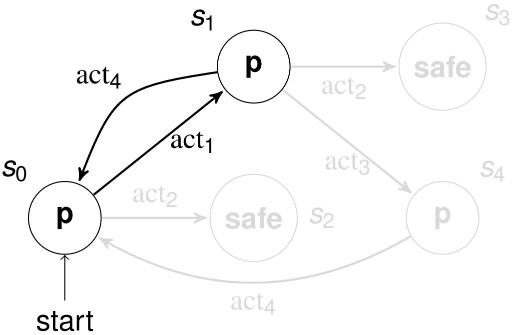
Action Synthesis: the Problem in a Nutshell



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

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

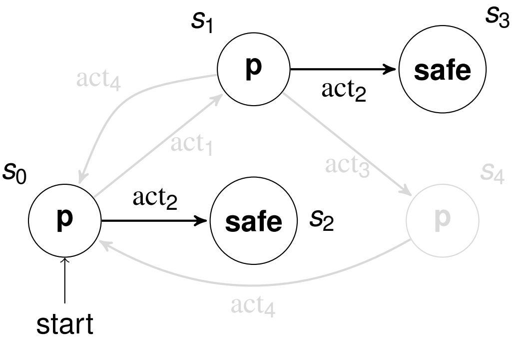
Action Synthesis: the Problem in a Nutshell



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

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

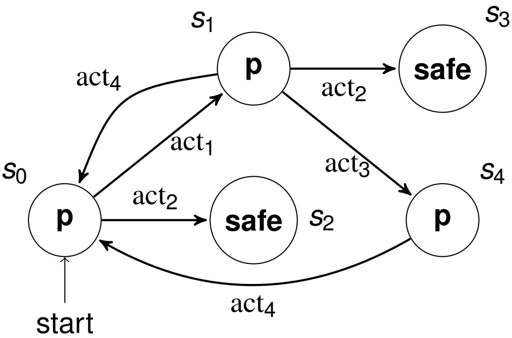
Action Synthesis: the Problem in a Nutshell



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

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

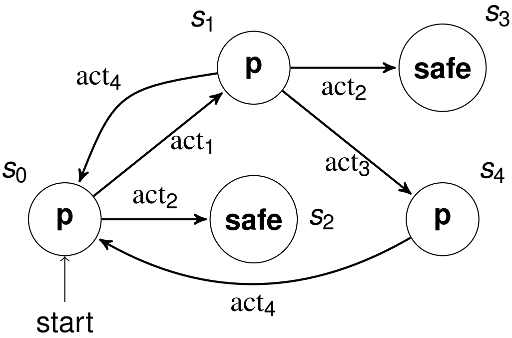
Action Synthesis: the Problem in a Nutshell



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

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

Action Synthesis: the Problem in a 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: describe all Y, Z s.t.: $\mathrm{start} \models A_Y G(\mathbf{p} \wedge E_Z F\mathbf{safe})$

Action Synthesis: Formal Statement

$\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{T}, \mathcal{L})$, $\phi \in$ 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)

Note: deciding if $f_\phi(s) \neq \emptyset$ is NP-complete (*emptiness problem*)

Action Synthesis: Building f_ϕ (1)

Recursive construction:
For all $s \in \mathcal{S}$:

$$f_p(s) = \begin{cases} \text{ActVals} & \text{if } p \in \mathcal{L}(s), \\ \emptyset & \text{if } p \notin \mathcal{L}(s). \end{cases} \tag{I}$$

(if p labels s then all valuations are OK, otherwise – none is)

$$f_{\phi \vee \psi}(s) = f_\phi(s) \cup f_\psi(s) \tag{II}$$

($\phi \vee \psi$ holds in s under υ iff either holds in s: sum up solutions)

$$f_{\neg\phi}(s) = \text{ActVals} \setminus f_\phi(s) \tag{III}$$

($\neg\phi$ holds in s under υ iff ϕ doesn't: complement of $f_\phi(s)$)

Action Synthesis: Building f_ϕ (2)

Parametric preimage of $f : \mathcal{S} \to 2^{\text{ActVals}}$ w.r.t. $Y \in \text{ActVars}$
$\text{parPre}^{\exists}_{Y}(f) : \mathcal{S} \to 2^{\text{ActVals}}$ s. t. for each $s \in \mathcal{S}$:

$$\text{parPre}^{\exists}_{Y}(f)(s) = \left\{ \upsilon \mid \exists_{s' \in \mathcal{S}} \; \exists_{a \in \upsilon(Y)} \; s \xrightarrow{a} s' \wedge \upsilon \in f(s') \right\}$$

Recursive construction, continued:

$$f_{E_Y X \phi}(s) = \text{parPre}^{\exists}_{Y}(f_\phi)(s) \qquad \text{(IV)}$$

($E_Y X \phi$ holds in s under υ iff ϕ holds in a $\upsilon(Y)$-successor of s)

Action Synthesis: Building f_ϕ (3)

Recursive construction, continued – CTL-like fixpoints:

$$E_Y G^{\omega} \phi \equiv \phi \wedge E_Y X E_Y G^{\omega} \phi \tag{V}$$

$$E_Y(\phi U \psi) \equiv \psi \vee (\phi \wedge E_Y X E_Y(\phi U \psi)) \tag{VI}$$

Algorithm 1 $Synth_{EG^{\omega}}\,(f_\phi, Y)$

Output: $f_{E_Y G^{\omega} \phi} \in \left(2^{\mathrm{ActVals}}\right)^{\mathcal{S}}$

$f := f_\phi$; $h := \emptyset$
while $f \neq h$ **do**
  $h := f$
  $f := f_\phi \cap \mathrm{parPre}^{\exists}_{Y}(h)$
end while
return f

Algorithm 2 $Synth_{EU}\,(f_\phi, f_\psi, Y)$

Output: $f_{E_Y \phi U \psi} \in \left(2^{\mathrm{ActVals}}\right)^{\mathcal{S}}$

$f := f_\psi$; $h := \emptyset$
while $f \neq h$ **do**
  $h := f$
  $f := f_\psi \cup (f_\phi \cap \mathrm{parPre}^{\exists}_{Y}(h))$
end while
return f

Action Synthesis: Building f_ϕ (4)

Recursive construction, concluded:

$$E_Y G\phi \equiv \phi \wedge \big(E_Y X E_Y G\phi \vee \neg E_Y X\mathbf{true}\big) \qquad \text{(VII)}$$

($s \models_\upsilon \neg E_Y X\mathbf{true}$ implies deadlock in s)

Algorithm 3 $\mathit{Synth}_{EG}(f_\phi, Y)$

Input: $f_\phi \in \left(2^{\mathrm{ActVals}}\right)^{\mathcal{S}}$

$f := f_\phi$; $h := \emptyset$
$D := f_{\phi \wedge \neg E_Y X\mathbf{true}}$
while $f \neq h$ **do**
    $h := f$
    $f := (f_\phi \cap \mathrm{parPre}^{\exists}_Y(h)) \cup D$
end while
return f

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

$B_0 := \mathit{False}$; $B_1 := \mathit{False}$

Process 0

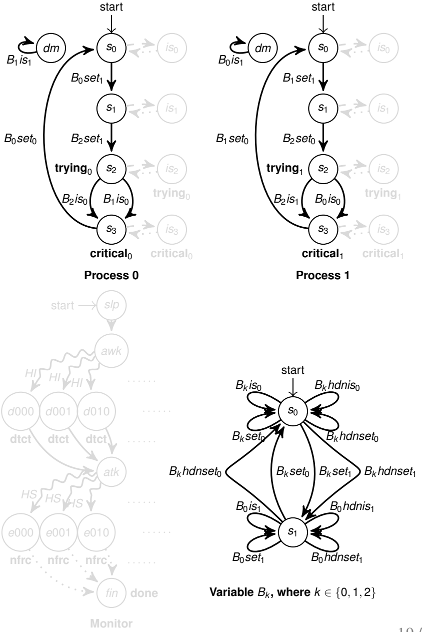
$B_0 := \mathit{True}$
$B_2 := \mathit{True}$
while $B_1 = \mathit{True}$ and $B_2 = \mathit{True}$ **do**
    pass {busy wait}
end while
{critical section}
$B_0 := \mathit{False}$

Process 1

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

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



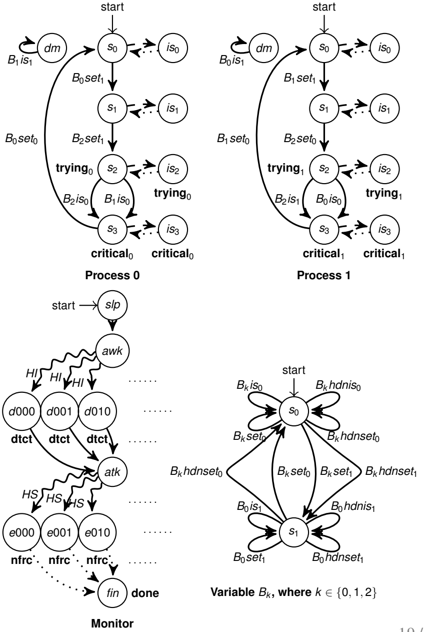
Evaluation: Peterson's Algorithm – Modelling Malicious Monitor (1)

- normal operation:

  basic properties verified

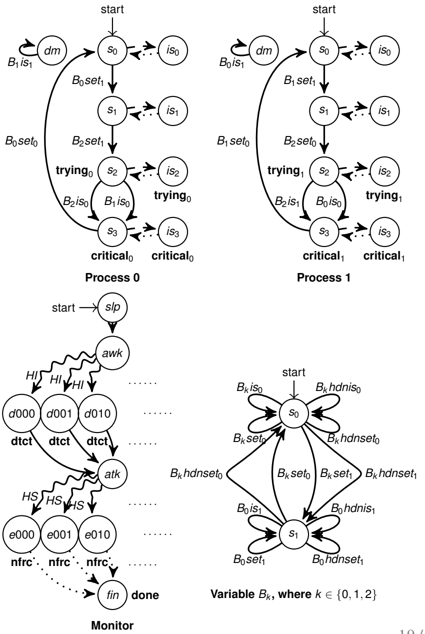
- interrupt request:
  1. freeze processes
  2. inspect variables
  3. play with variables
  4. un-freeze

- 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?"*

$$\phi_{dtct} = E_{\mathcal{A}_{\text{norm}}} F A_Y G(\text{dtct} \implies (\text{trying}_0 \vee \text{trying}_1 \vee \text{critical}_0 \vee \text{critical}_1)) \wedge E_Y F \text{dtct}$$

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?"*

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

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

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

Evaluation: Peterson's Algorithm – Modelling Malicious Monitor (3)

Question: *"can the monitor test and set shared variables s.t. if the test is positive then after return from interrupt and in future both processes simultaneously attempt to enter critical section?"*

$$\phi_{\mathsf{nfrcAF}} = E_{\mathcal{A}_{\mathsf{norm}}} F A_Y G(\mathsf{nfrc} \implies A_{\{irqret\}} X A_{\mathcal{A}_{\mathsf{norm}}} F (\mathsf{trying}_0 \land \mathsf{trying}_1)) \land E_Y F \mathsf{done}.$$

Answer: keep setting $B_0 := B_1 := 1$ to get a 50% success rate.

if $(B_0, B_1, B_2) \in \{(0,0,0),(0,0,1),(0,1,1),(1,0,0)\}$ **then**
  set (B_0, B_1) to $(1,1)$
end if

(0.08 sec. param. vs 79.36 sec. naïve)

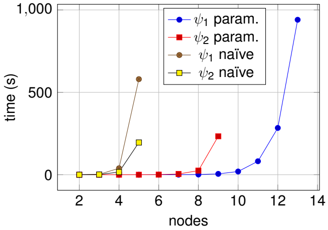
Evaluation: Faulty Train Gate Controller

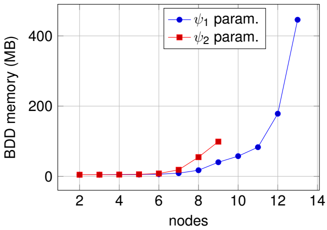
Model: k **Trains** attempt to enter single-train **Tunnel.**
Communication – red / green signals. One channel is faulty.

Properties:

$$\psi_1 = A_Y G(\neg \bigvee_{1 \le i < l \le k} (\mathrm{in_i} \wedge \mathrm{in_l})) \wedge \bigwedge_{1 \le i \le k} E_Y F \mathrm{in_i}$$

$$\psi_2 = E_Y F A_Y G\big((\bigwedge_{1 \le i \le k} \neg \mathrm{in_i}) \wedge \mathrm{green}\big)$$





Property	**Speedup** (naïve/parametric time)			
	2 trains	3 trains	4 trains	5 trains
ψ_1	76.0	463.59	4021.68	17378.02
ψ_2	48.96	276.01	703.97	1553.73

Evaluation: Generic Pipeline Paradigm

Model: k **Nodes** attempt build a processing **Pipeline** by selective synchronisation with up to four neighbours.

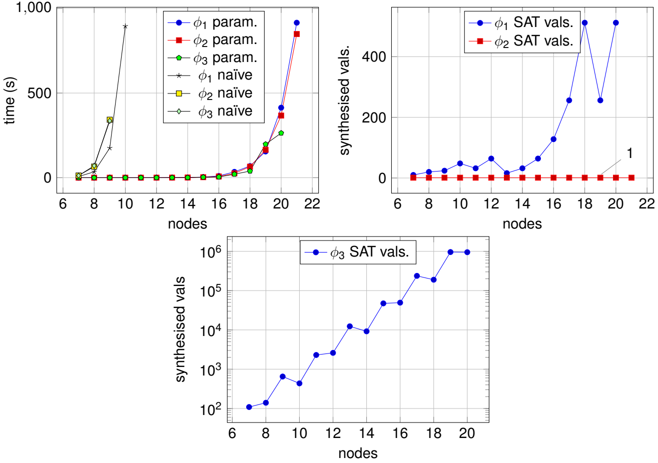
Properties:

$$\phi_1 = A_Y F(\bigwedge_{1 \leq i \leq \lfloor \frac{k}{2} \rfloor} \mathrm{out_i} \wedge \bigwedge_{\lceil \frac{k}{2} \rceil < j \leq k} \mathrm{in_j})$$
$$\phi_2 = A_Y G A_Y F(\bigwedge_{1 \leq i \leq k} \mathrm{in_i})$$
$$\phi_3 = E_Y F A_Y G(\bigwedge_{1 \leq i \leq \lceil \frac{k}{2} \rceil} \mathrm{in_{2i-1}} \wedge \bigwedge_{1 \leq i \leq \lfloor \frac{k}{2} \rfloor} \mathrm{out_{2i}})$$

Property	**Speedup** (naïve/parametric time)			
	7 processes	8 processes	9 processes	10 processes
ϕ_1	1402.60	4115.96	9171.02	22669.83
ϕ_2	1202.53	3265.79	8723.40	$>$ 12344.49†
ϕ_3	2985.93	7979.04	18633.09	$>$ 34531.71†

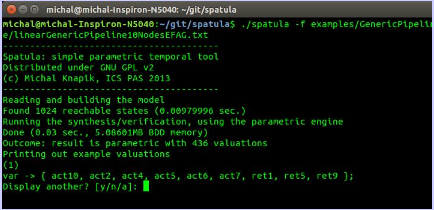
(† - the naïve approach exceeded set timeout of 15 minutes)

Evaluation: Generic Pipeline Paradigm, ct'd



SPATULA: BDD-based pmARCTL synthesis and verification

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

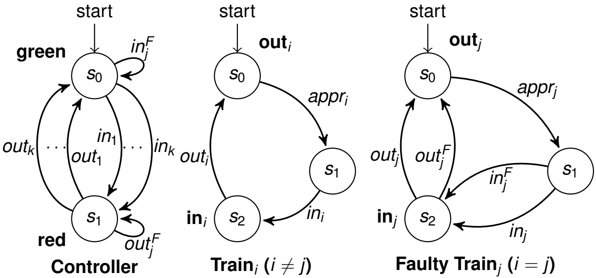



```
module Controller:
  trainsNo = k;
  faultyTrainNo = j;

  /* 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

Thank you

FTGC details

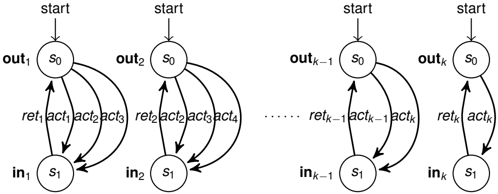


Properties:

(1) $\psi_1 = A_Y G(\neg \bigvee_{1 \leq i < l \leq k}(\mathrm{in_i} \wedge \mathrm{in_l})) \wedge \bigwedge_{1 \leq i \leq k} E_Y F \mathrm{in_i}$: *it is not possible for any pair of trains to be in the tunnel at the same time, and each train will eventually be in the tunnel;*

(2) $\psi_2 = E_Y F A_Y G\big((\bigwedge_{1 \leq i \leq k} \neg \mathrm{in_i}) \wedge \mathrm{green}\big)$, *it is possible for the system to execute in such a way that at some state, in all the possible executions of the system, all the trains remain outside the tunnel while the controller remains in the* **green** *state.*

GPPP details



Properties:

(1) $\phi_1 = A_Y F(\bigwedge_{1\leq i\leq\lfloor\frac{k}{2}\rfloor} \mathrm{out_i} \wedge \bigwedge_{\lceil\frac{k}{2}\rceil<j\leq k} \mathrm{in_j})$: *the configuration in which the first half of the nodes is in* **out**- *and the other half is in* **in** *states is unavoidable;*

(2) $\phi_2 = A_Y G A_Y F(\bigwedge_{1\leq i\leq k} \mathrm{in_i})$: *the configuration with all the nodes simultaneously in their* **in** *states appears infinitely often or ends a path;*

(3) $\phi_3 = E_Y F A_Y G(\bigwedge_{1\leq i\leq\lceil\frac{k}{2}\rceil} \mathrm{in_{2i-1}} \wedge \bigwedge_{1\leq i\leq\lfloor\frac{k}{2}\rfloor} \mathrm{out_{2i}})$: *the configuration such that the odd nodes are in their* **in**- *and the even are in their* **out** *states becomes persistent starting from some state in the future.*