Prime-Event Structures for Partial-Order Reduction and Abstract Interpretation

César Rodríguez [1,2]

[1] Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, France

[2] Diffblue Ltd., Oxford, UK

Dutch Model Checking Day ’18, Utrecht, 21 June 2018

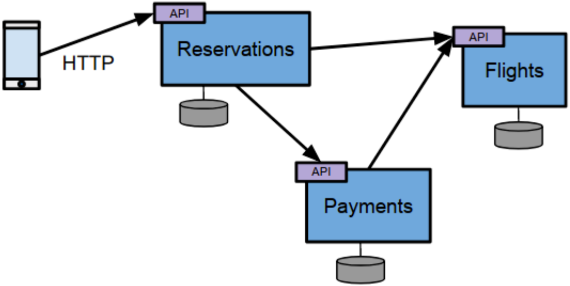
R&D at Diffblue

- Mission: automate all traditional coding tasks
- Founded by Daniel Kroening (CBMC) and Peter Schrammel 2y ago
- Develop verification/testing tools by applying existing and new research

Four tools developed:

Deeptest	Security scanner	Semantic refactoring	Microservice regression testing
Automated generation of unit tests.	Automated detection of security vulnerabilities.	Modernizes old code via synthesis of equivalent code snippets.	Generates regression tests for distributed systems.

Microservice Regression Testing



- Challenge: very large systems, difficult to comprehend for developers
- Approaches: BMC, static and dynamic analyses, PORs, taint analysis, fuzz testing

- Constantly hiring!

Model Checking

System $\models$ Specification

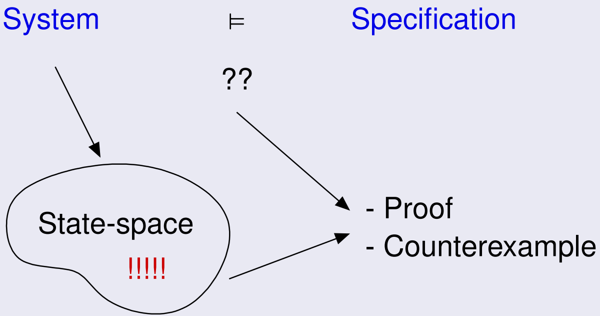
??

- Proof
- Counterexample

Sources of state-space explosion

- Concurrency
- Nondeterminism
- Data
- Unboundedness. . .

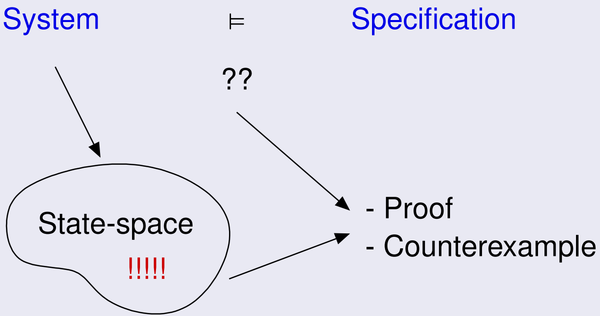
Model Checking



Sources of state-space explosion

- Concurrency
- Nondeterminism
- Data
- Unboundedness...

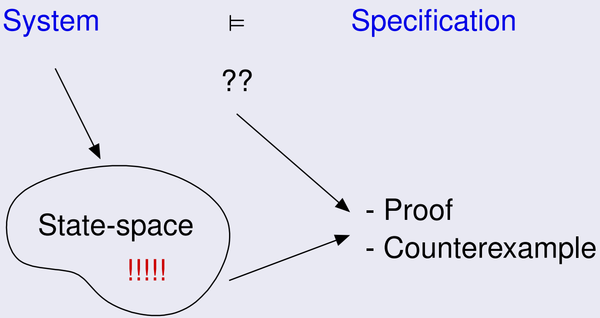
Model Checking



Sources of state-space explosion

- Concurrency $\rightarrow$ Partial-order reduction and unfoldings
- Nondeterminism
- Data
- Unboundedness. . .

Model Checking



Sources of state-space explosion

- Concurrency $\rightarrow$ Partial-order reduction and unfoldings
- Nondeterminism $\searrow$
- Data $\rightarrow$ Abstract interpretation
- Unboundedness... $\nearrow$

Partial-Order Reductions (PORs) and Unfoldings

POR: large family of techniques, interleaving semantics

- Scope: explicit-state, independence-based PORs for reachability
- Stubborn sets [Valmari 91], ample sets [Peled 93], persistent sets [Godefroid 96]

Unfoldings: partial-order semantics + algorithms (from the 90s)

- Mainly for Petri nets
- Processes [Petri 66], event structures [Winskel 87], finite prefixes [McMillan 92]

Quite independent fields of research for the last 20 years

To what extent both algorithms

1. exploit the same source of reduction?
2. can be used to mutually improve each other?

Independence

A transition system is a tuple $M := \langle \Sigma, \rightarrow, A, s_0 \rangle$ consisting on

- Σ, a set of states
- $\rightarrow \subseteq \Sigma \times A \times \Sigma$, a transition relation
- A, the set of actions
- s_0, the initial state

Independence

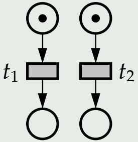
A transition system is a tuple $M := \langle \Sigma, \rightarrow, A, s_0 \rangle$ consisting on

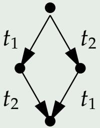
- Σ, a set of states
- $\rightarrow \subseteq \Sigma \times A \times \Sigma$, a transition relation
- A, the set of actions
- s_0, the initial state

Definition (Independence)

Relation $\diamond \subseteq A \times A$ is an independence relation in M if it is symmetric, irreflexive and when $a \diamond b$, then:

- Firing action a neither enables nor disables action b, and vice versa.
- Firing ab and ba (if possible) produces the same state.





Trace Equivalence

Let $\diamond \subseteq A \times A$ be an independence relation on A.

Definition (Mazurkiewicz trace equivalence)

Given two strings $\sigma, \sigma' \in A^*$ we have that

$$\sigma \equiv_\diamond \sigma'$$

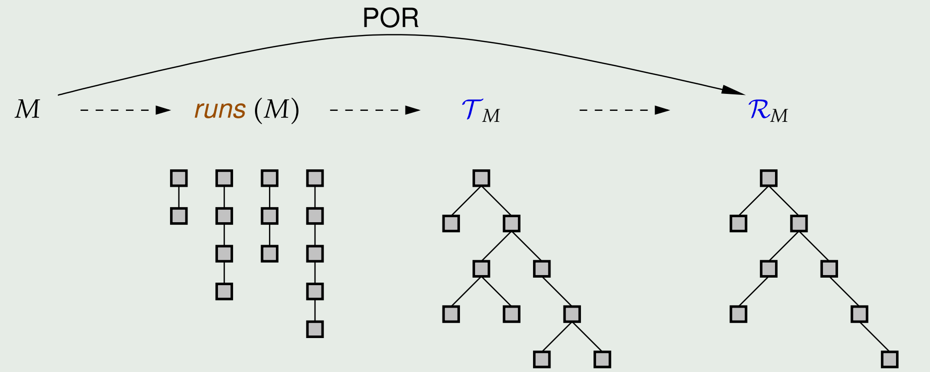
if it is possible to rewrite σ into σ' by swapping adjacent actions related by $\diamond$.

Remark

If $\sigma \equiv_\diamond \sigma'$ then necessarily $\mathit{state}(\sigma) = \mathit{state}(\sigma')$.

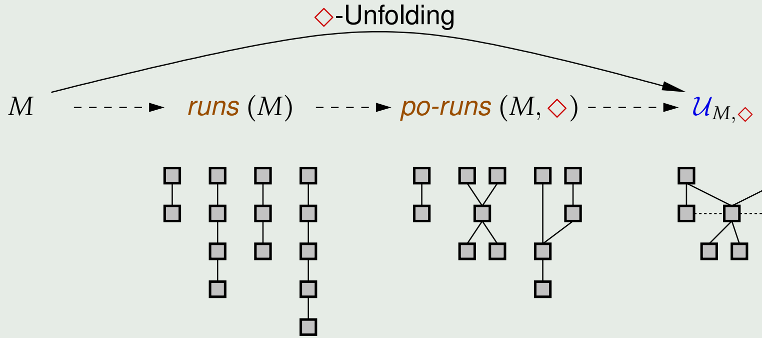
Each equivalence class of $\equiv_\diamond$ uniquely corresponds some A-labelled partial-order.

Independence-based Partial-Order Reductions (conceptually)



- Sound: explores at least one run within each equivalence class of $\equiv_\diamond$
- Optimal: explores no more than one such run

The Unfolding Approach (conceptually) [CONCUR'15]



- Each partial order corresponds to one equivalence class of $\equiv_\diamond$
- How do we bound together multiple partial orders?

Prime-Event Structures (PES)

A labelled prime event structure is a tuple $\langle E, <, \#, \lambda \rangle$ where

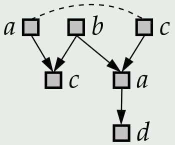
- E is the set of events, labelled by $\lambda \colon E \to A$
- $e < e'$ iff e' occurs $\Rightarrow$ e occurs before (causality)
- $e \# e'$ iff e and e' cannot occur in same execution (conflict)

A configuration is any set $\mathcal{C} \subseteq E$ s.t:

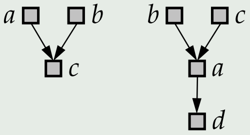
- if $e \in \mathcal{C}$ and $e' < e$, then $e' \in \mathcal{C}$ (causally closed)
- no two events in $\mathcal{C}$ are in conflict (conflict free)

Intuition: a configuration represents a (partially-ordered) execution of a system.

A PES:



Two configurations:

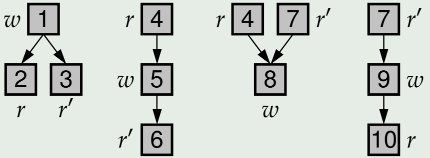


Unfolding Example

w	r	r'
`x=1`	`y=x`	`z=x`

$A = \{w, r, r'\}$

$\diamond = \{(r, r'),\ (r', r)\}$

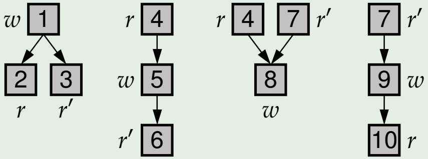


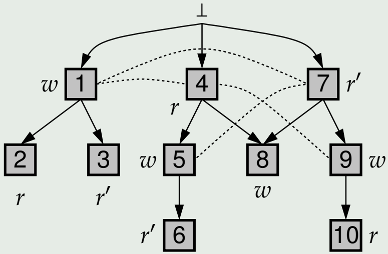
- Some equivalence classes in $\equiv_\diamond$ are not singletons

w	r	r'
`x=1`	`y=x`	`z=x`

$A = \{w, r, r'\}$

$\diamond = \{(r, r'),\ (r', r)\}$





- Some equivalence classes in $\equiv_\diamond$ are not singletons

Overview so far

$$\begin{tikzcd}
 & \mathcal{U}_{M,\diamond} \\
P \arrow[r, leftrightarrow] & M \arrow[u]
\end{tikzcd}$$

So far: parametric definition of the PES semantics for M under $\diamond$.

- Every execution of M is the interleaving of exactly 1 configuration of $\mathcal{U}_{M,\diamond}$.

Overview so far

$$
\begin{tikzcd}
 & \mathcal{U}_{M,\diamond} \\
P \arrow[r, leftrightarrow] & M \arrow[u]
\end{tikzcd}
$$

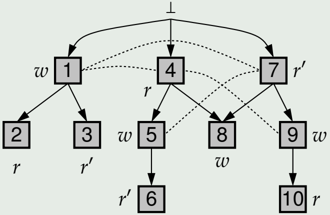
So far: parametric definition of the PES semantics for M under $\diamond$.

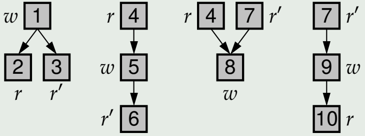
- Every execution of M is the interleaving of exactly 1 configuration of $\mathcal{U}_{M,\diamond}$.

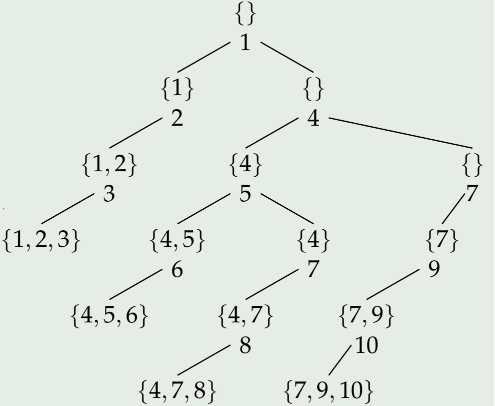
Next: POR algorithm to construct $\mathcal{U}_{M,\diamond}$, one configuration at a time.

- Super-optimal: can explore fewer executions than Mazurkiewicz traces

w	r	r'
`x=1`	`y=x`	`z=x`


⊥
w 1
4
7 r'
r
2
3
w 5
8
9 w
r
r'
w
r' 6
10 r


w 1
2
3
r
r'
r 4
w 5
r' 6
r 4
7 r'
8
w
7 r'
9 w
10 r


{}
1
{1}
{}
2
4
{1,2}
{4}
{}
3
5
7
{1,2,3}
{4,5}
{4}
{7}
6
7
9
{4,5,6}
{4,7}
{7,9}
8
10
{4,7,8}
{7,9,10}

Termination, Completeness, Optimality

For terminating systems (acyclic state-space), the algorithm:

- Always stops (termination)
- Explores at least once every maximal configuration of $\mathcal{U}_{M,\diamond}$ (completeness)
- Explores at most once any maximal configuration (optimality)

What about non-terminating systems?

- Next: we use cutoff events to prune infinite configurations
- This makes the algorithm super-optimal!

Cutoffs – Intuitions

```
while (1):                          while (1):
  lock(m)                             lock(m)
  if (buf < MAX): buf++               if (buf > MIN): buf--
  unlock(m)                           unlock(m)
```

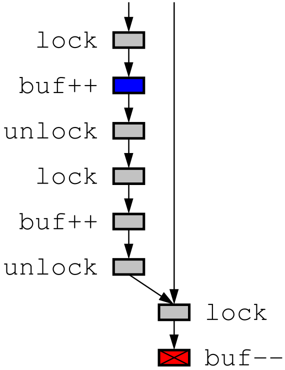
- $\overbrace{l,\ b+,\ u,\ \ l,\ b+,\ u}^{\textit{Thread 1}},\ \overbrace{l,\ \color{red}{b-}}^{\textit{Thread 2}}$ $\qquad (m = 0, b = 1)$
- $\overbrace{l,\ \color{blue}{b+}}^{\textit{Thread 1}}$

Cutoffs – Intuitions

```
while (1):                        while (1):
  lock(m)                           lock(m)
  if (buf < MAX): buf++             if (buf > MIN): buf--
  unlock(m)                         unlock(m)
```

- $\overbrace{l,\ b+,\ u,\ \ l,\ b+,\ u,}^{\textit{Thread 1}}\ \overbrace{l,\ \color{red}{b-}}^{\textit{Thread 2}}$
- $\overbrace{l,\ \color{blue}{b+}}^{\textit{Thread 1}}$

$(m = 0, b = 1)$



Experiments — Non-acyclic State-Space

Benchmark		Nidhugg				Poet			
Name	$\|P\|$	b	$\|I\|$	$\|B\|$	$t(s)$	$\|E\|$	$\|E_{\mathtt{cut}}\|$	$\|\Omega\|$	$t(s)$
Szymanski	3	--	103	0	0.07	1121	313	159	0.36
Dekker	3	10	199	0	0.11	217	14	21	0.07
Lamport	3	10	32	0	0.06	375	28	30	0.12
Peterson	3	10	266	0	0.11	175	15	20	0.05
Pgsql	3	10	20	0	0.06	51	8	4	0.03
Rwlock	5	10	2174	14	0.83	<7317	531	770	12.29
Rwlock(2)*	5	2	--	--	7.88	--	--	--	0.40
Prodcons	4	5	756756	0	332.62	3111	568	386	5.00
Prodcons(2)	4	5	63504	0	38.49	640	25	15	1.61

Remarks:

- Poet: complete verification; Nidhugg: bounded verification
- Significant, sometimes dramatic, reduction in nr. of executions

Overview so far: PES Semantics + Optimal POR Algorithm

$$\begin{tikzcd} & \mathcal{U}_{M,\diamond} \\ P \arrow[r, leftrightarrow] & M \arrow[u, "\text{Super-optimal POR}"] \end{tikzcd}$$

So far:

- Parametric definition of the PES semantics for M under $\diamond$.
- Super-optimal POR algorithm to construct it.

Overview so far: PES Semantics + Optimal POR Algorithm

$$\begin{tikzcd} & \mathcal{U}_{M,\diamond} \\ P \arrow[r, leftrightarrow] & M \arrow[u, "\text{Super-optimal POR}"] \end{tikzcd}$$

So far:

- Parametric definition of the PES semantics for M under $\diamond$.
- Super-optimal POR algorithm to construct it.

Observations: [CAV'18]

1. Computing alternatives (finding next branch) is NP-complete
2. State-of-the-art, non-optimal Source DPOR [Abdulla et al. 14] rarely explores redundant executions

Example: Exponentially Many Redundant Executions

Example instance with $n = 3$:

writer 0	writer 1	writer 2	count	master
`x[0] = 7`	`x[1] = 8`	`x[2] = 9`	`c = 1`	`i = c`
			`c = 2`	`x[i] = 1`

When generalized to n writer threads:

- $\mathcal{O}(n)$ Mazurkiewicz traces, but SDPOR explores $\mathcal{O}(2^n)$ interleavings
- Reason: SDPOR disregards "coupled" races

Can we get polynomial-time alternatives and avoid the exponential blowup?

- Yes, Quasi-Optimal POR!

Key idea: approximation algorithm via a user-defined constant k

- Compute k-partial alternatives, which revert at least k races (P-time)
- Experimentally: very low values of k suffice to achieve optimal exploration

Details are in the paper!

Implementation and Experiments: Tool DPU [CAV'18]

New tool DPU (Dynamic Program Unfolding)

- Deterministic C programs, POSIX threads
- Clang front-end, LLVM JIT engine

https://github.com/cesaro/dpu

Goals of the experiments:

- Evaluate the values of k necessary to achieve optimal exploration
- Compare with SDPOR
- Evaluate DPU on system code (two Debian packages) for bug finding

Experimental results:

- SDPOR performance can be strongly reduced by redundant executions
- QPOR more resilient to complex synchronization than SDPOR
- DPU can handle large codes (40KLOC)
- Orders of magnitude faster than state-of-the-art testing tools (Maple)

So far: PES Semantics + Optimal & Quasi-Optimal POR

$$\begin{tikzcd} & \mathcal{U}_{M,\diamond} \\ P \arrow[r, leftrightarrow] & M \arrow[u, "\text{Super-optimal POR}"', "\text{Quasi-optimal POR}"] \end{tikzcd}$$

So far:

- Unfolding-based super-optimal POR [CONCUR'15]
- Unfolding-based quasi-optimal POR [CAV'18]

So far: PES Semantics + Optimal & Quasi-Optimal POR

$$\begin{tikzcd} & \mathcal{U}_{M,\diamond} \\ P \arrow[r, leftrightarrow] & M \arrow[u, "\begin{array}{c}\text{Super-optimal POR}\\ \text{Quasi-optimal POR}\end{array}"'] \end{tikzcd}$$

So far:

- Unfolding-based super-optimal POR [CONCUR’15]
- Unfolding-based quasi-optimal POR [CAV’18]

None of the above works tackle data explosion. Next:

- Integration of abstract interpretation into the POR [CAV’17]

Example: Explosion due to Concurrent and Data

```
while (++i < 100)
  if (*)
    break;
k += i;
```

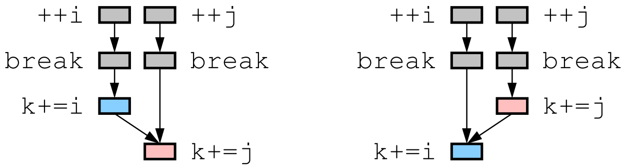
```
while (++j < 150)
  if (*)
    break;
k += j;
```

How many Mazurkiewicz traces does this program have?

Example: Explosion due to Concurrent and Data

```
while (++i < 100)          while (++j < 150)
  if (*)                     if (*)
    break;                     break;
k += i;                    k += j;
```

One iteration 1st thread, one iteration 2nd thread:

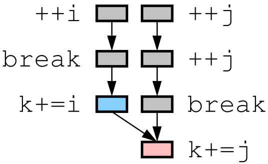


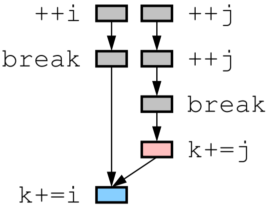
Example: Explosion due to Concurrent and Data

```
while (++i < 100)
  if (*)
    break;
k += i;
```

```
while (++j < 150)
  if (*)
    break;
k += j;
```

One iteration 1st thread, two iterations 2nd thread:

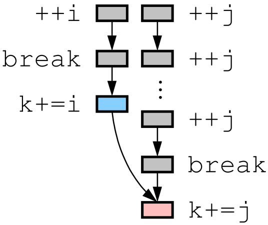


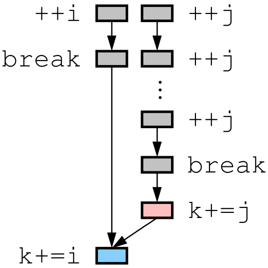


Example: Explosion due to Concurrent and Data

`while (++i < 100)`
`  if (*)`
`    break;`
`k += i;`
	`while (++j < 150)`
	`  if (*)`
	`    break;`
	`k += j;`

One iteration 1st thread, 150 iterations 2nd thread:

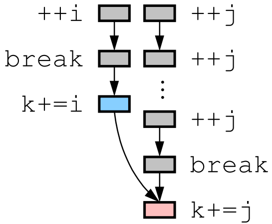


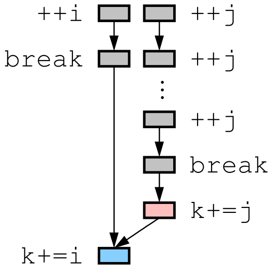


Example: Explosion due to Concurrent and Data

```
while (++i < 100)          while (++j < 150)
  if (*)                     if (*)
    break;                     break;
k += i;                    k += j;
```

One iteration 1st thread, 150 iterations 2nd thread:



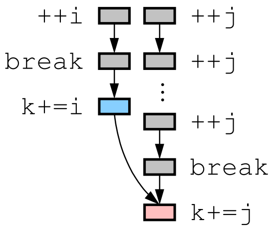


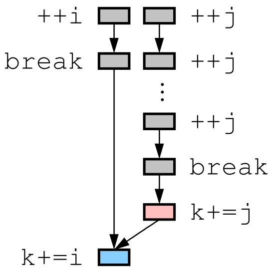
- So how many Mazurkiewicz traces does the program have?

Example: Explosion due to Concurrent and Data

```
while (++i < 100)          |   while (++j < 150)
  if (*)                   |     if (*)
    break;                 |       break;
k += i;                    |   k += j;
```

One iteration 1st thread, 150 iterations 2nd thread:

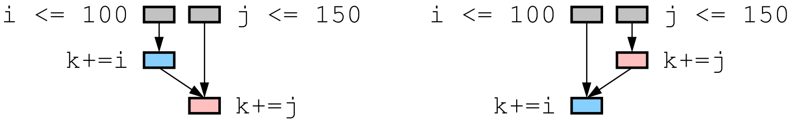




- So how many Mazurkiewicz traces does the program have?
- 100 local iterations × 150 local iterations × 2 ways for threads to interact.

```
while (++i < 100)        |   while (++j < 150)
  if (*)                 |     if (*)
    break;               |       break;
k += i;                  |   k += j;
```

Idea: merging the results of local computation before the global statements, mimicking the fixpoint analysis of an abstract interpreter.



- Next: how to handle states via an abstraction domain.

Introducing a Concrete and Abstract Domain

$$
\begin{tikzcd}
& \mathcal{U}_{M,\diamond} & & \\
P \arrow[r, leftrightarrow] & M \arrow[u] \arrow[r, leftrightarrow] & \mathcal{C}_M \arrow[r, shift right, "\alpha"'] & \mathcal{D} \arrow[l, shift right, "\gamma"']
\end{tikzcd}
$$

$M := \langle \Sigma, \rightarrow, A, s_0 \rangle$ is a transition system:

- Σ: set of states
- $\rightarrow \subseteq \Sigma \times A \times \Sigma$: transition relation
- A: program statements
- s_0: initial state

Introducing a Concrete and Abstract Domain

$$
\begin{tikzcd}
 & \mathcal{U}_{M,\diamond} & & \\
P \arrow[r, leftrightarrow] & M \arrow[u] \arrow[r, leftrightarrow] & \mathcal{C}_M \arrow[r, shift right, "\alpha"'] & \mathcal{D} \arrow[l, shift right, "\gamma"']
\end{tikzcd}
$$

$M := \langle \Sigma, \to, A, s_0 \rangle$ is a transition system:

- Σ: set of states
- $\to \subseteq \Sigma \times A \times \Sigma$: transition relation
- A: program statements
- s_0: initial state

$\mathcal{D} := \langle D, \sqsubseteq, F, d_0 \rangle$ is an abstraction domain:

- D is a set of abstract states
- $\sqsubseteq$ in $D \times D$ is the abstraction order
- $F \subseteq D \to D$ is a set of transformers
- $d_0 \in D$ is the abstract initial state

Introducing a Concrete and Abstract Domain

$$\begin{tikzcd}
& \mathcal{U}_{M,\diamond} & & \\
P \arrow[r, leftrightarrow] & M \arrow[u] \arrow[r, leftrightarrow] & \mathcal{C}_M \arrow[r, shift right, "\alpha"'] & \mathcal{D} \arrow[l, shift right, "\gamma"']
\end{tikzcd}$$

$M := \langle \Sigma, \rightarrow, A, s_0 \rangle$ is a transition system:

- Σ: set of states
- $\rightarrow \subseteq \Sigma \times A \times \Sigma$: transition relation
- A: program statements
- s_0: initial state

$\mathcal{C}_M := \langle D, \sqsubseteq, F, d_0 \rangle$ is the collecting semantics:

- $D := 2^\Sigma$ are the concrete states
- $\sqsubseteq := \subseteq$ is the lattice order
- F is the set of concrete transformers
- $d_0 := \{s_0\}$ is the initial state

For every statement $a \in A$, set F contains a concrete transformer

$$f_a(S) := \{s' \in \Sigma \colon \text{ for some } s \in S \text{ we have } s \xrightarrow{a} s'\},$$

Introducing a Concrete and Abstract Domain

$$
\begin{tikzcd}
 & \mathcal{U}_{M,\diamond} & & \\
P \arrow[r, leftrightarrow] & M \arrow[u] \arrow[r, leftrightarrow] & \mathcal{C}_M \arrow[r, shift right, "\alpha"'] & \mathcal{D} \arrow[l, shift right, "\gamma"']
\end{tikzcd}
$$

$M := \langle \Sigma, \to, A, s_0 \rangle$ is a transition system:

- Σ: set of states
- $\to \subseteq \Sigma \times A \times \Sigma$: transition relation
- A: program statements
- s_0: initial state

$\mathcal{C}_M := \langle D, \sqsubseteq, F, d_0 \rangle$ is the collecting semantics:

- $D := 2^{\Sigma}$ are the concrete states
- $\sqsubseteq := \subseteq$ is the lattice order
- F is the set of concrete transformers
- $d_0 := \{s_0\}$ is the initial state

For every statement $a \in A$, set F contains a concrete transformer

$$f_a(S) := \{s' \in \Sigma \colon \text{ for some } s \in S \text{ we have } s \xrightarrow{a} s'\},$$

and $\mathcal{C}_M \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}$ is a Galois connection.

Weak Independence

$$\begin{tikzcd}
& \mathcal{U}_{M,\diamond} & & \\
P \arrow[r, leftrightarrow] & M \arrow[u] \arrow[r, leftrightarrow] & \mathcal{C}_M \arrow[r, shift right, "\alpha"'] & \mathcal{D} \arrow[l, shift right, "\gamma"'] \\
& \diamond \arrow[r, dashed] & \diamond_1 \arrow[r, dashed] & \diamond_2 \\
& & \text{weak} & \text{inherited}
\end{tikzcd}$$

Definition (Weak Independence)

A relation $\diamond_1 \in F \times F$ on the set of transformers is a weak independence if it is symmetric, reflexive, and for any $f \diamond_1 g$ we get

$$f(g(d)) = g(f(d))$$

for any abstract state $d \in D$ reachable in the domain.

$$
\begin{tikzcd}
 & \mathcal{U}_{M,\diamond} & \mathcal{U}_{\mathcal{C}_M,\diamond_1} & \mathcal{U}_{\mathcal{D},\diamond_2} \\
P \arrow[r, leftrightarrow] & M \arrow[u] \arrow[r, leftrightarrow] & \mathcal{C}_M \arrow[u] \arrow[r, shift right, "\alpha"'] & \mathcal{D} \arrow[u] \arrow[l, shift right, "\gamma"'] \\
 & \diamond \arrow[r, dashed] & \diamond_1 \arrow[r, dashed] & \diamond_2 \\
 & & \text{weak} & \text{inh.}
\end{tikzcd}
$$

Collecting semantics:

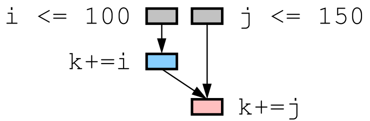
- Every execution σ of M has a unique representative configuration in $\mathcal{U}_{\mathcal{C}_M,\diamond_1}$.
- Every interleaving of a configuration $\mathcal{C}$ of $\mathcal{U}_{\mathcal{C}_M,\diamond_1}$ s.t. $state(\mathcal{C}) \neq \bot$ is a run of M.

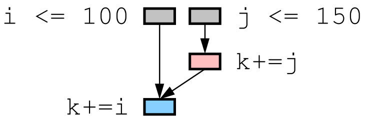
Abstract unfolding:

- Every execution σ of M has a unique representative configuration in $\mathcal{U}_{\mathcal{D},\diamond_2}$.

$$
\begin{tikzcd}
 & \mathcal{U}_{M,\diamond} & & & \\
P \arrow[r, leftrightarrow] & M \arrow[u] \arrow[r, leftrightarrow] & \mathcal{C}_M \arrow[r, shift right, "\alpha"'] & \mathcal{D} \arrow[l, shift right, "\gamma"'] \arrow[r, shift right] & \hat{\mathcal{D}} \arrow[l, shift right, "\text{TLF}"'] \\
 & \diamond \arrow[r, dashed] & \diamond_1 \arrow[r, dashed] & \diamond_2 & \\
 & & \text{weak} & \text{inh.} &
\end{tikzcd}
$$

```
while (++i < 100)              while (++j < 150)
  if (*)                         if (*)
    break;                         break;
k += i;                        k += j;
```





$$
\begin{tikzcd}
& \mathcal{U}_{M,\diamond} & & & \\
P \arrow[r, leftrightarrow] & M \arrow[u] \arrow[r, leftrightarrow] & \mathcal{C}_M \arrow[r, shift right, "\alpha"'] & \mathcal{D} \arrow[l, shift right, "\gamma"'] \arrow[r, shift right] & \hat{\mathcal{D}} \arrow[l, shift right, "\text{TLF}"'] \\
& \diamond \arrow[r, dashed] & \diamond_1 \arrow[r, dashed] & \diamond_2 & \\
& & \text{weak} & \text{inh.} &
\end{tikzcd}
$$

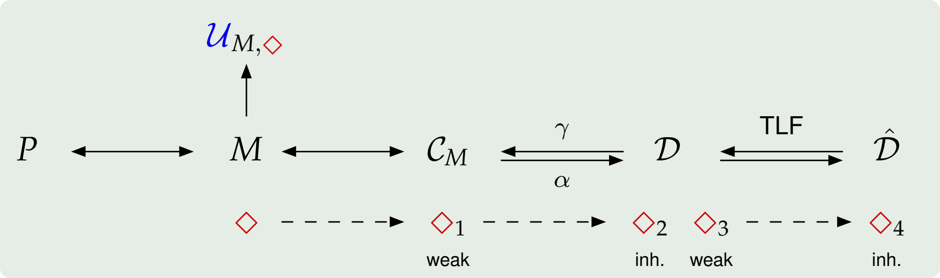
$$\begin{tikzcd}
 & \mathcal{U}_{M,\diamond} & & & \\
P \arrow[r, leftrightarrow] & M \arrow[u] \arrow[r, leftrightarrow] & \mathcal{C}_M \arrow[r, shift right, "\alpha"'] & \mathcal{D} \arrow[l, shift right, "\gamma"'] \arrow[r, shift right] & \hat{\mathcal{D}} \arrow[l, shift right, "\text{TLF}"'] \\
 & \diamond \arrow[r, dashed] & \diamond_1 \arrow[r, dashed] & \diamond_2 & \\
 & & \text{weak} & \text{inh.} &
\end{tikzcd}$$

- Partition transformers in $\mathcal{D}$ in two classes: global and local transformers.

Thread-Local Analysis: the Collapsing Domain [CAV'18]

$$
\begin{tikzcd}
 & \mathcal{U}_{M,\diamond} & & & \\
P \arrow[r, leftrightarrow] & M \arrow[u] \arrow[r, leftrightarrow] & \mathcal{C}_M \arrow[r, shift right, "\alpha"'] & \mathcal{D} \arrow[l, shift right, "\gamma"'] \arrow[r, shift right] & \hat{\mathcal{D}} \arrow[l, shift right, "\text{TLF}"'] \\
 & \diamond \arrow[r, dashed] & \underset{\text{weak}}{\diamond_1} \arrow[r, dashed] & \underset{\text{inh.}}{\diamond_2} &
\end{tikzcd}
$$

- Partition transformers in $\mathcal{D}$ in two classes: global and local transformers.
- For each global transformer f we define a collapsing transformer $\hat{f} : D \to D$ as:
  - Apply an off-the-shelf abstract interpreter restricted to local transformers.
  - Apply the global transformer f.



- Partition transformers in $\mathcal{D}$ in two classes: global and local transformers.
- For each global transformer f we define a collapsing transformer $\hat{f} : D \to D$ as:
  - Apply an off-the-shelf abstract interpreter restricted to local transformers.
  - Apply the global transformer f.

$$
\begin{tikzcd}
 & \mathcal{U}_{M,\diamond} & & & \mathcal{U}_{\hat{\mathcal{D}},\diamond_4} \\
P \arrow[r, leftrightarrow] & M \arrow[u] \arrow[r, leftrightarrow] & \mathcal{C}_M \arrow[r, shift right, "\alpha"'] & \mathcal{D} \arrow[l, shift right, "\gamma"'] \arrow[r, shift right] & \hat{\mathcal{D}} \arrow[u] \arrow[l, shift right, "\text{TLF}"'] \\
 & \diamond \arrow[r, dashed] & \underset{\text{weak}}{\diamond_1} \arrow[r, dashed] & \underset{\text{inh.}}{\diamond_2} \quad \underset{\text{weak}}{\diamond_3} \arrow[r, dashed] & \underset{\text{inh.}}{\diamond_4}
\end{tikzcd}
$$

- Partition transformers in $\mathcal{D}$ in two classes: global and local transformers.
- For each global transformer f we define a collapsing transformer $\hat{f} : D \to D$ as:
  - Apply an off-the-shelf abstract interpreter restricted to local transformers.
  - Apply the global transformer f.

Example Thread-Local Fixpoints

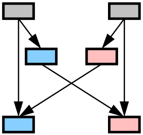
```
while (++i < 100)
  if (*)
    break;
k += i;
```

```
while (++j < 150)
  if (*)
    break;
k += j;
```

$i, j, k \mapsto [0, 100], 0, 0$

$i, j, k \mapsto [0, 100], 0, [0, 100]$

$i, j, k \mapsto [0, 100], [0, 150], [0, 250]$



$i, j, k \mapsto 0, [0, 150], 0$

$i, j, k \mapsto 0, [0, 150], [0, 150]$

$i, j, k \mapsto [0, 100], [0, 150], [0, 250]$

Abstract intepreter on local code = thread-local fixpoint analysis = event merging

Experimental Results

Benchmark			APOET				ASTREEA		IMPARA			CBMC 5.6	
Name	P	A	$t(s)$	E	E_{cut}	W	$t(s)$	W	V	$t(s)$	N	V	$t(s)$
ATGC(3)	4	7	5.78	432	0	1	1.69	2	–	TO	–	S	6.6
ATGC(4)	5	7	132.08	7195	0	1	2.68	2	–	TO	–	S	20.22
COND	5	2	0.55	982	0	2	0.71	2	–	TO	–	S	34.39
FMAX(5,3)	2	8	0.56	85	11	0	1.50	2	–	TO	–	–	TO
FMAX(2,4)	2	8	3.38	277	43	0	<2	2	–	TO	–	–	TO
FMAX(2,6)	2	8	45.82	1663	321	0	<2	2	–	TO	–	–	TO
FMAX(2,7)	2	8	146.19	3709	769	0	1.87	2	–	TO	–	–	TO
FMAX(4,7)	2	8	285.23	6966	671	0	<2	2	–	TO	–	–	TO
LAZY	4	2	0.01	72	0	0	0.50	2	–	TO	–	S	3.59
LAZY*	4	2	0.01	72	0	1	0.49	2	–	TO	–	U	3.50
SIGMA	5	5	2.62	7126	0	0	0.43	0	–	TO	–	S	189.09
SIGMA*	5	5	2.64	7126	0	1	0.43	1	–	TO	–	U	141.35
TPOLL(2)*	3	11	1.23	141	7	1	1.97	2	U	0.64	80	–	TO
TPOLL(3)*	4	11	109.22	1712	90	2	3.77	3	U	0.72	113	–	TO

- ASTREEA: 6x false positives
- CBMC: TOs in 54% of the benchmarks
- IMPARA: TOs in 83% of the benchmarks

Collaborators

- Marcelo Sousa
- Huyen Nguyen
- Subodh Sharma
- Vijay D'Silva
- Daniel Kroening
- Laure Petrucci
- Camille Coti
- ...

$$
\begin{tikzcd}
 & \mathcal{U}_{M,\diamond} & \mathcal{U}_{\mathcal{C}_M,\diamond_1} & \mathcal{U}_{\mathcal{D},\diamond_2} & \mathcal{U}_{\hat{\mathcal{D}},\diamond_4} \\
P \arrow[r, leftrightarrow] & M \arrow[r, leftrightarrow] \arrow[u, "\begin{array}{r}\text{(Super-)Optimal POR}\\ \text{Quasi-optimal POR}\end{array}"'] & \mathcal{C}_M \arrow[u] \arrow[r, shift right, "\alpha"'] & \mathcal{D} \arrow[u] \arrow[l, shift right, "\gamma"'] \arrow[r, shift right] & \hat{\mathcal{D}} \arrow[u] \arrow[l, shift right, "\text{TLF}"'] \\
 & \diamond \arrow[r, dashed] & \diamond_1 \arrow[r, dashed] & \diamond_2 \quad \diamond_3 \arrow[r, dashed] & \diamond_4 \\
 & & \text{weak} & \text{inh.} \quad \text{weak} & \text{inh.}
\end{tikzcd}
$$

- Application to other models of computation
- Combination with AI: foundations for symbolic execution

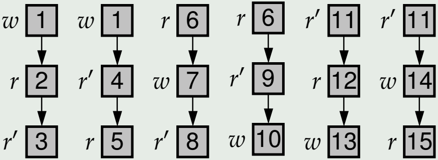
Extra Slides

Unfolding Example (Empty Independence)

w	r	r'
`x=1`	`y=x`	`z=x`

$A = \{w, r, r'\}$

$\diamond = \emptyset$

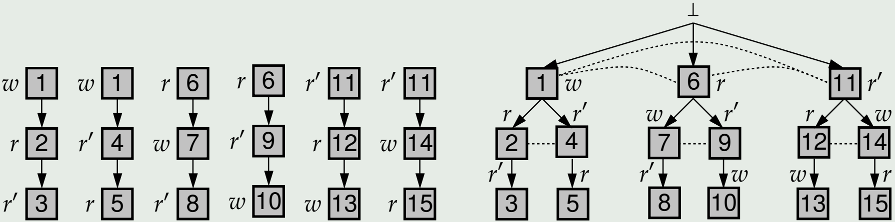


- All equivalence classes in $\equiv_\diamond$ are singletons

w	r	r'
`x=1`	`y=x`	`z=x`

$A = \{w, r, r'\}$

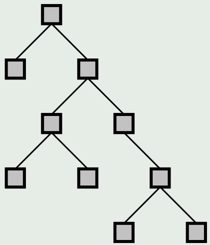
$\diamond = \varnothing$

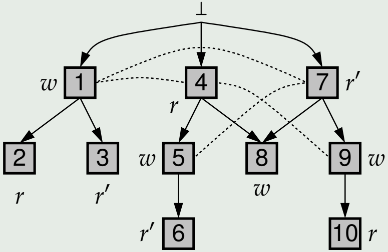


- All equivalence classes in $\equiv_{\diamond}$ are singletons

POR vs Unfoldings: 6 Algorithmic Differences

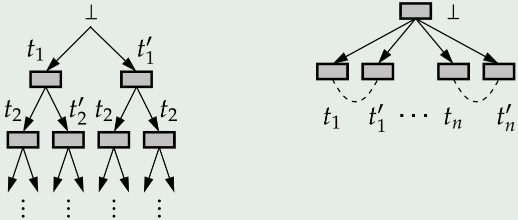
1. Unfolding extension is NP-complete; POR extension is constant-time





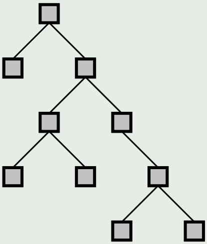
POR vs Unfoldings: 6 Algorithmic Differences

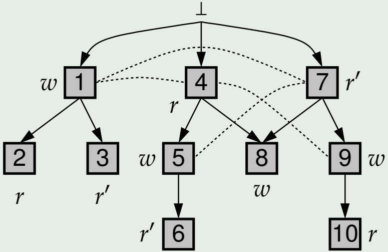
1. Unfolding extension is NP-complete; POR extension is constant-time
2. $\mathcal{R}_{M,\diamond}$ can be exponentially larger than $\mathcal{U}_{M,\diamond}$



POR vs Unfoldings: 6 Algorithmic Differences

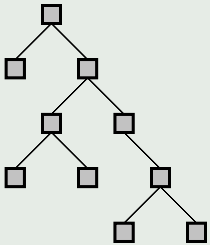
1. Unfolding extension is NP-complete; POR extension is constant-time
2. $\mathcal{R}_{M,\diamond}$ can be exponentially larger than $\mathcal{U}_{M,\diamond}$
3. Unfolding algorithms are inherently stateful; state-of-the-art DPORs are stateless
   - [Flanagan, Godefroid, POPL'05], [Abdulla et al., POPL'14]

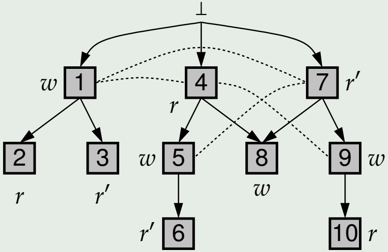




POR vs Unfoldings: 6 Algorithmic Differences

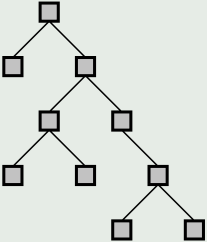
1. Unfolding extension is NP-complete; POR extension is constant-time
2. $\mathcal{R}_{M,\diamond}$ can be exponentially larger than $\mathcal{U}_{M,\diamond}$
3. Unfolding algorithms are inherently stateful; state-of-the-art DPORs are stateless
4. Dynamic POR: difficult to avoid repeated exploration of same states

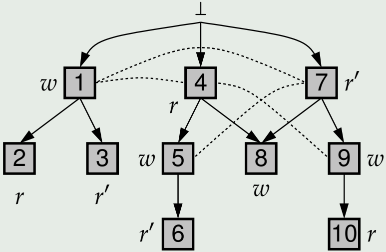




POR vs Unfoldings: 6 Algorithmic Differences

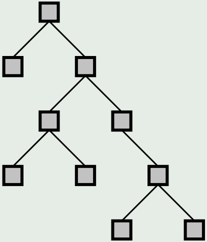
1. Unfolding extension is NP-complete; POR extension is constant-time
2. $\mathcal{R}_{M,\diamond}$ can be exponentially larger than $\mathcal{U}_{M,\diamond}$
3. Unfolding algorithms are inherently stateful; state-of-the-art DPORs are stateless
4. Dynamic POR: difficult to avoid repeated exploration of same states
5. Dynamic POR: difficult to handle non-terminating executions

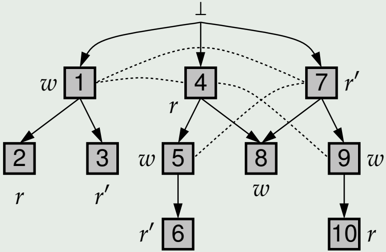




POR vs Unfoldings: 6 Algorithmic Differences

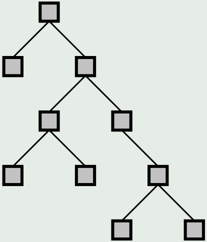
1. Unfolding extension is NP-complete; POR extension is constant-time
2. $\mathcal{R}_{M,\diamond}$ can be exponentially larger than $\mathcal{U}_{M,\diamond}$
3. Unfolding algorithms are inherently stateful; state-of-the-art DPORs are stateless
4. Dynamic POR: difficult to avoid repeated exploration of same states
5. Dynamic POR: difficult to handle non-terminating executions
6. Stateless PORs do not profit fom additional RAM

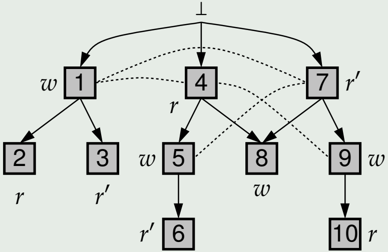




POR vs Unfoldings: 6 Algorithmic Differences

1. Unfolding extension is NP-complete; POR extension is constant-time
2. $\mathcal{R}_{M,\diamond}$ can be exponentially larger than $\mathcal{U}_{M,\diamond}$
3. Unfolding algorithms are inherently stateful; state-of-the-art DPORs are stateless
4. Dynamic POR: difficult to avoid repeated exploration of same states
5. Dynamic POR: difficult to handle non-terminating executions
6. Stateless PORs do not profit fom additional RAM





POR vs Unfoldings: 6 Algorithmic Differences

1. Unfolding extension is NP-complete; POR extension is constant-time
2. $\mathcal{R}_{M,\diamond}$ can be exponentially larger than $\mathcal{U}_{M,\diamond}$
3. Unfolding algorithms are inherently stateful; state-of-the-art DPORs are stateless
4. Dynamic POR: difficult to avoid repeated exploration of same states
5. Dynamic POR: difficult to handle non-terminating executions
6. Stateless PORs do not profit fom additional RAM

Unfolding-based POR (next slide)

A novel stateless POR exploration of unfolding semantics

- Retains advantages of both approaches
- (Super-)Optimal: can explore fewer executions than Mazurkiewicz traces
- Addresses all above points except (2)

Procedure `Explore` $(\mathcal{C}, D, A)$

if $state(\mathcal{C})$ *enables no event* **return**

e = some event enabled by $state(\mathcal{C})$, from A if possible

`Explore` $(\mathcal{C} \cup \{e\}, D, A \setminus \{e\})$

if *there is some* $J \in$ `Alt` $(\mathcal{C}, D \cup \{e\})$

`Explore` $(\mathcal{C}, D \cup \{e\}, J \setminus \mathcal{C})$

end

The set `Alt` $(\mathcal{C}, X)$ contains all configurations J such that:

- $J \cup \mathcal{C}$ is a configuration
- for all $e \in X$ there is some $e' \in J \cup C$ such that $e \# e'$

Benchmark		Nidhugg			Poet			
Name	$\|P\|$	$\|I\|$	$\|B\|$	$t(s)$	$\|E\|$	$\|E_{\mathrm{cut}}\|$	$\|\Omega\|$	$t(s)$
STF	3	6	0	0.06	121	0	6	0.06
STF*	3	--	--	0.05	--	--	--	0.03
SPIN08	3	84	0	0.08	2974	0	84	2.93
FIB	3	8953	0	3.36	<185K	0	8953	704
FIB*	3	--	--	0.74	--	--	--	133
CCNF(9)	9	16	0	0.05	49	0	16	0.06
CCNF(19)	19	512	0	0.28	109	0	512	22.0
SSB(1)	5	22	14	0.06	237	4	23	0.11
SSB(4)	5	336	103	0.15	2179	74	142	2.07
SSB(8)	5	2014	327	0.85	<12K	240	470	32.1

Remarks:

- Narrow, deep, relatively small unfoldings
- Half of the benchmarks display no concurrency (STF, SPIN08, Fib)
- In SSB we achieve a super-optimal exploration

Experiments: QPOR vs SDPOR [CAV'18]

Benchmark			DPU (k=1)		DPU (k=2)		DPU (k=3)		DPU (optimal)		NIDHUGG		
Name	Th	Confs	Time	SSB	Time	SSB	Time	SSB	Time	Mem	Time	Mem	SSB
DISP(5,4)	10	15K	58.5	105K	16.4	6K	10.3	213	**10.3**	87	109	33	115K
DISP(5,5)	11	151K	TO	–	476	53K	280	2K	**257**	729	TO	33	–
MPAT(6)	13	46K	50.6	0	N/A		N/A		73.2	214	**21.5**	33	0
MPAT(7)	15	645K	TO	–	TO	–	TO	–	TO	660	**359**	33	0
MPC(2,5)	8	60	0.6	560	0.4	0			**0.4**	38	2.0	34	3K
MPC(3,5)	9	3K	26.5	50K	3.0	3K	1.7	0	**1.7**	38	70.7	34	90K
MPC(4,5)	10	314K	TO	–	TO	–	391	30K	**296**	239	TO	33	–
MPC(5,5)	11	?	TO	–	TO	–	TO	–	TO	834	TO	34	–
PI(6)	7	720	**0.7**	0	N/A		N/A		0.7	39	123	35	0
PI(8)	9	40K	48.1	0	N/A		N/A		**42.9**	246	TO	34	–
POL(7,3)	14	3K	48.5	72K	2.9	1K	**1.9**	6	1.9	39	74.1	33	90K
POL(9,3)	16	5K	464	592K	9.5	5K	**4.8**	15	4.8	73	TO	33	–
POL(11,3)	18	10K	TO	–	27.2	12K	**9.7**	28	10.6	138	TO	33	–

- SDPOR performance can be strongly reduced by redundant executions
- More complex synchronization $\Longrightarrow$ higher k necesary for optimal exploration
- With few redundant executions QPOR can be faster than Optimal POR