Unfolding-Based Partial Order Reduction

César Rodríguez [1] Marcelo Sousa [2] Subodh Sharma [3] Daniel Kroening [2]

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

[2] Department of Computer Science, University of Oxford, UK

[3] Indian Institute of Technology Delhi, India

CONCUR'15, Madrid, 4 September 2015

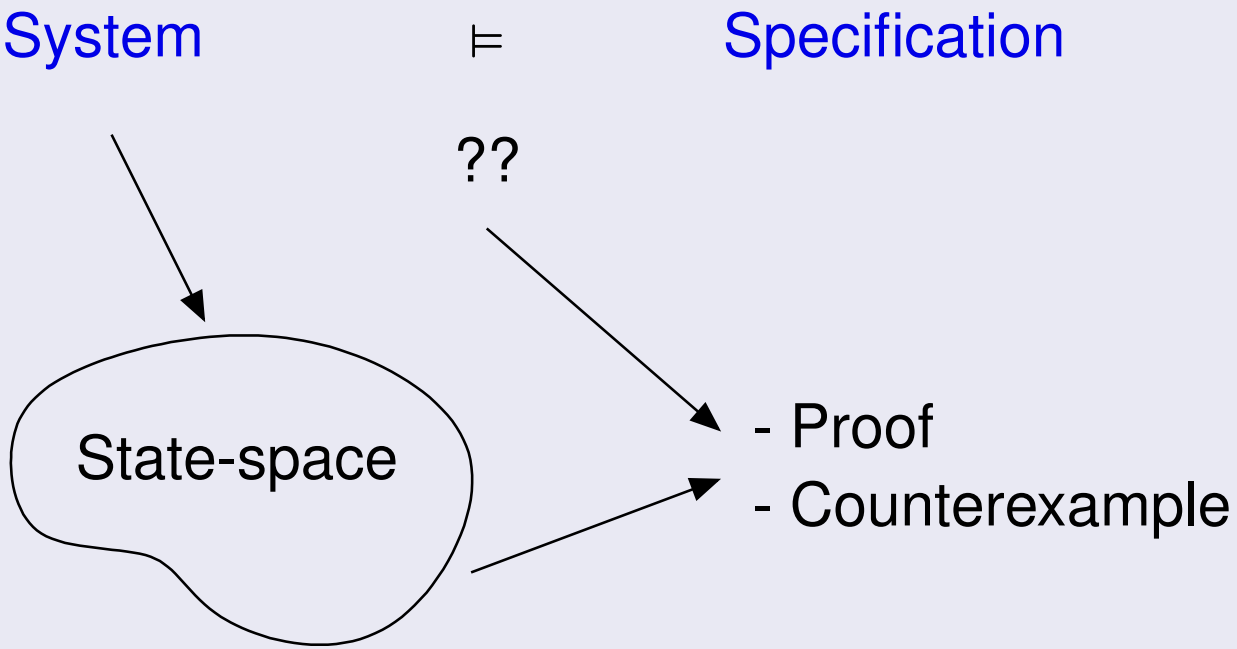
Model Checking

System $\models$ Specification

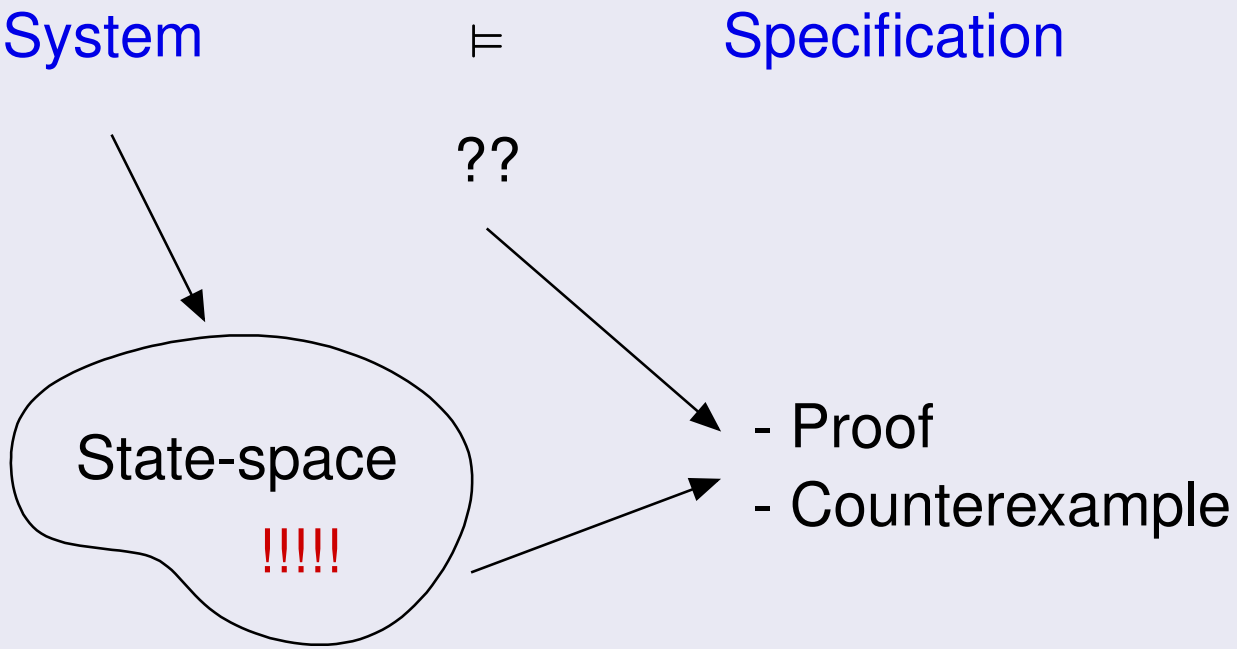
??

- Proof
- Counterexample

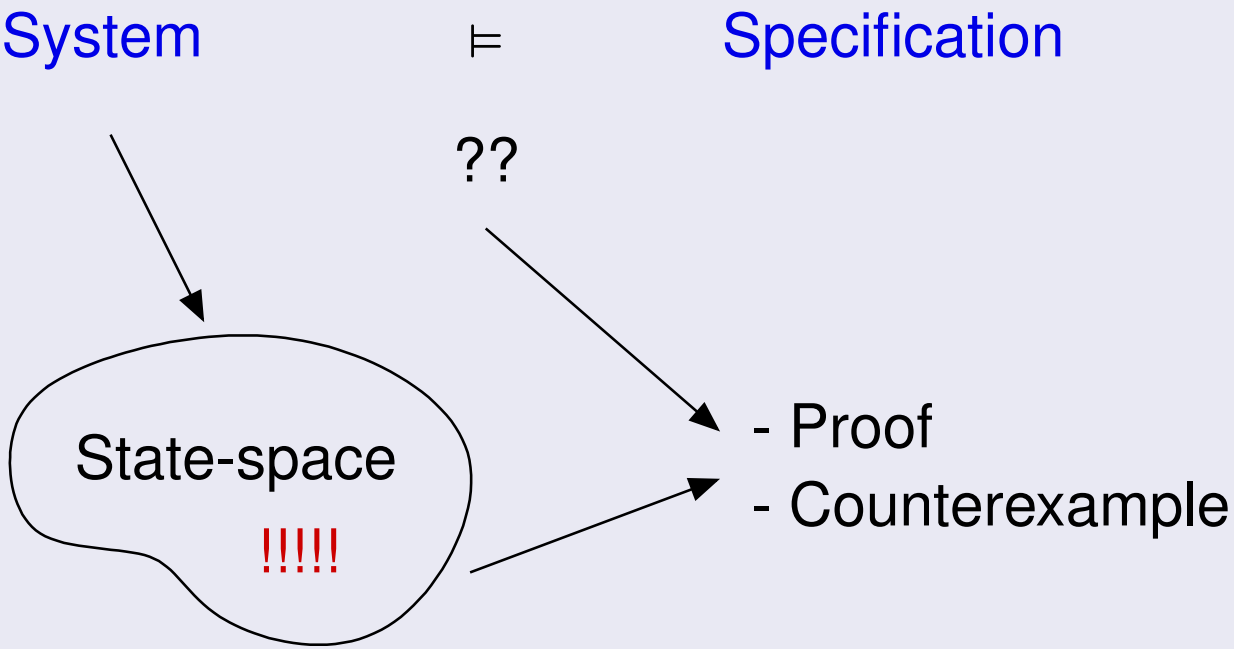
Model Checking



Model Checking



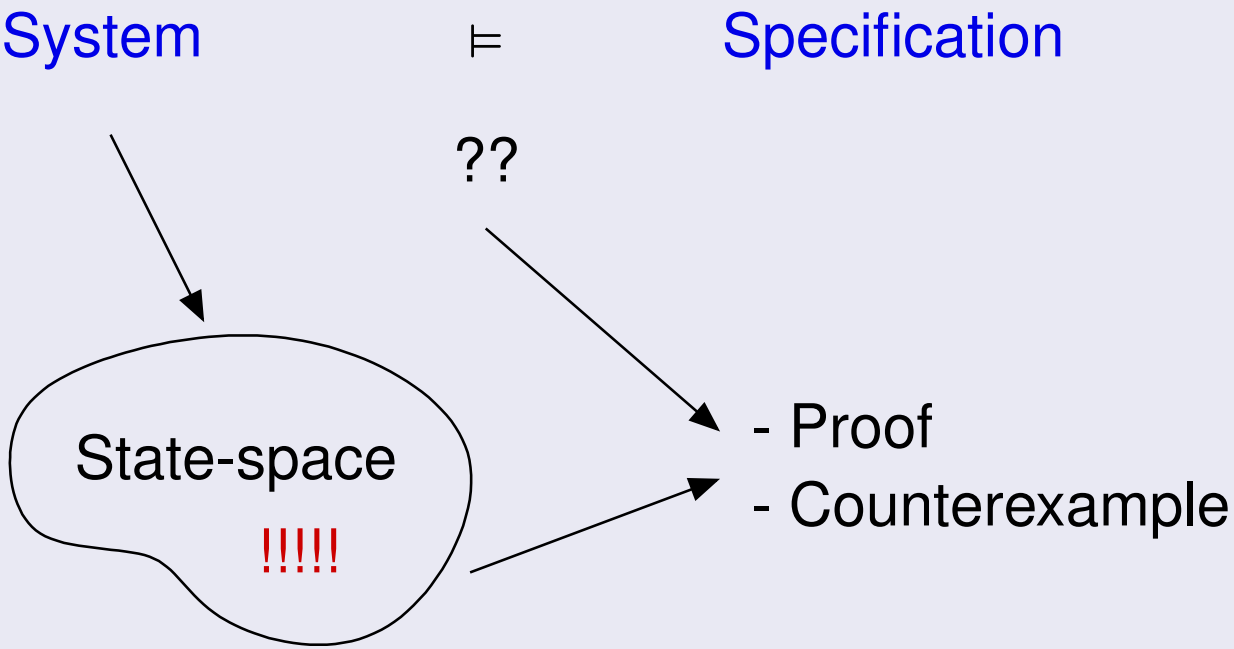
Model Checking



Sources of state-space explosion

- Concurrency
- Nondeterminism
- Data
- Unboundedness. . .

Model Checking



Sources of state-space explosion

- Concurrency ⟶ Partial-order reduction and unfoldings
- Nondeterminism
- Data
- Unboundedness. . .

Partial-Order Reductions (PORs) and Unfoldings

POR: large family of techniques, interleaving semantics

- Scope of this work: explicit-state, independence-based PORs for reachability
- Persistent sets [Godefroid 96], and early stubborn sets [Valmari 91]

Unfoldings: partial-order semantics

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

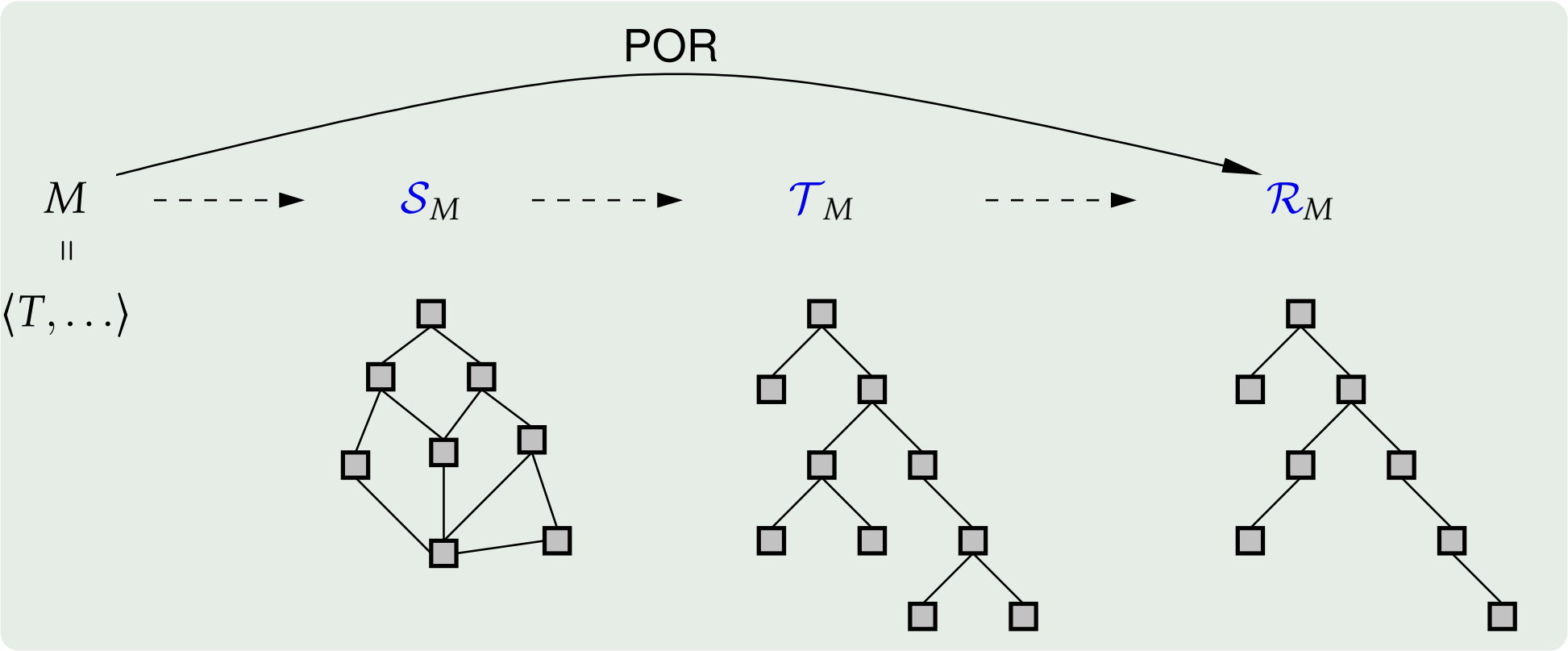
Quite independent fields of research for the last 20 years

To what extent both approaches

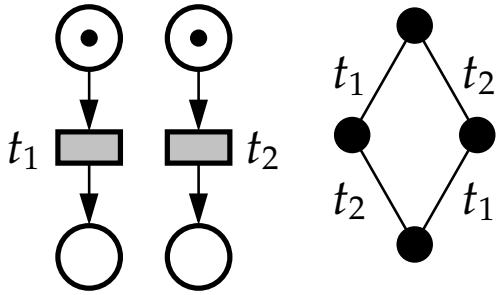
1. exploit the same source of reduction?
2. are just different exploration algorithms over the same underlying reduced space?

3. How can that be exploited to improve both methods?

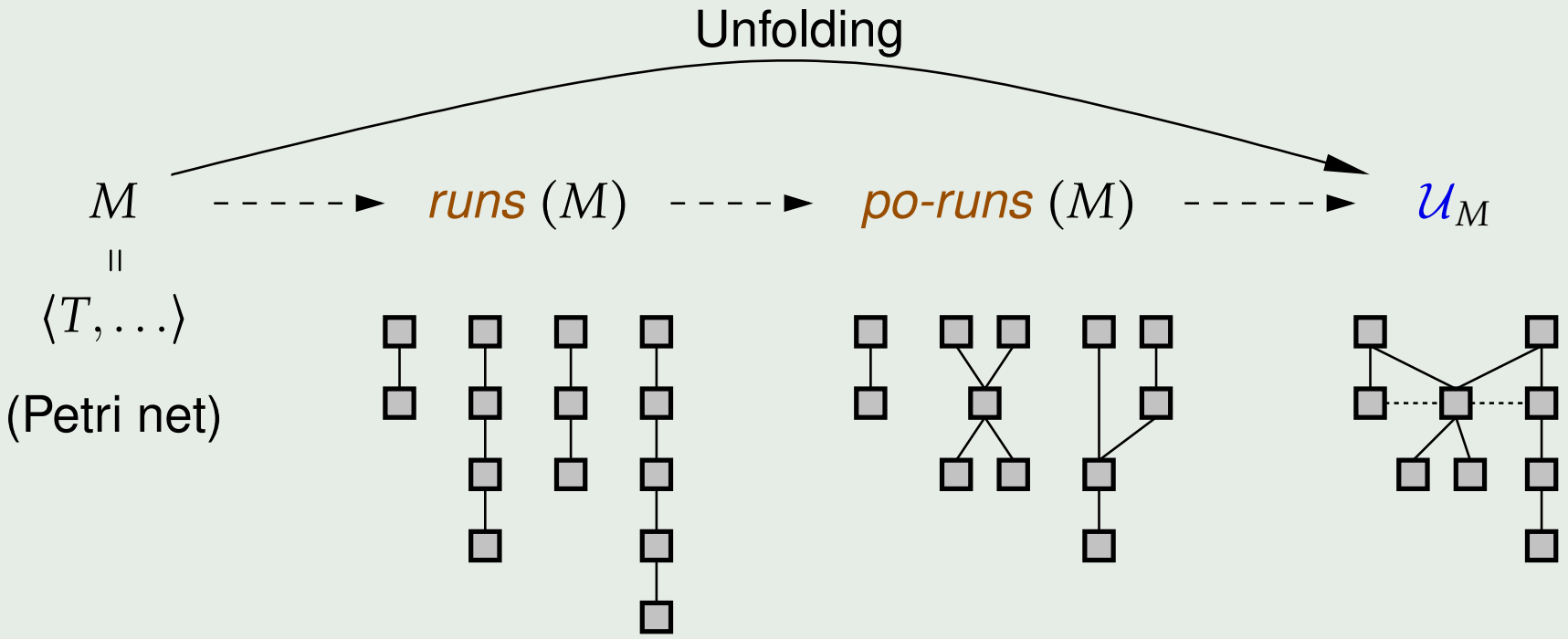
Partial-Order Reductions (conceptually)



- Independence relation $\diamond \subseteq T \times T$
- At least one $\diamond$-equivalent run to every terminating run
- POR is optimal if exactly one



The Unfolding Approach (conceptually)

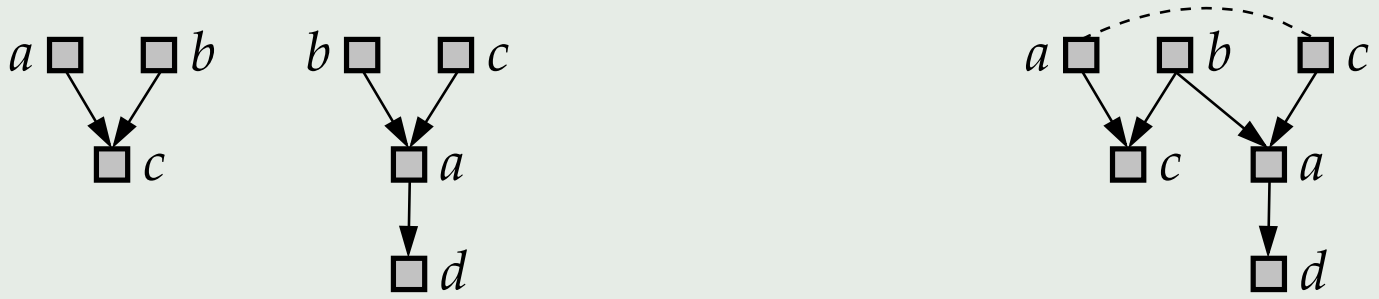


The Unfolding Approach (conceptually)

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

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

Intuition: a set of partial orders bound together by $\#$ in a prefix-sharing fashion:

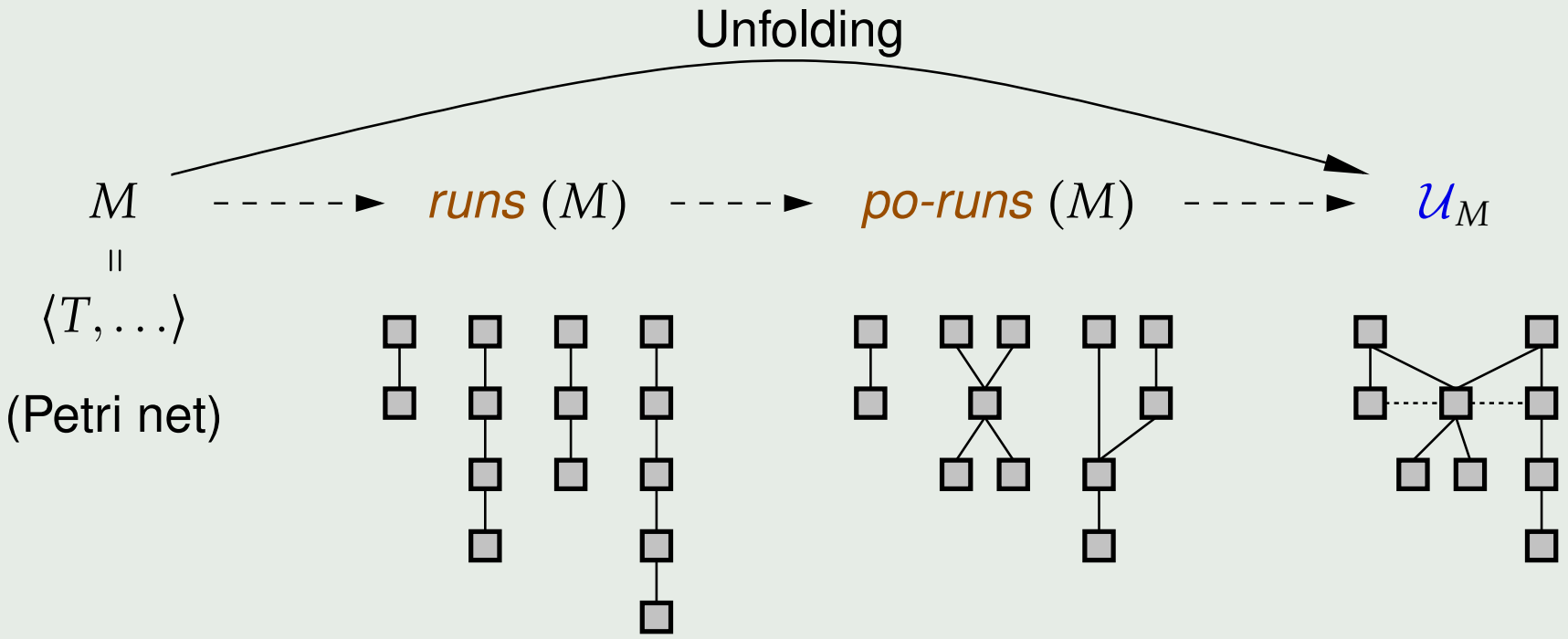


A configuration is any set $\mathcal{C} \subseteq E$ s.t:

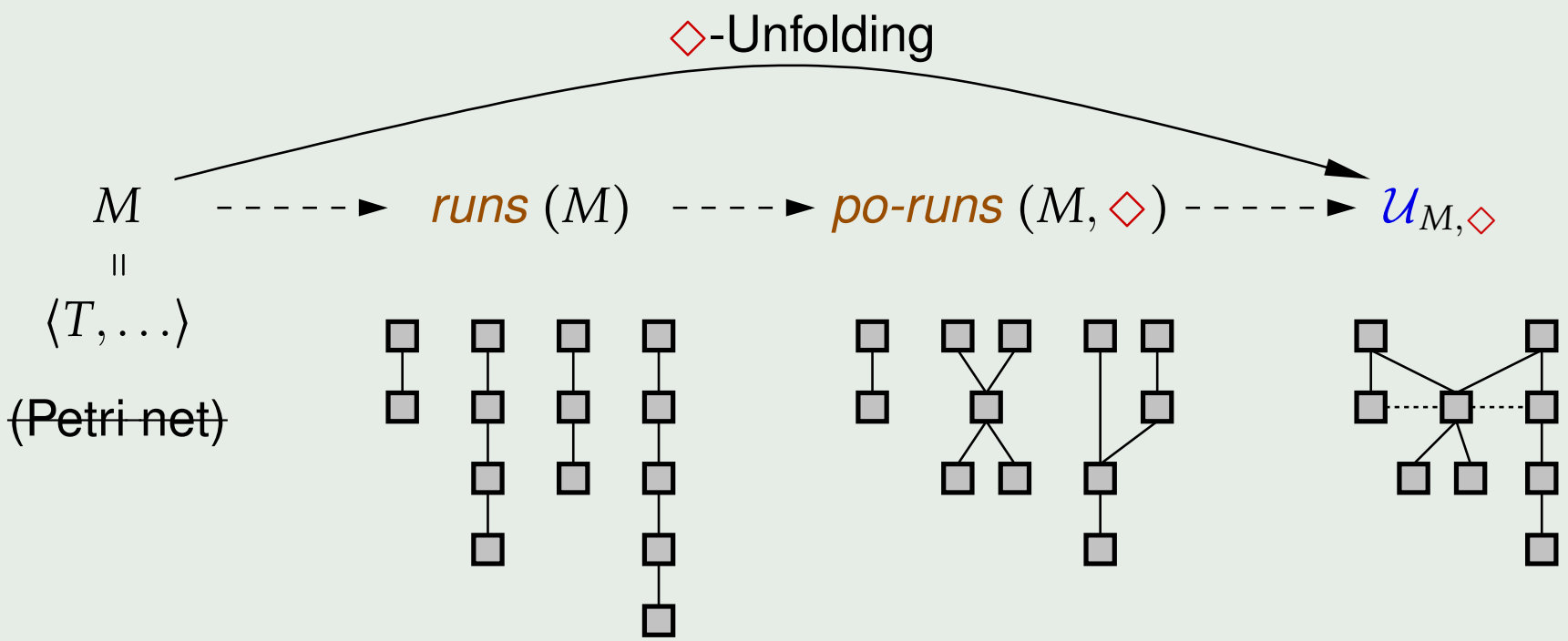
- if $e \in \mathcal{C}$ and $e' < e$, then $e' \in \mathcal{C}$ (causally closed)
- no two events in $\mathcal{C}$ are in conflict (conflict free)

Intuition: configuration $\equiv$ constituent partial order (or prefix of)

The Unfolding Approach (conceptually)



The Unfolding Approach (conceptually)



First contribution:

- Unfolding semantics for general model of computation $M := \langle T, \ldots \rangle$
- Parametric on user-defined independence relation $\diamond \subseteq T \times T$

Unfolding over User-Defined Independence Relation

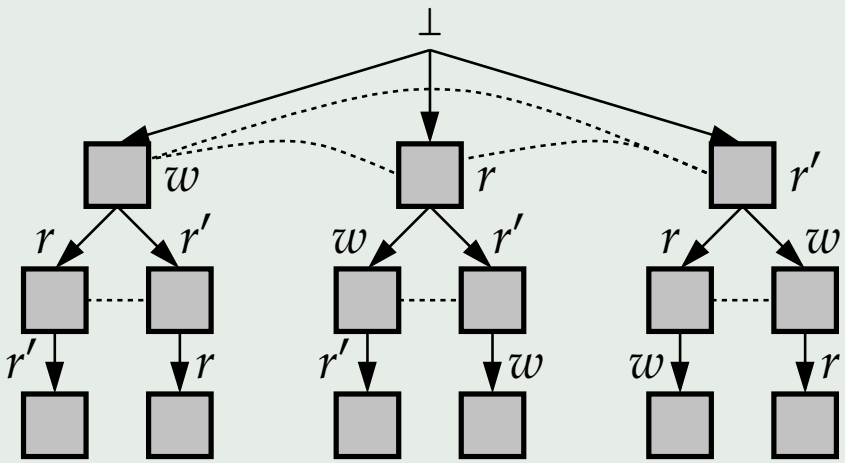
Let

- $M := \langle T, \ldots \rangle$ be a general model of computation
- $\diamond \subseteq T \times T$ a valid unconditional independence relation

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

$T = \{w, r, r'\}$

$\diamond = \varnothing$



Theorem (Soundness & Uniqueness)

- *The unfolding $\mathcal{U}_{M,\diamond}$ exists and is unique.*
- *Every sequential execution is represented by exactly 1 configuration.*

Unfolding over User-Defined Independence Relation

Let

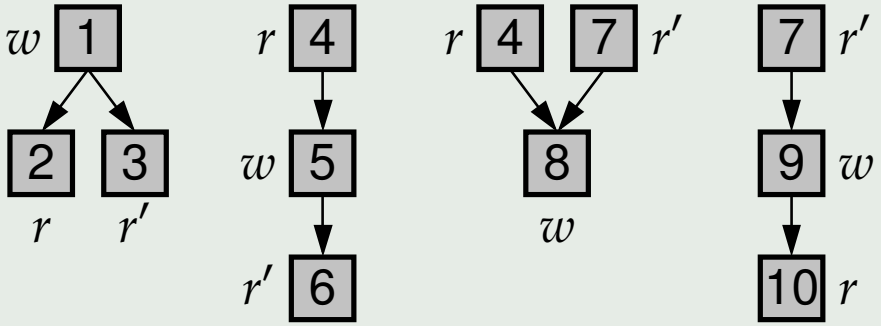
- $M := \langle T, \ldots \rangle$ be a general model of computation
- $\diamond \subseteq T \times T$ a valid unconditional independence relation

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

$T = \{w, r, r'\}$

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

Partially-ordered runs:



Theorem (Soundness & Uniqueness)

- *The unfolding $\mathcal{U}_{M,\diamond}$ exists and is unique.*
- *Every sequential execution is represented by exactly 1 configuration.*

Unfolding over User-Defined Independence Relation

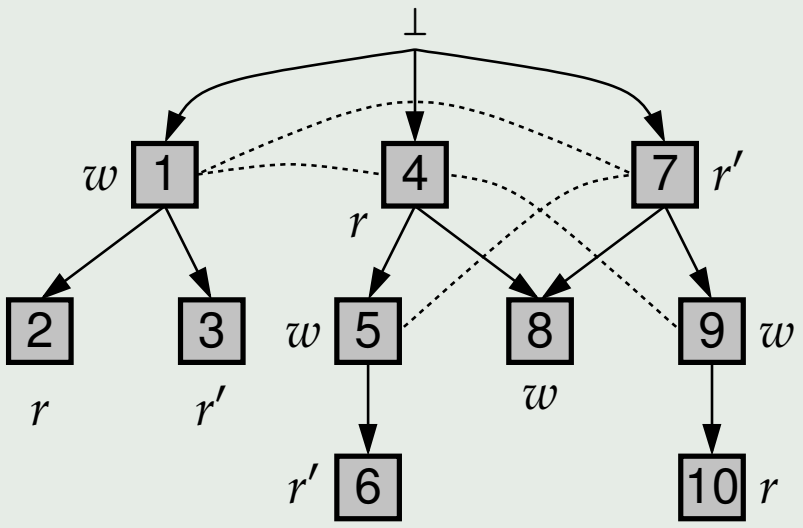
Let

- $M := \langle T, \ldots \rangle$ be a general model of computation
- $\diamond \subseteq T \times T$ a valid unconditional independence relation

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

$T = \{w, r, r'\}$

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



Theorem (Soundness & Uniqueness)

- *The unfolding $\mathcal{U}_{M,\diamond}$ exists and is unique.*
- *Every sequential execution is represented by exactly 1 configuration.*

Motivating Questions

Do both approaches exploit the same source of reduction?

- Yes, independence-based PORs and $\mathcal{U}_{M,\diamond}$ solely exploit $\diamond$
- $\mathcal{R}_{M,\diamond}$ is polynomial-time convertible into $\mathcal{U}_{M,\diamond}$

Are they just different exploration algorithms over the same underlying reduced space?

- Necessarily yes!
- In the sequel: 6 algorithmic differences

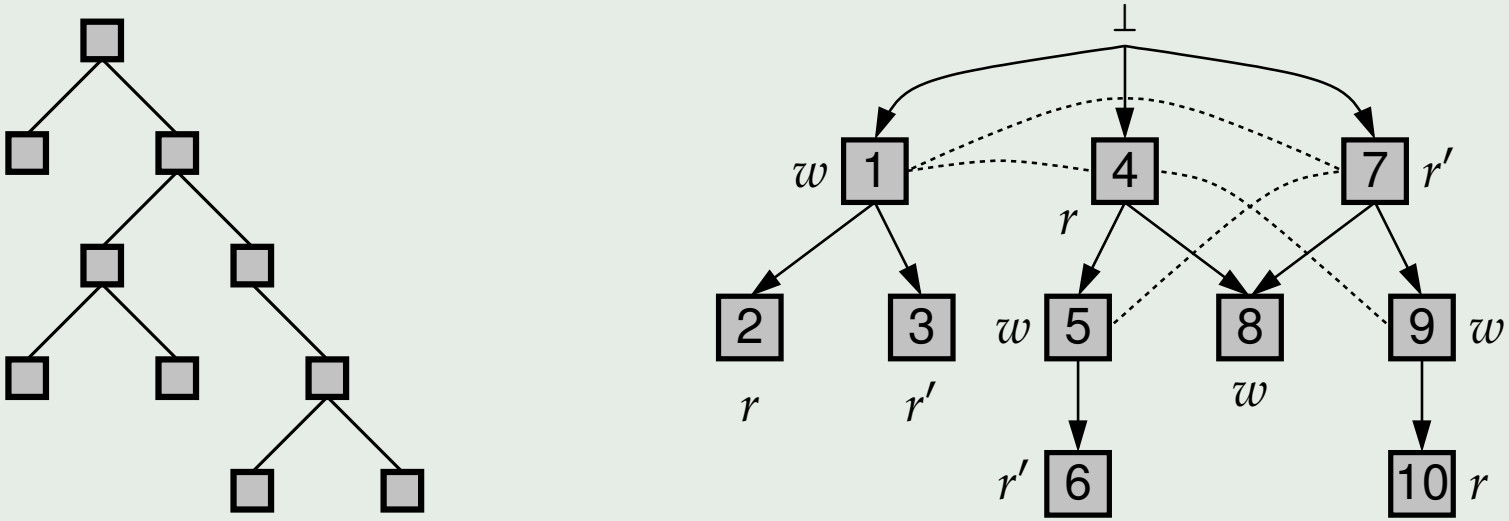
Main contribution

A novel stateless POR exploration of unfolding semantics

- Retains advantages of both approaches
- (Super-)Optimal: can explore fewer executions than Mazurkiewicz traces

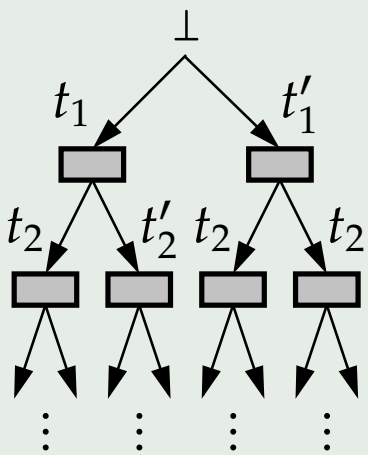
POR vs Unfoldings: 6 Algorithmic Differences

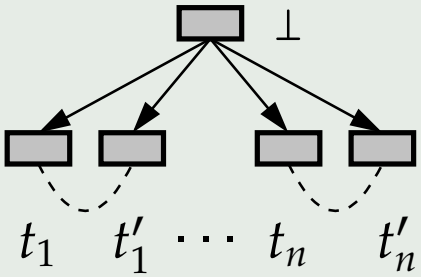
1. Unfolding extension is NP-complete; POR extension is constant-time



POR vs Unfoldings: 6 Algorithmic Differences

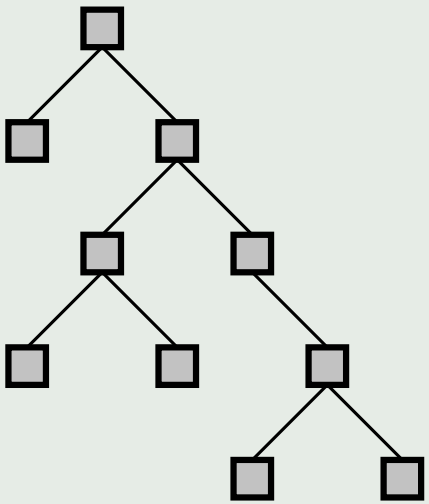
1. Unfolding extension is NP-complete; POR extension is constant-time
2. $\mathcal{R}_{M,\diamond}$ can be exponentially larger than $\mathcal{U}_{M,\diamond}$

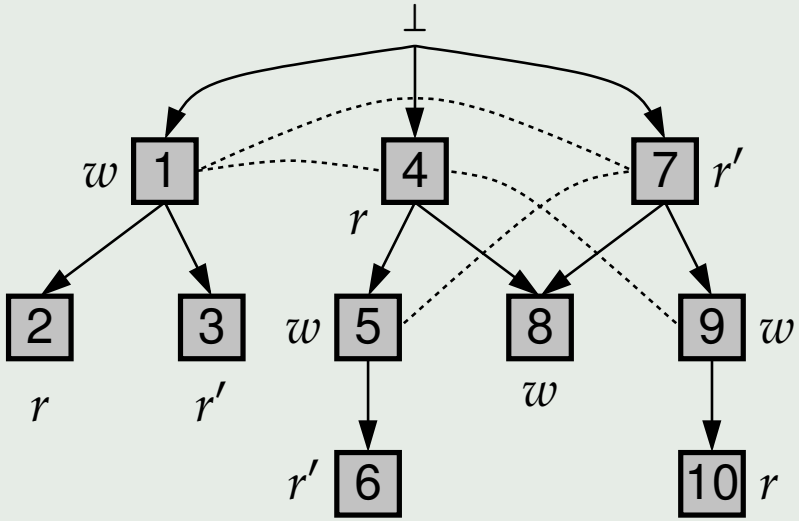




POR vs Unfoldings: 6 Algorithmic Differences

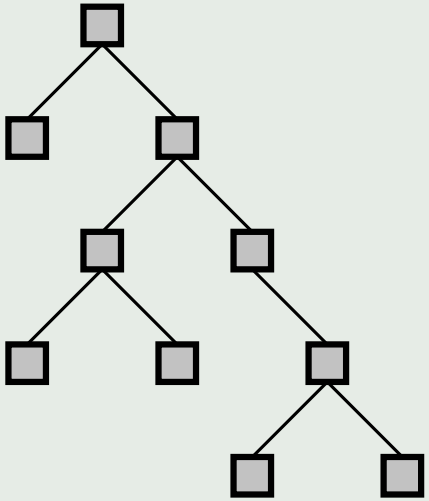
1. Unfolding extension is NP-complete; POR extension is constant-time
2. $\mathcal{R}_{M,\diamond}$ can be exponentially larger than $\mathcal{U}_{M,\diamond}$
3. Unfolding algorithms are inherently stateful; state-of-the-art DPORs are stateless
   - [Flanagan, Godefroid, POPL'05], [Abdulla et al., POPL'14]

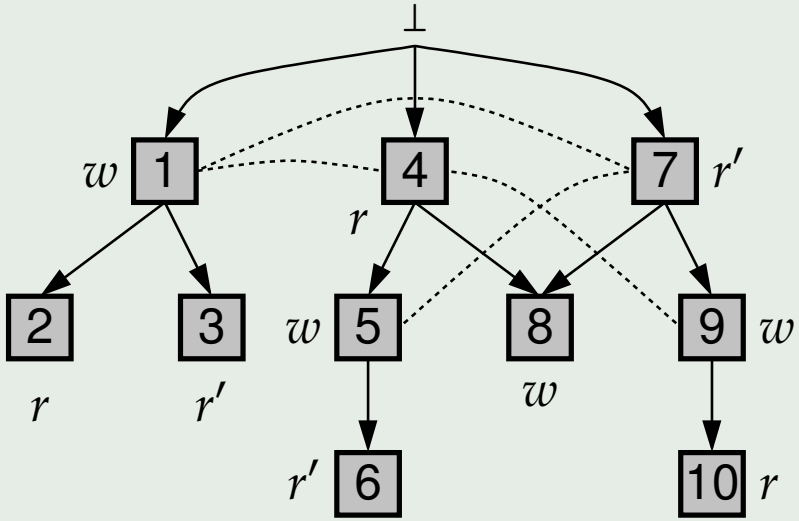




POR vs Unfoldings: 6 Algorithmic Differences

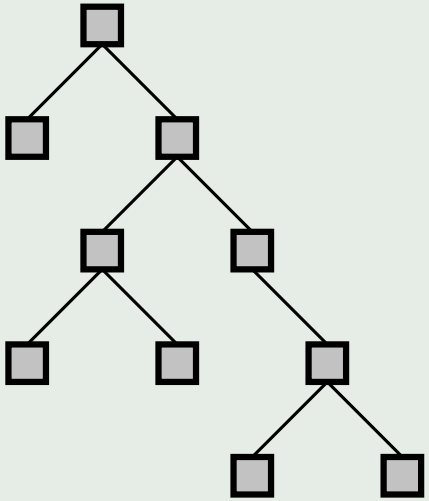
1. Unfolding extension is NP-complete; POR extension is constant-time
2. $\mathcal{R}_{M,\diamond}$ can be exponentially larger than $\mathcal{U}_{M,\diamond}$
3. Unfolding algorithms are inherently stateful; state-of-the-art DPORs are stateless
4. Dynamic POR: difficult to avoid repeated exploration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

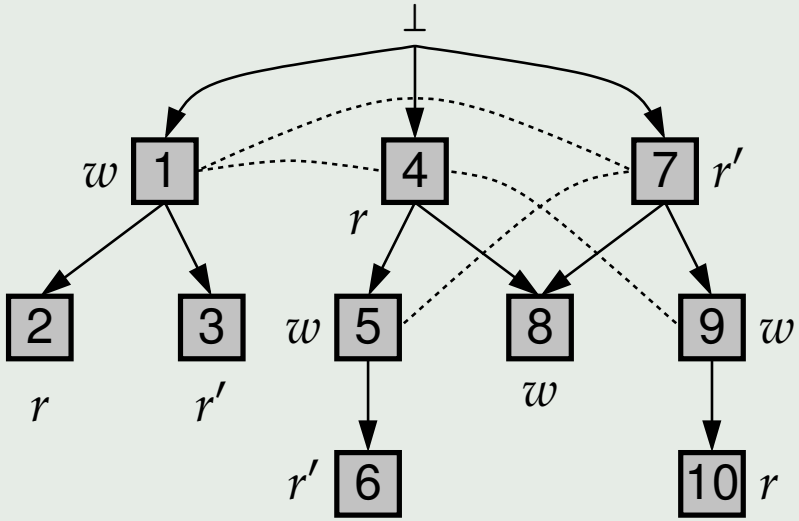




POR vs Unfoldings: 6 Algorithmic Differences

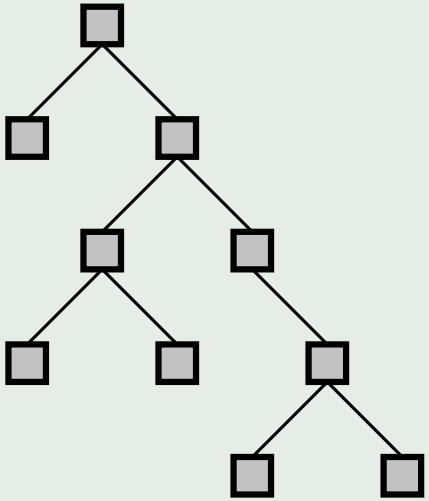
1. Unfolding extension is NP-complete; POR extension is constant-time
2. $\mathcal{R}_{M,\diamond}$ can be exponentially larger than $\mathcal{U}_{M,\diamond}$
3. Unfolding algorithms are inherently stateful; state-of-the-art DPORs are stateless
4. Dynamic POR: difficult to avoid repeated exploration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

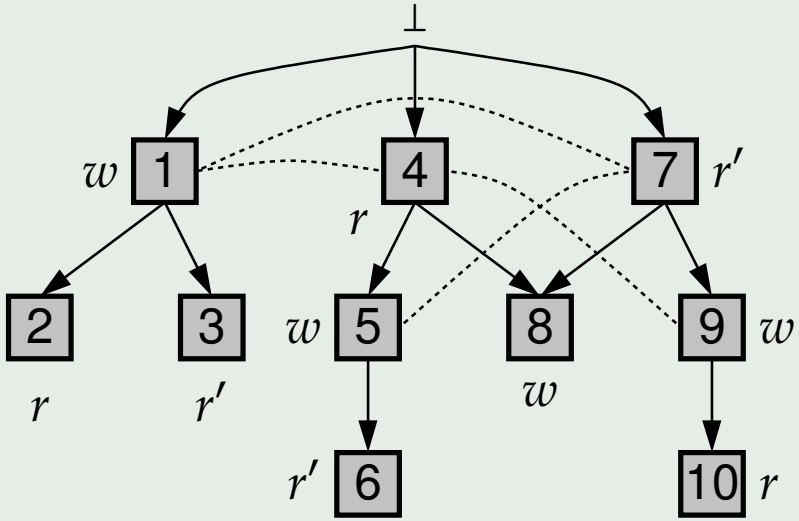




POR vs Unfoldings: 6 Algorithmic Differences

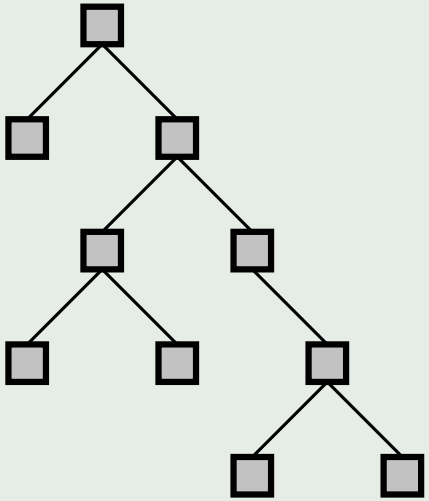
1. Unfolding extension is NP-complete; POR extension is constant-time
2. $\mathcal{R}_{M,\diamond}$ can be exponentially larger than $\mathcal{U}_{M,\diamond}$
3. Unfolding algorithms are inherently stateful; state-of-the-art DPORs are stateless
4. Dynamic POR: difficult to avoid repeated exploration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

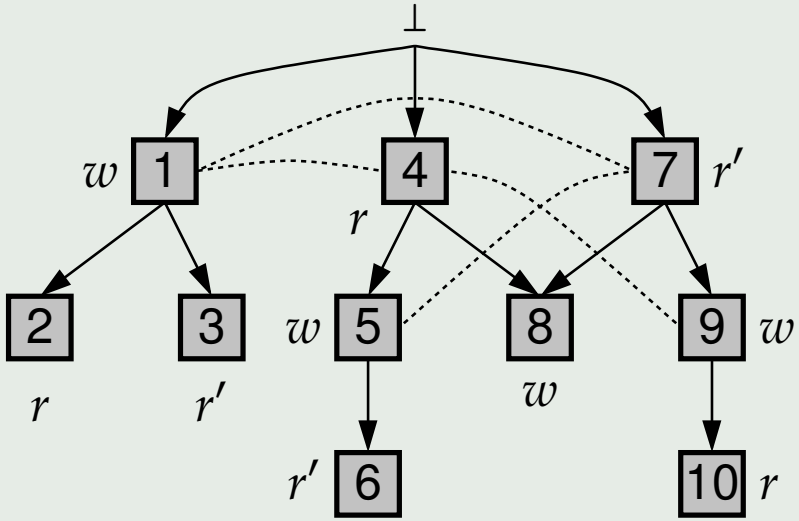




POR vs Unfoldings: 6 Algorithmic Differences

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

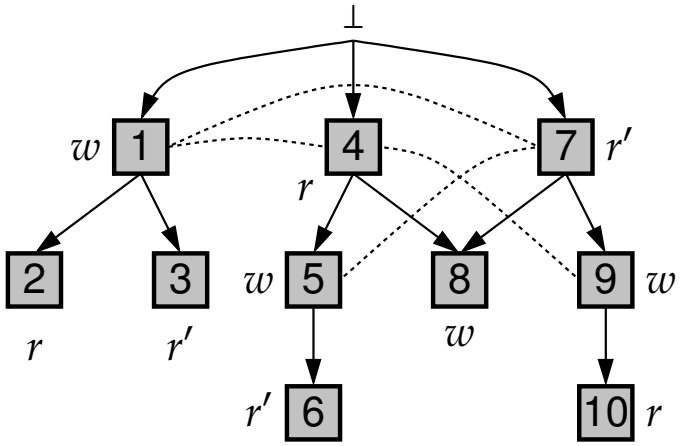
Our algorithm (main contribution, next slide)

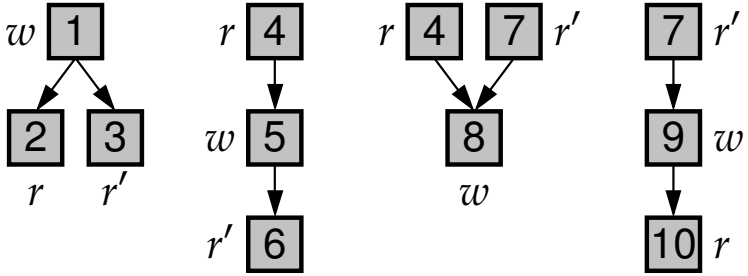
A novel stateless POR exploration of unfolding semantics

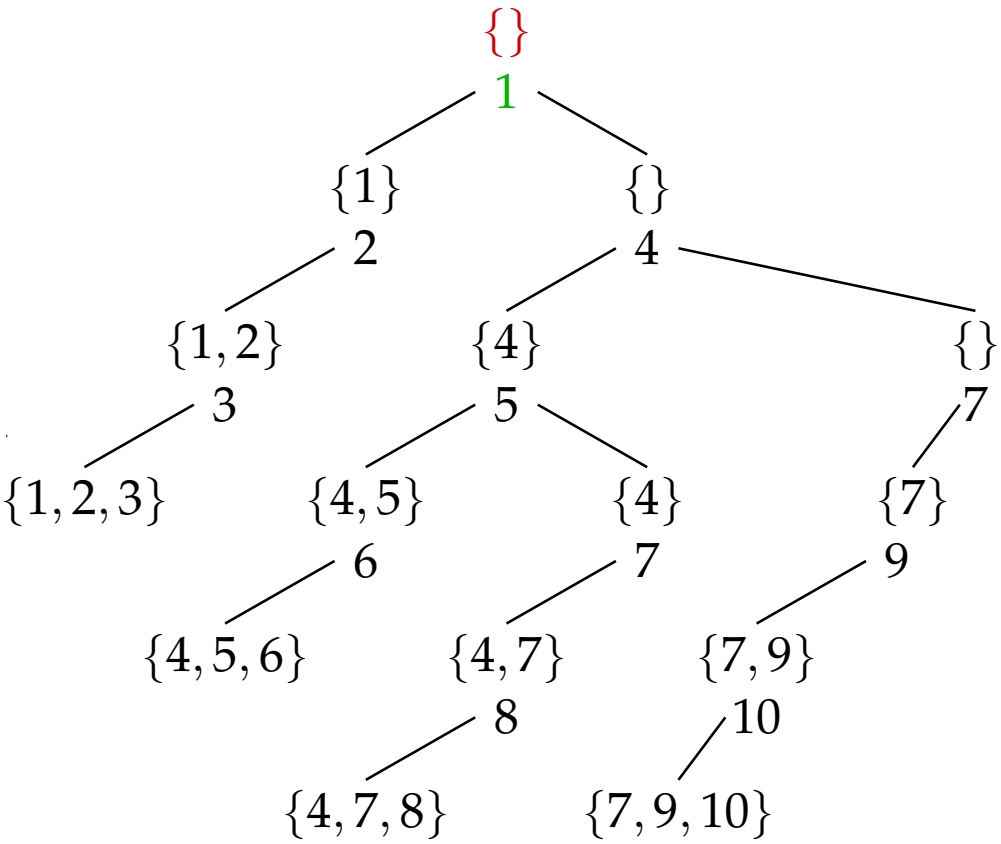
- Retains advantages of both approaches
- (Super-)Optimal: can explore fewer executions than Mazurkiewicz traces
- Addresses all above points except (2)

Stateless Unfolding-based Partial-Order Reduction

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





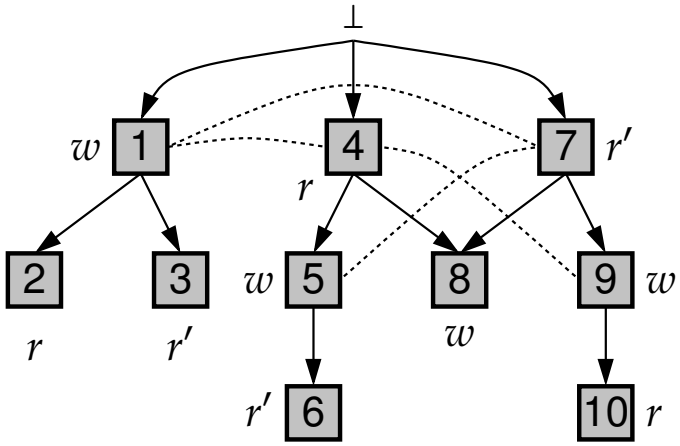


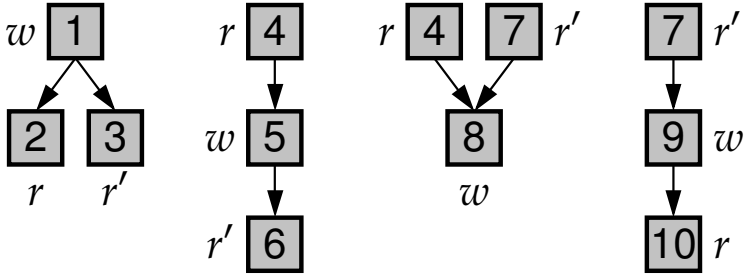
$C = \{\}$ $\quad e = 1$ $\quad A = \{\}$

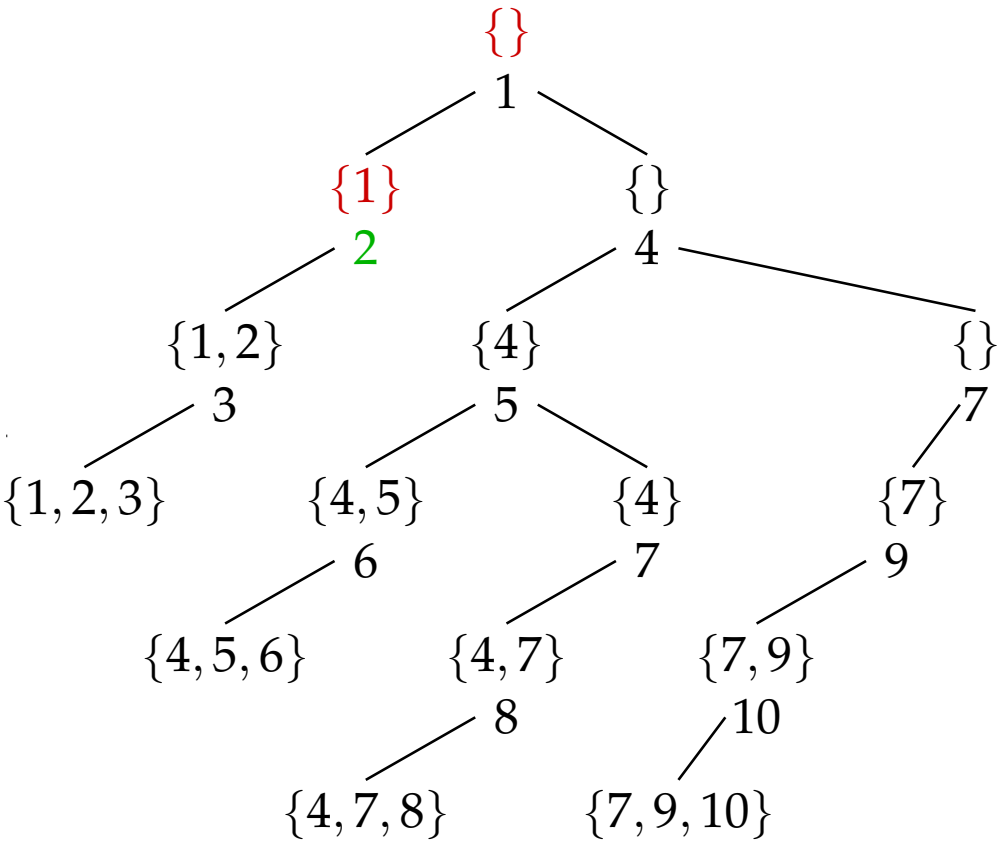
$U = \{\}$ $\quad G = \{\}$

Stateless Unfolding-based Partial-Order Reduction

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





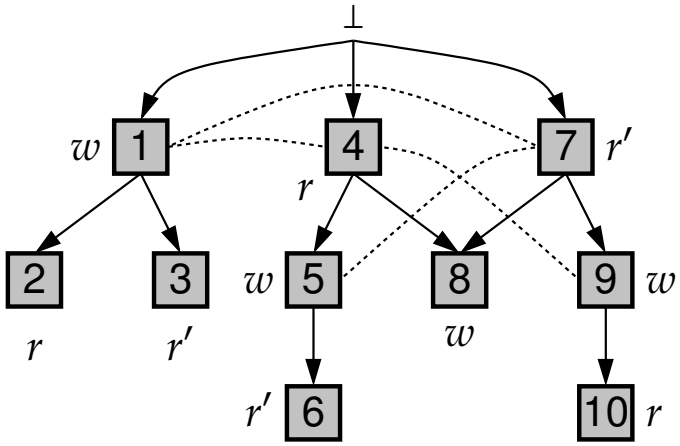


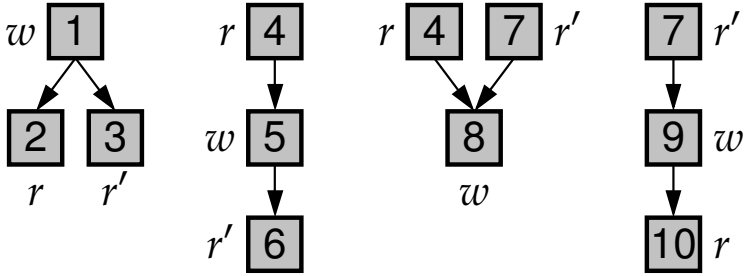
$C = \{1\}$ $e = 2$ $A = \{\}$

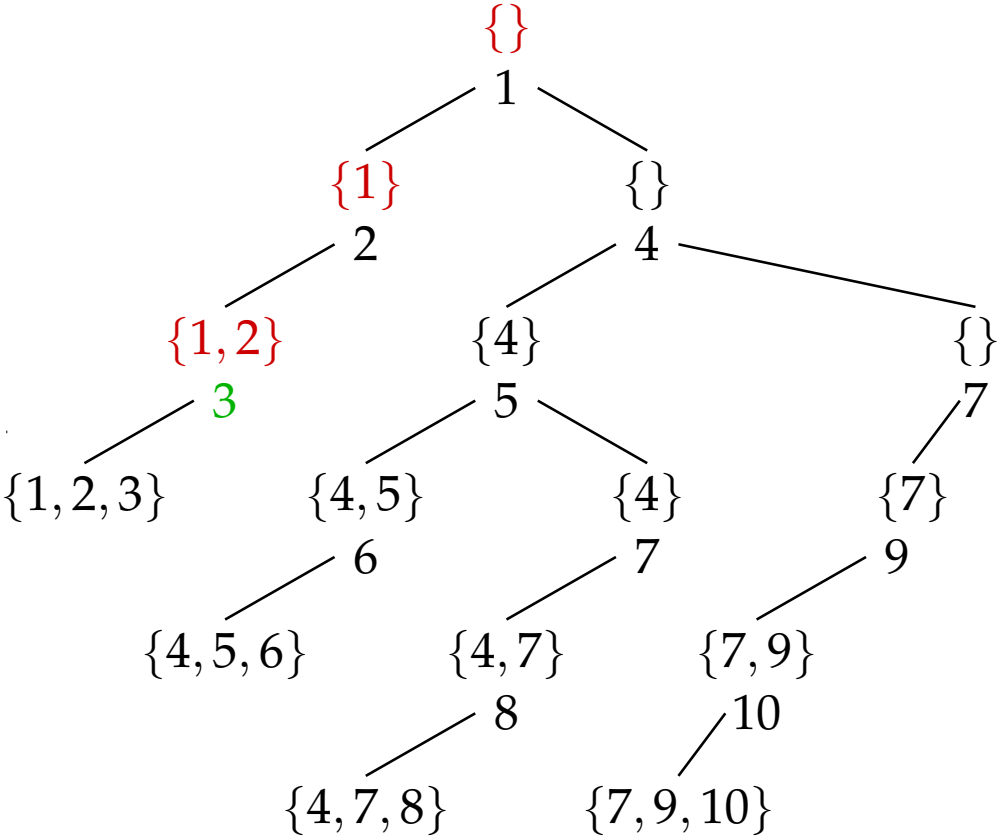
$U = \{1\}$ $G = \{\}$

Stateless Unfolding-based Partial-Order Reduction

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





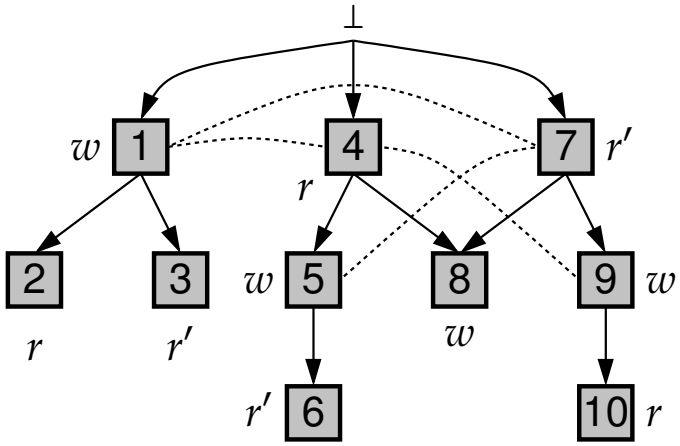


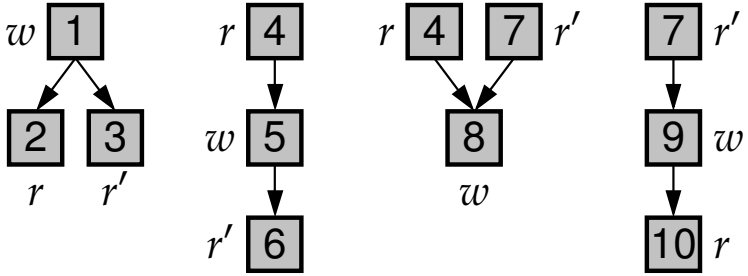
$C = \{1,2\}$ $e = 3$ $A = \{\}$

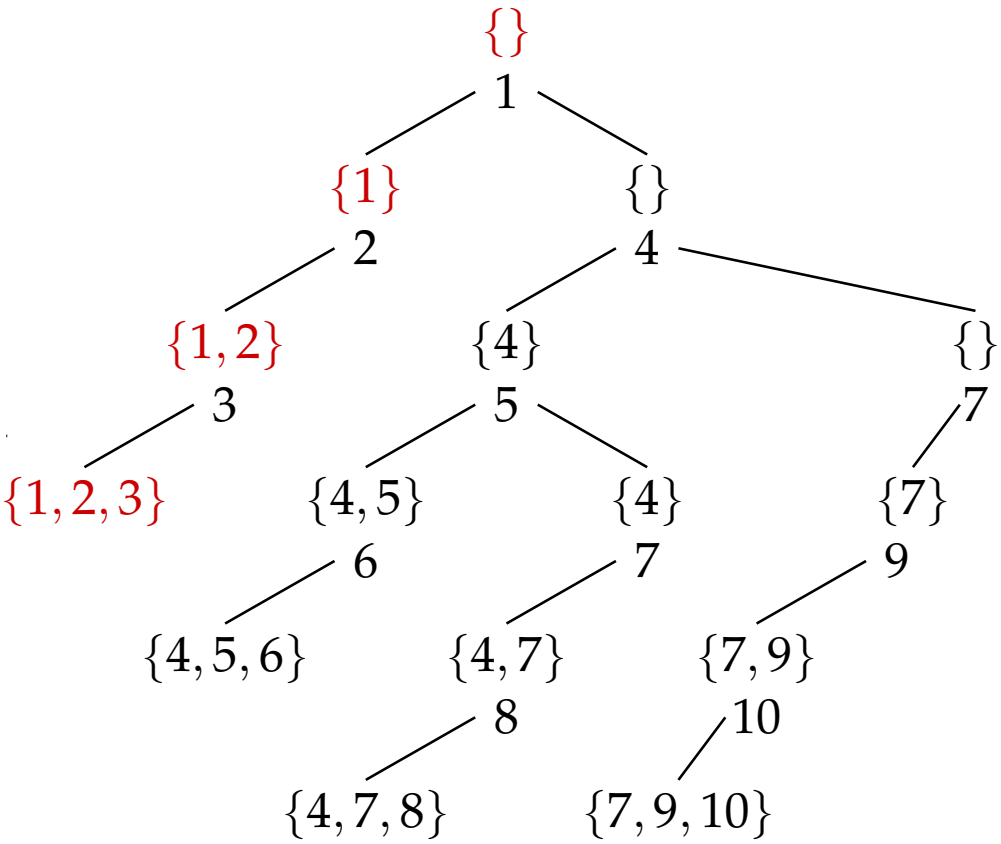
$U = \{1,2\}$ $G = \{\}$

Stateless Unfolding-based Partial-Order Reduction

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





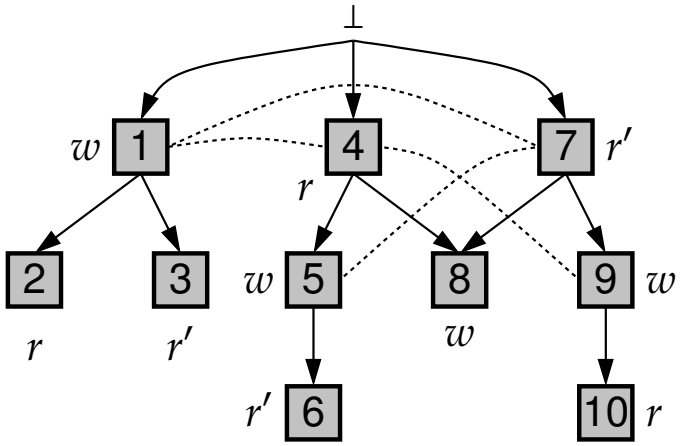


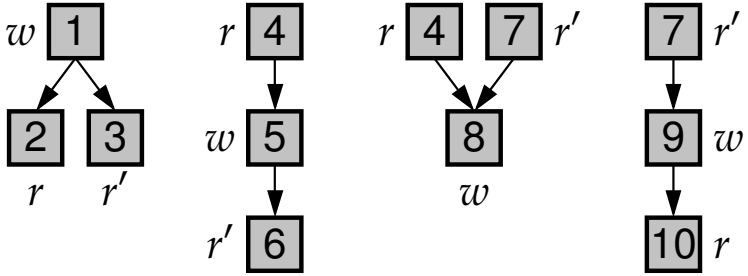
$C = \{1,2,3\}$ $\quad e = n/a \quad$ $A = \{\}$

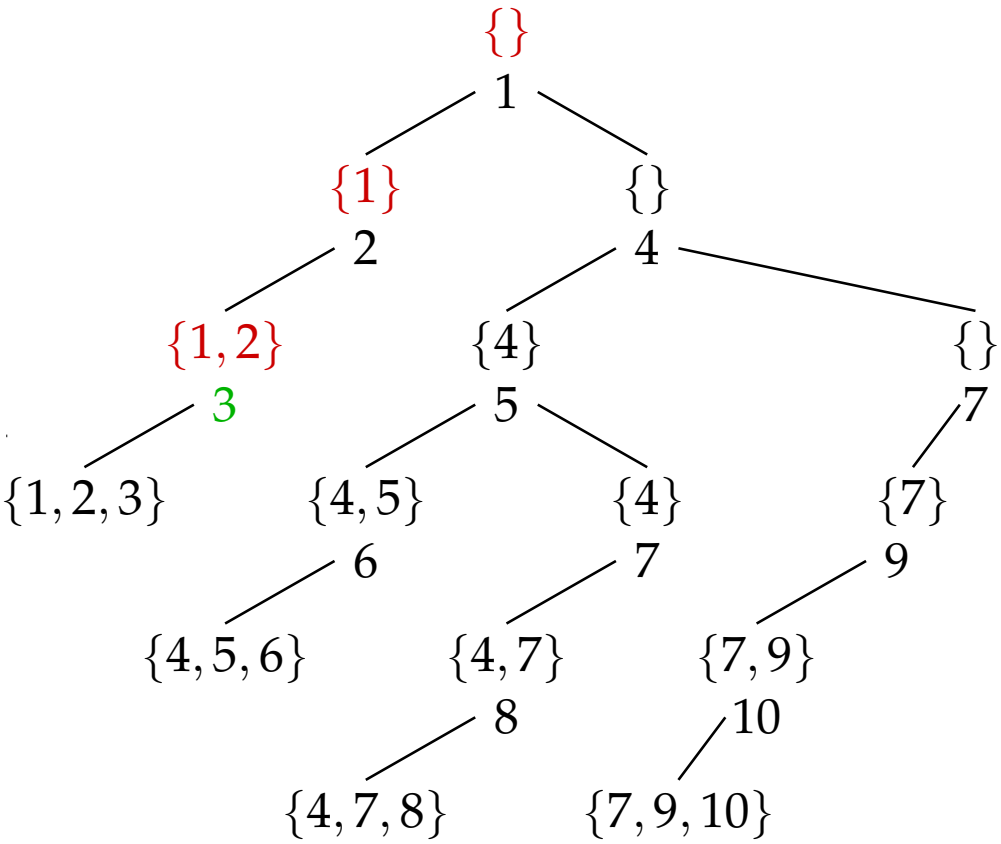
$U = \{1,2,3,4,7\}$ $\quad G = \{\}$

Stateless Unfolding-based Partial-Order Reduction

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





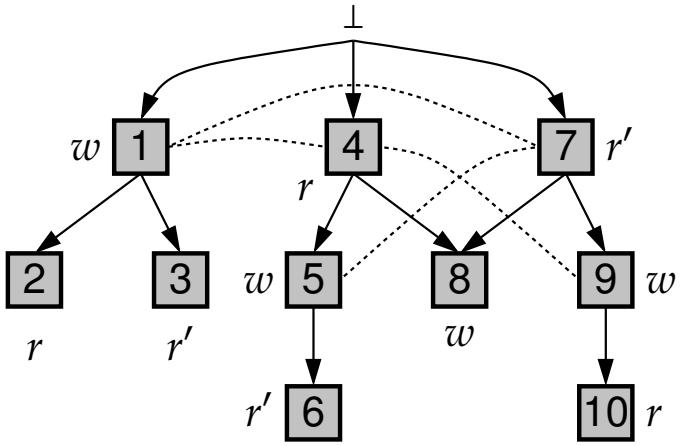


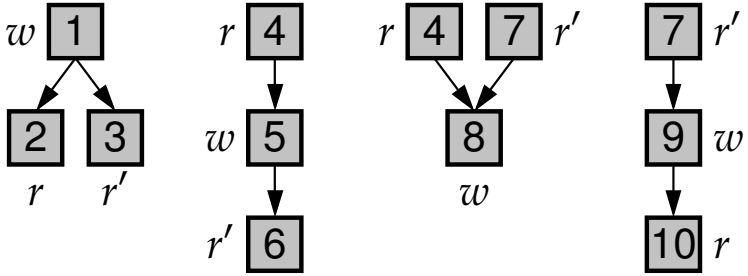
$C = \{1,2\}$ $\quad e = 3$ $\quad A = \{\}$

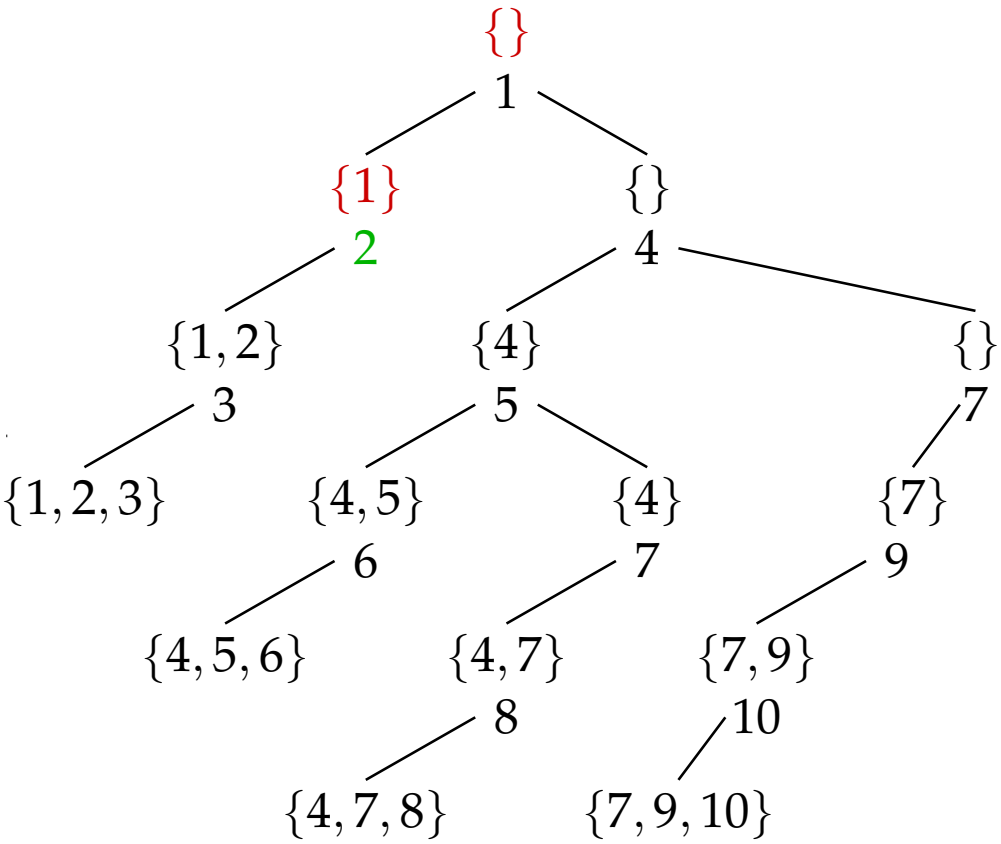
$U = \{1,2,4,7\}$ $\quad G = \{3\}$

Stateless Unfolding-based Partial-Order Reduction

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





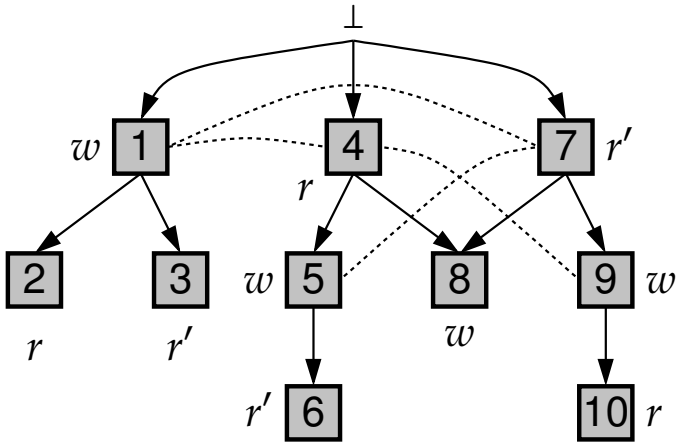


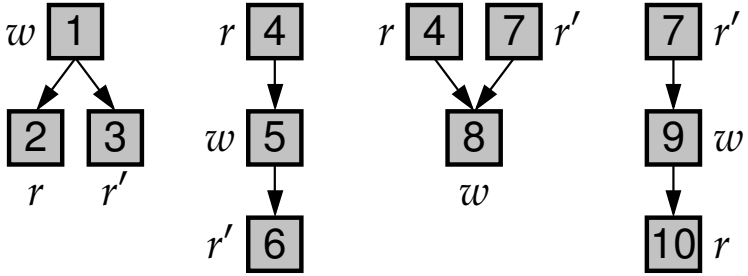
$C = \{1\}$ $\quad e = 2$ $\quad A = \{\}$

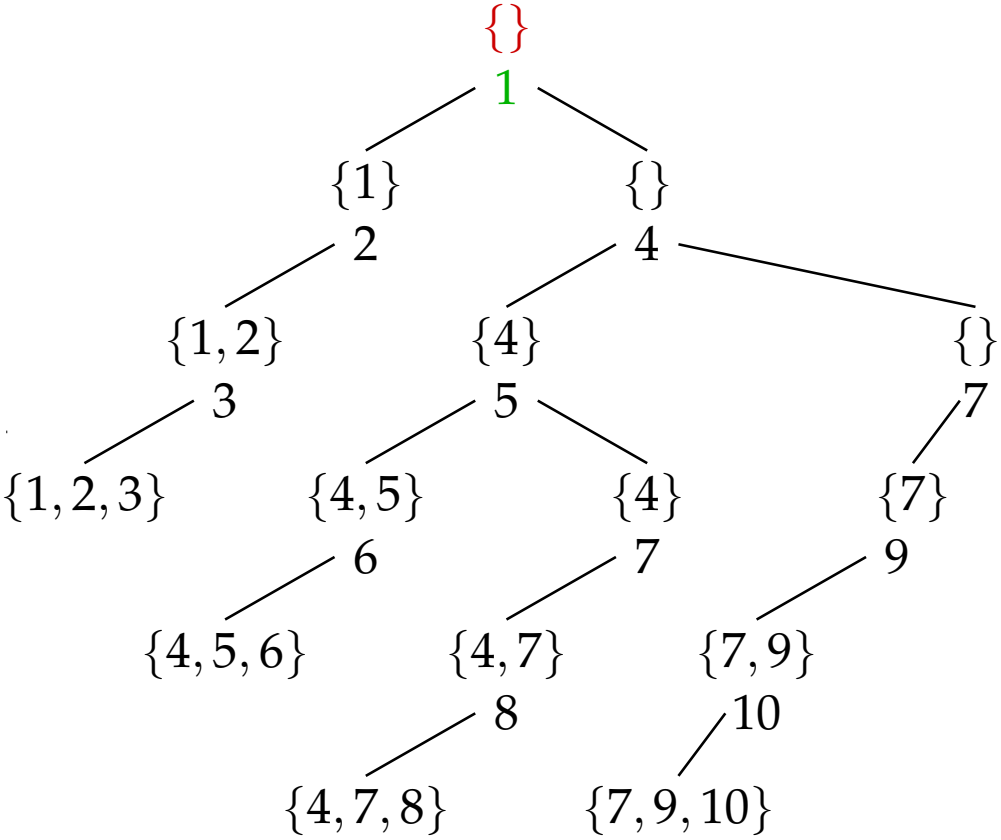
$U = \{1, 4, 7\}$ $\quad G = \{3, 2\}$

Stateless Unfolding-based Partial-Order Reduction

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





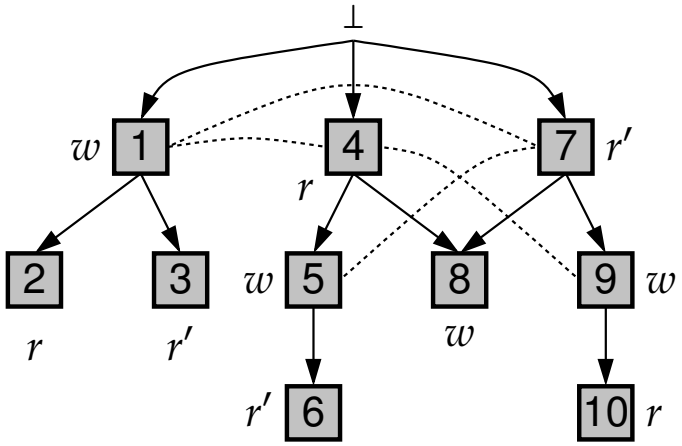


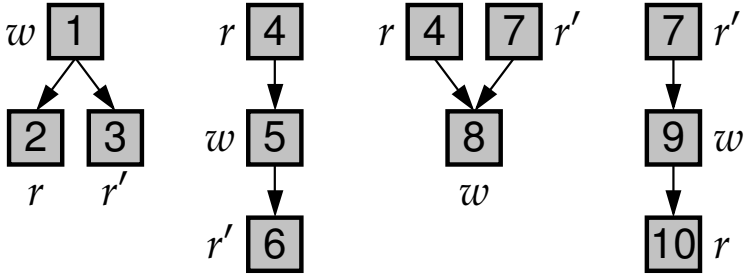
$C = \{\}$ $\quad e = 1$ $\quad A = \{\}$

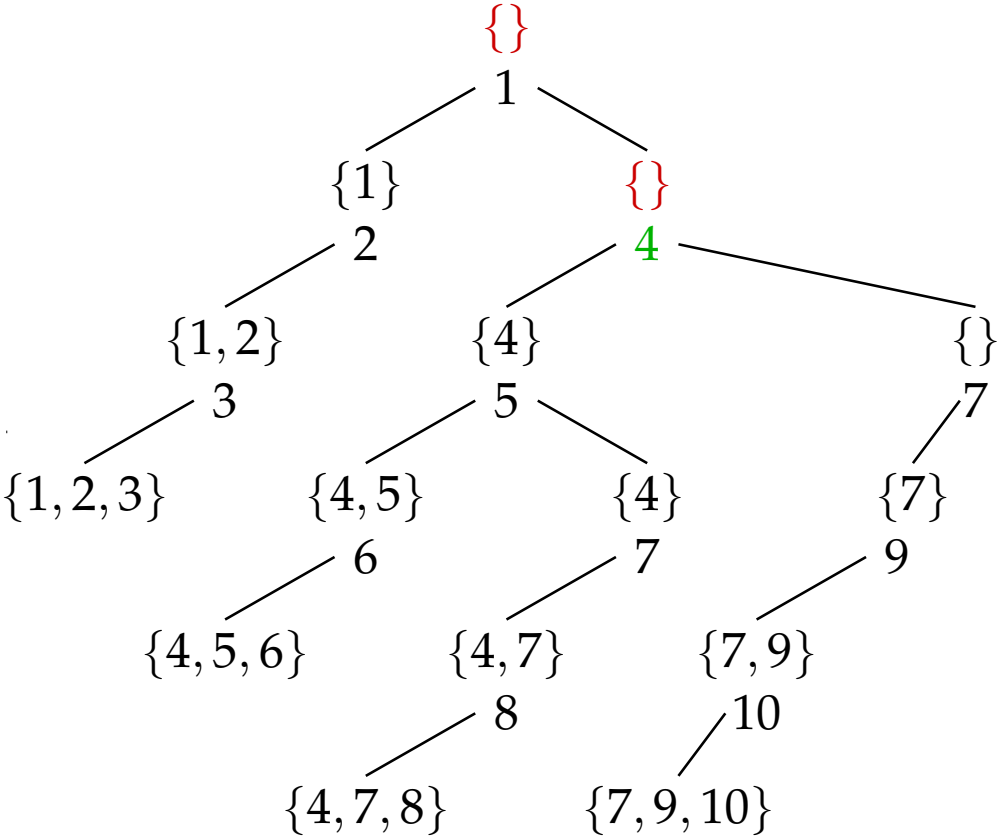
$U = \{4, 7\}$ $\quad G = \{3, 2\}$

Stateless Unfolding-based Partial-Order Reduction

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





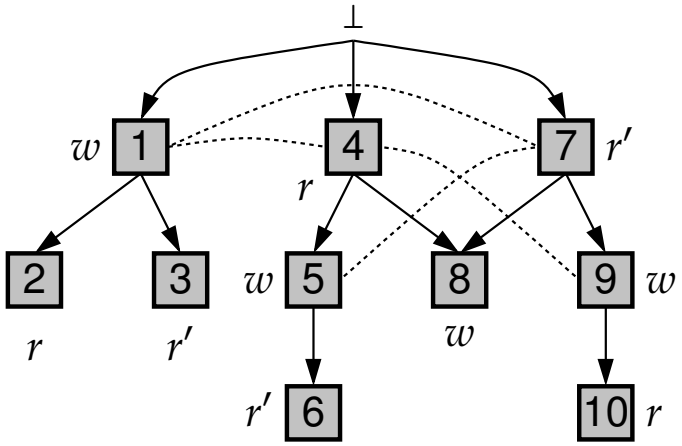


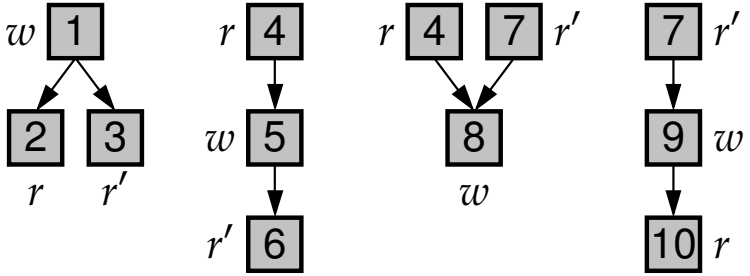
$C = \{\}$ $\quad e = 4$ $\quad A = \{4\}$

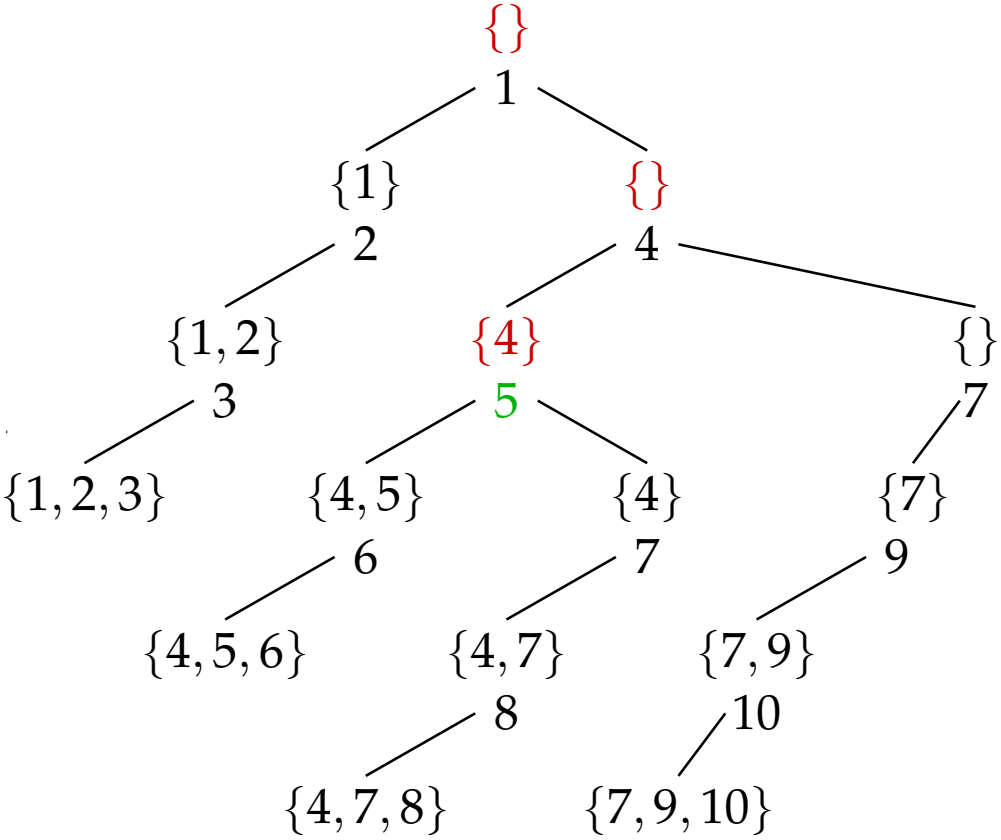
$U = \{4, 7, 1\}$ $\quad G = \{3, 2\}$

Stateless Unfolding-based Partial-Order Reduction

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





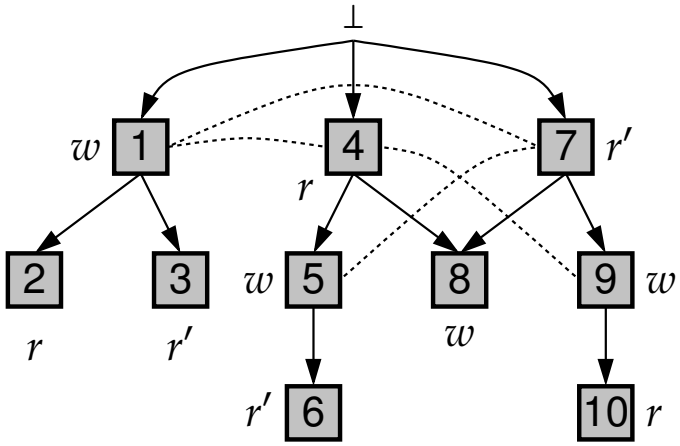


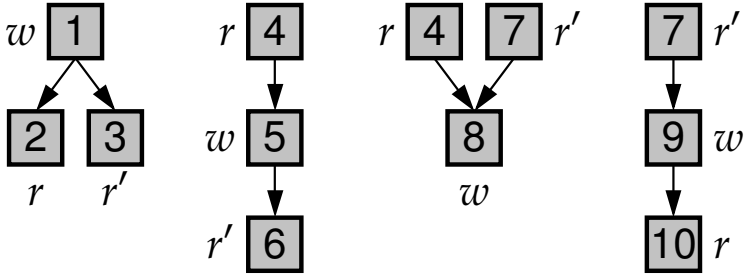
$C = \{4\}$ $\quad e = 5$ $\quad A = \{\}$

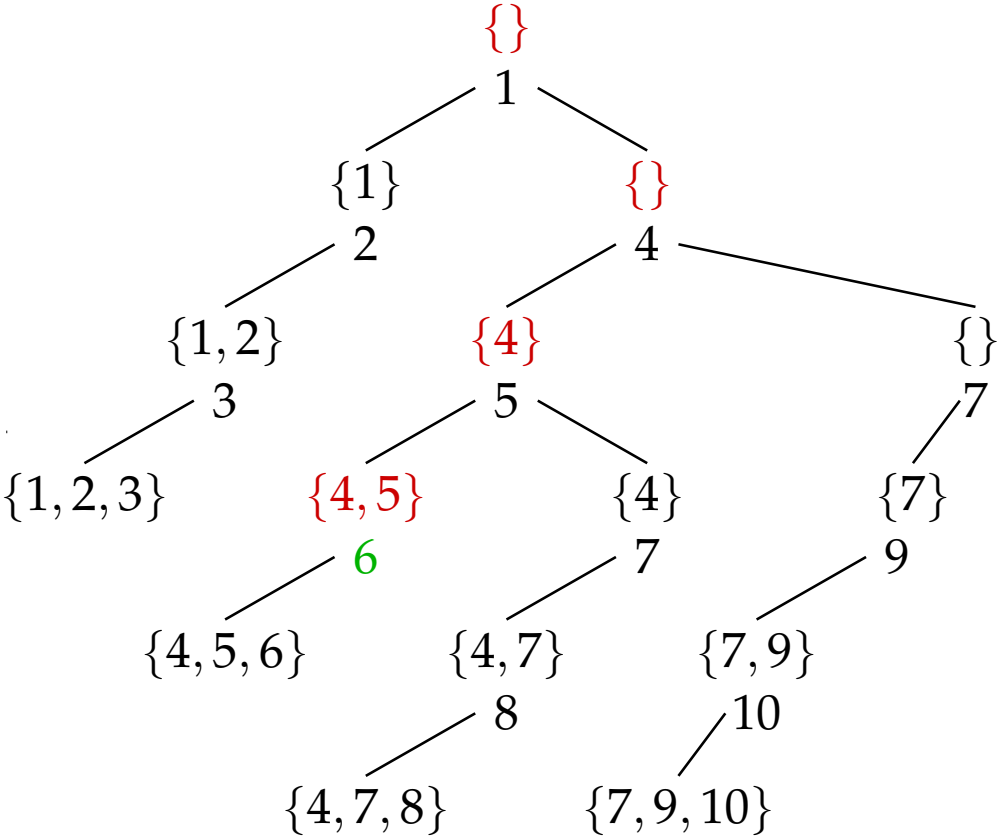
$U = \{4, 7, 1\}$ $\quad G = \{3, 2\}$

Stateless Unfolding-based Partial-Order Reduction

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





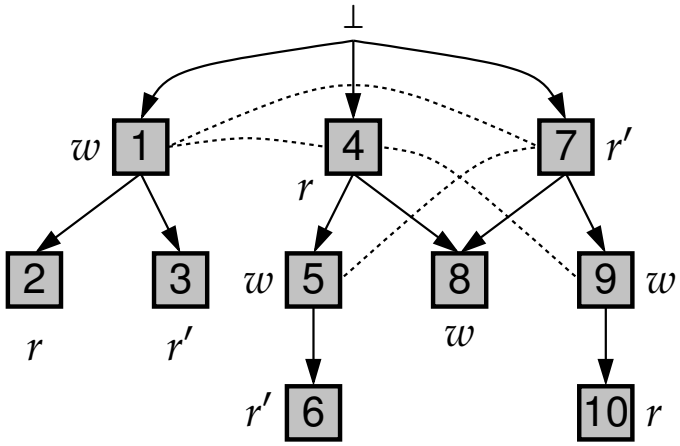


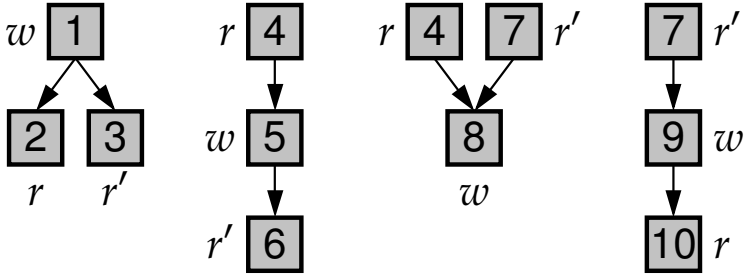
$C = \{4, 5\}$ $\quad e = 6$ $\quad A = \{\}$

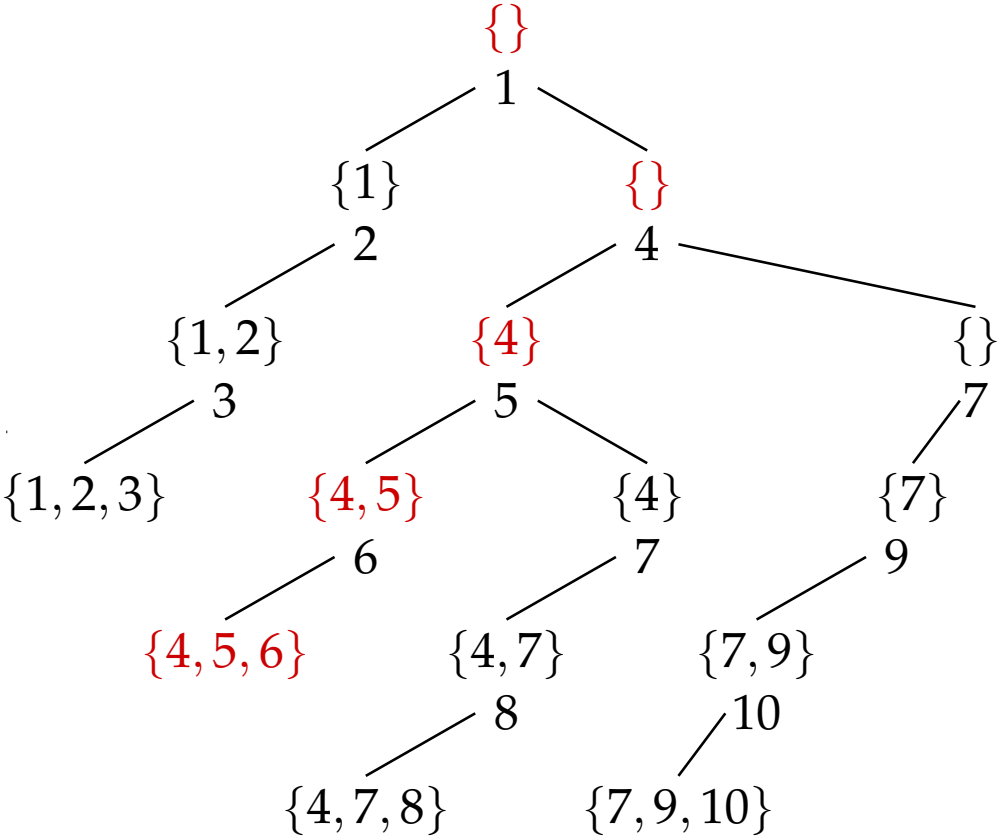
$U = \{4, 5, 7, 1\}$ $\quad G = \{3, 2\}$

Stateless Unfolding-based Partial-Order Reduction

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





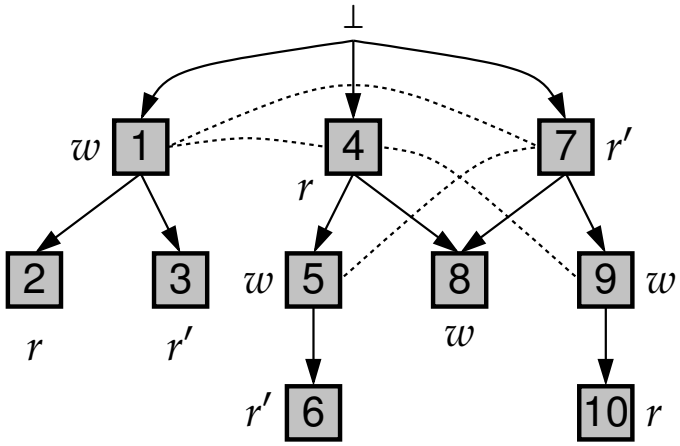


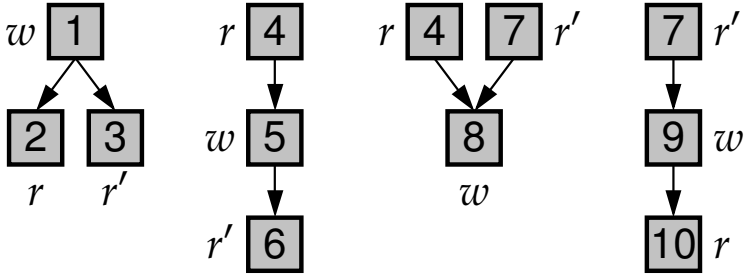
$C = \{4, 5, 6\}$ $\quad e = n/a \quad$ $A = \{\}$

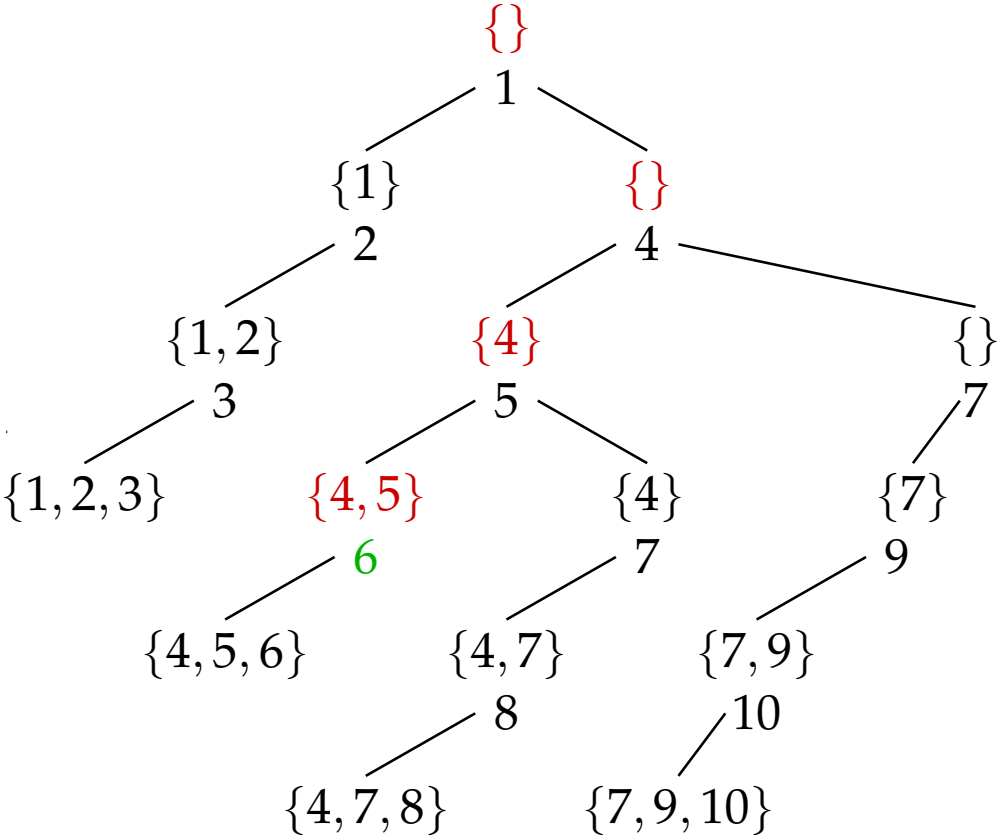
$U = \{4, 5, 6, 7, 1\}$ $\quad G = \{3, 2\}$

Stateless Unfolding-based Partial-Order Reduction

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





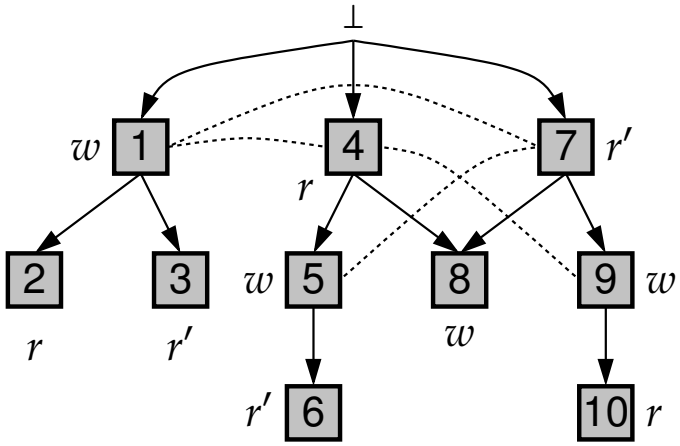


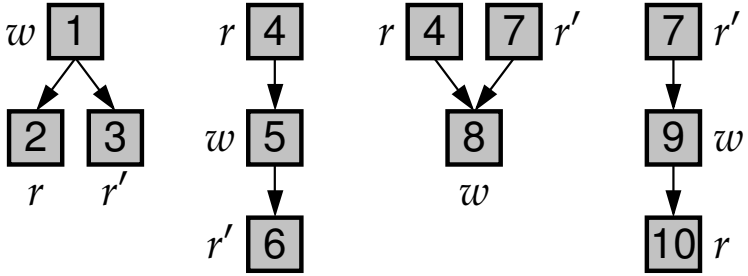
$C = \{4, 5\}$ $\quad e = 6$ $\quad A = \{\}$

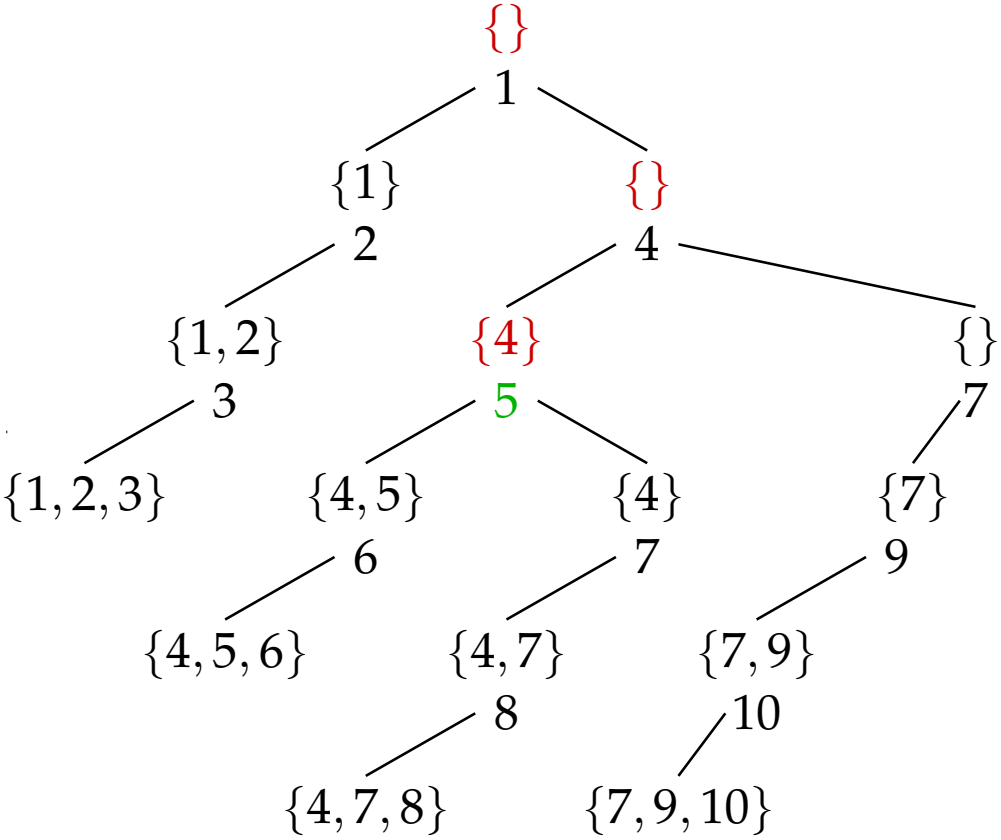
$U = \{4, 5, 7, 1\}$ $\quad G = \{3, 2, 6\}$

Stateless Unfolding-based Partial-Order Reduction

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





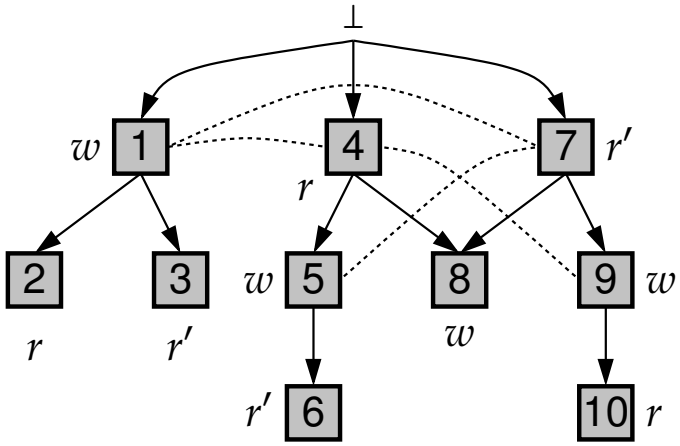


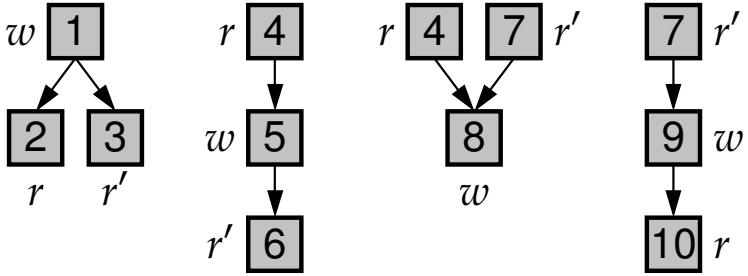
$C = \{4\}$ $\quad e = 5$ $\quad A = \{\}$

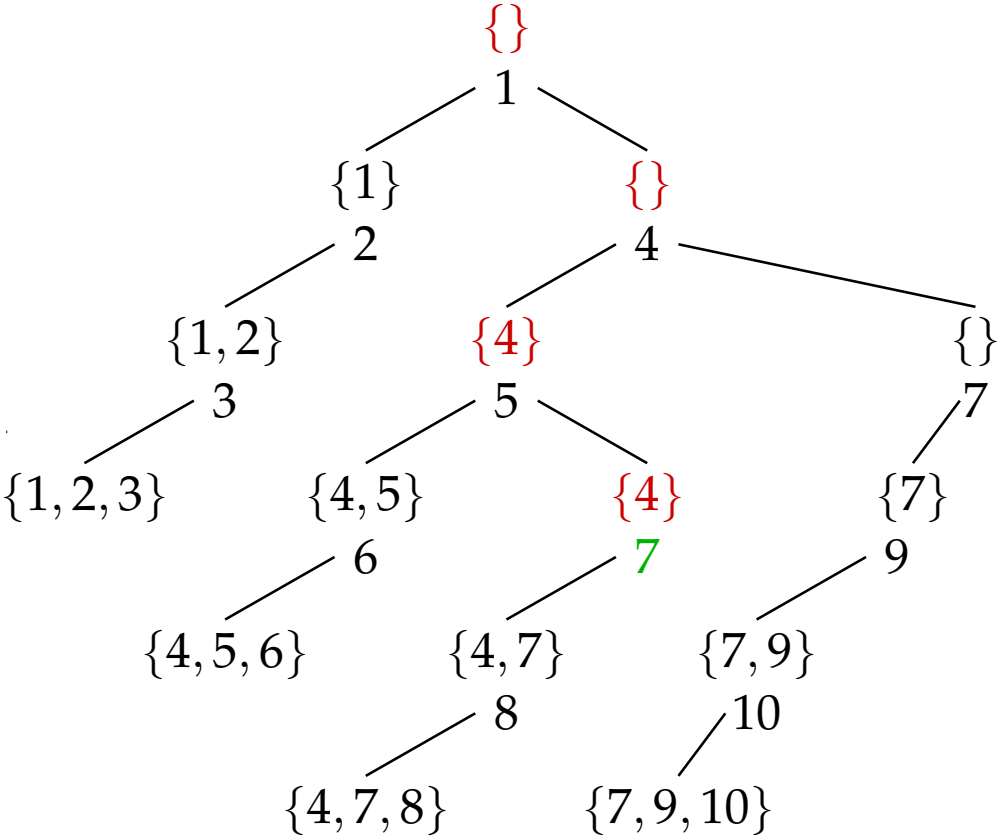
$U = \{4, 7, 1\}$ $\quad G = \{3, 2, 6\}$

Stateless Unfolding-based Partial-Order Reduction

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





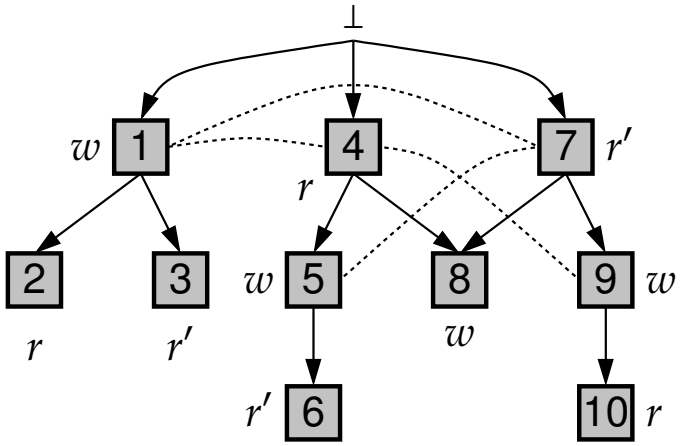


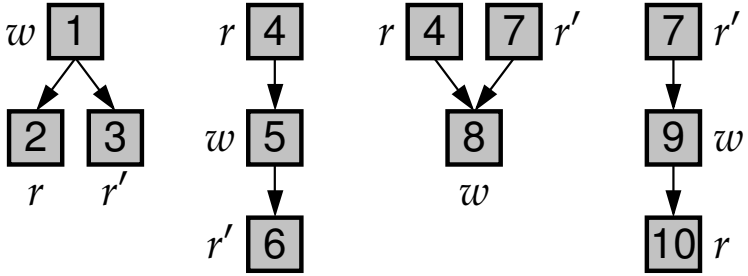
$C = \{4\}$ $\quad e = 7$ $\quad A = \{7\}$

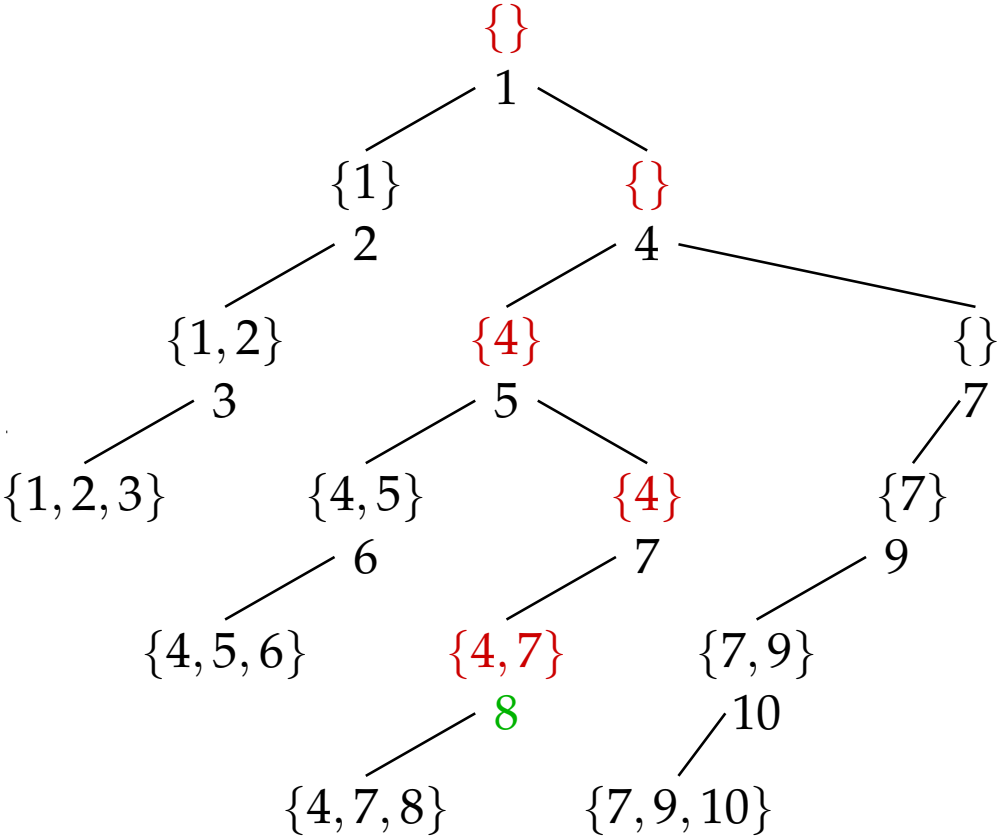
$U = \{4, 7, 1, 5\}$ $\quad G = \{3, 2, 6\}$

Stateless Unfolding-based Partial-Order Reduction

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





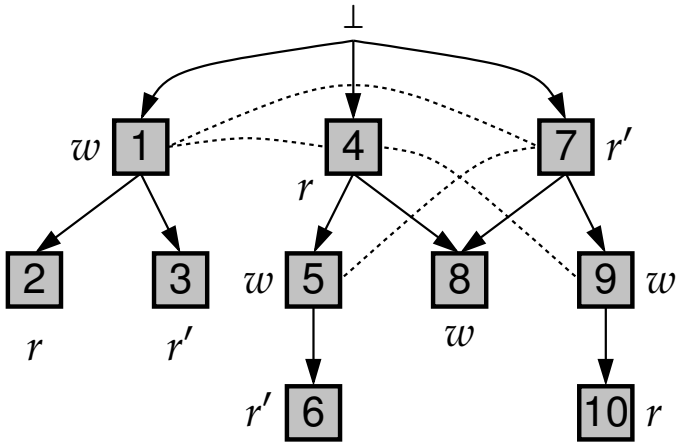


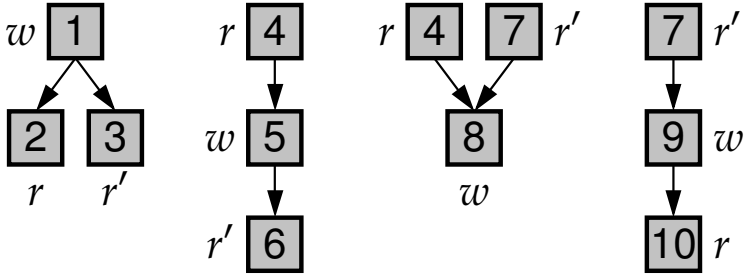
$C = \{4, 7\}$ $\quad e = 8$ $\quad A = \{\}$

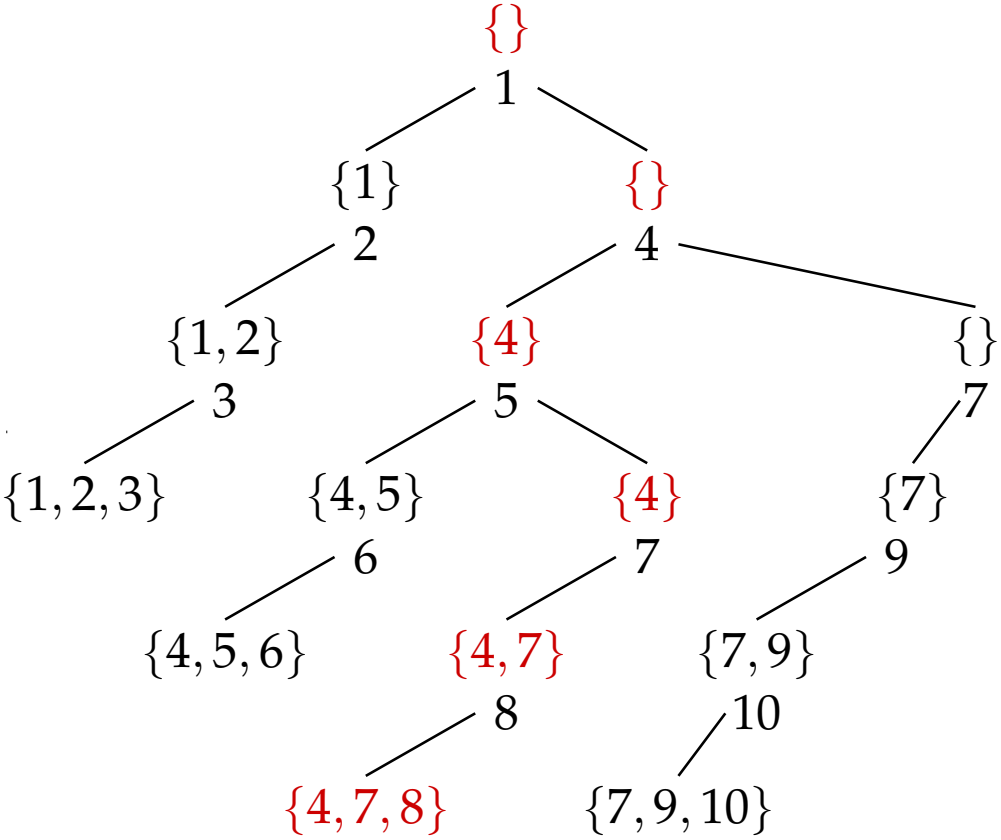
$U = \{4, 7, 1, 5\}$ $\quad G = \{3, 2, 6\}$

Stateless Unfolding-based Partial-Order Reduction

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





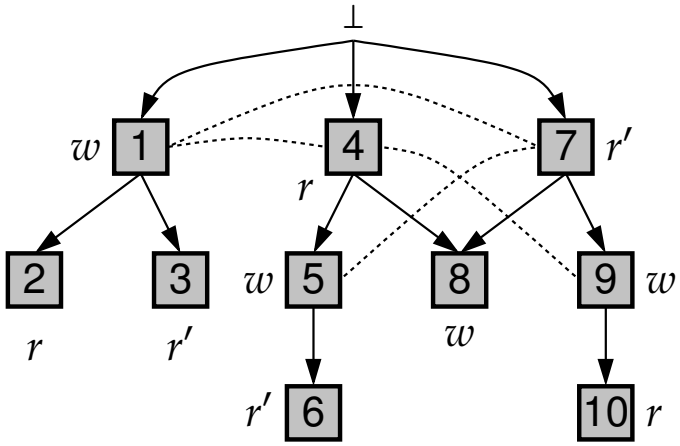


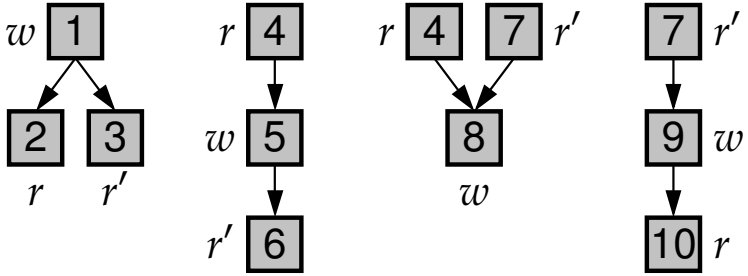
$C = \{4,7,8\}$ $\quad e = n/a \quad A = \{\}$

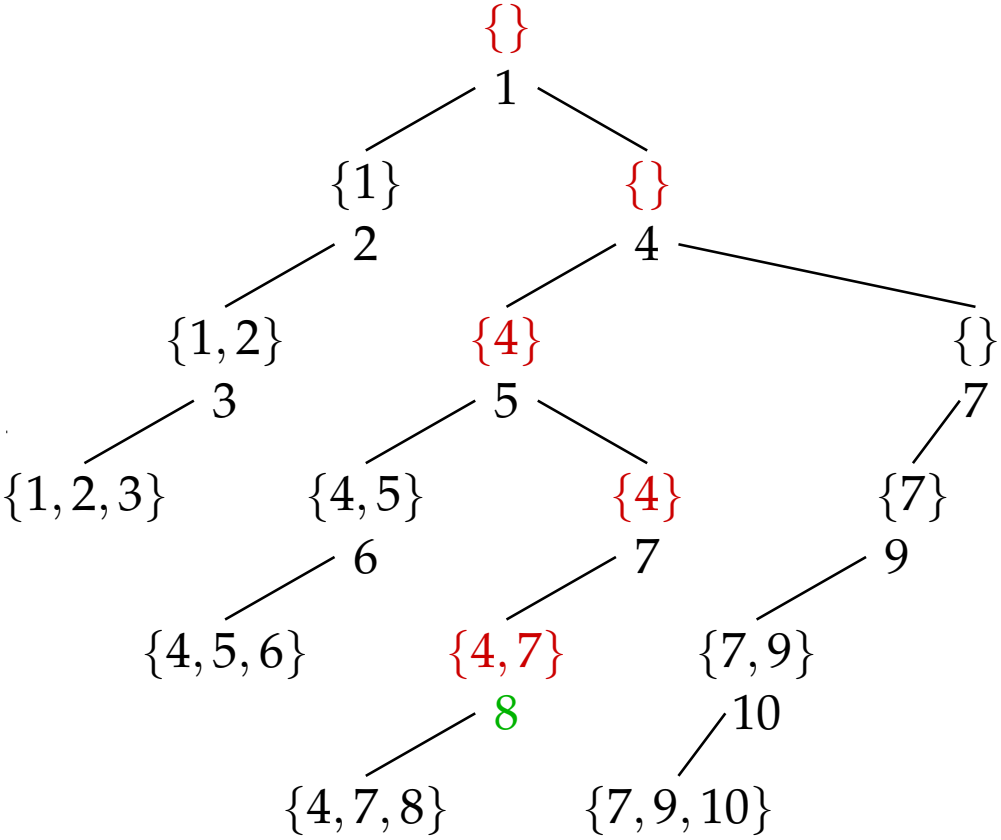
$U = \{4,7,8,1,5\}$ $\quad G = \{3,2,6\}$

Stateless Unfolding-based Partial-Order Reduction

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





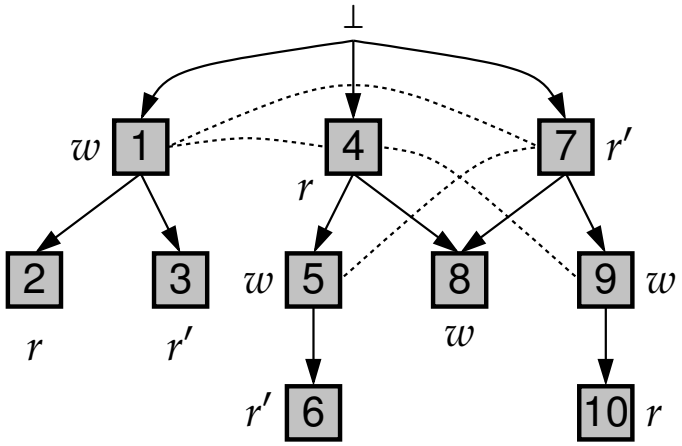


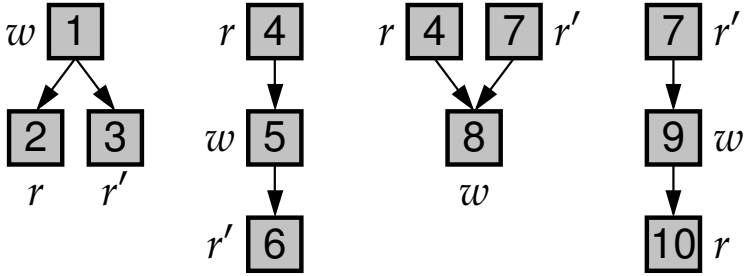
$C = \{4, 7\}$ $\quad e = 8$ $\quad A = \{\}$

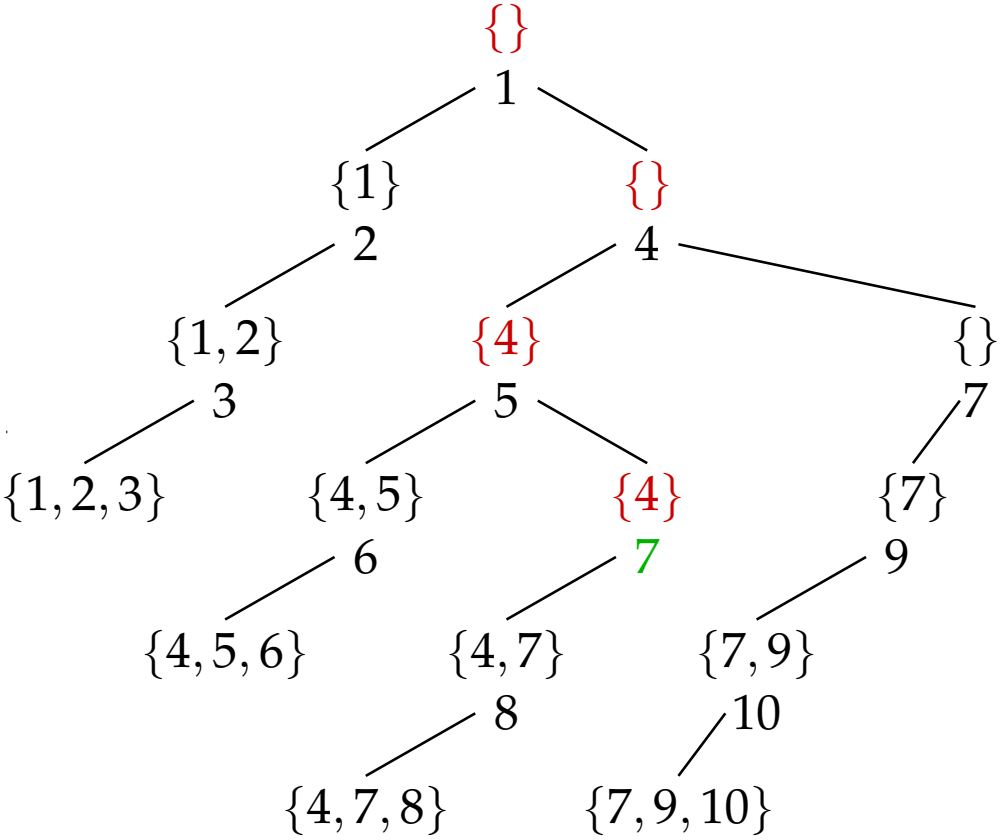
$U = \{4, 7, 1, 5\}$ $\quad G = \{3, 2, 6, 8\}$

Stateless Unfolding-based Partial-Order Reduction

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





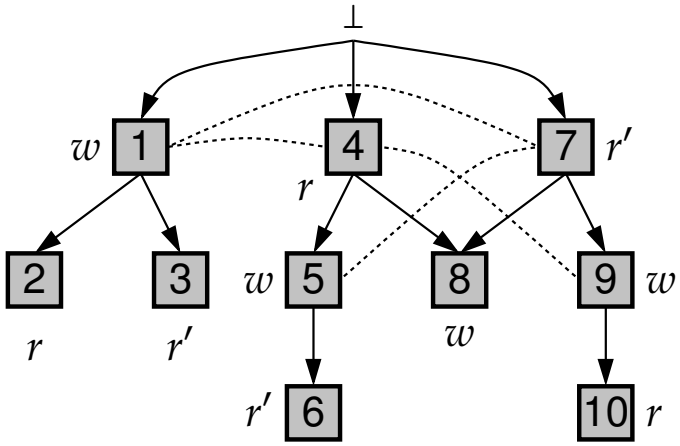


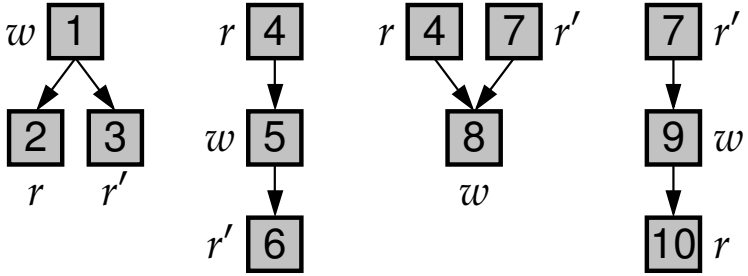
$C = \{4\}$ $\quad e = 7$ $\quad A = \{\}$

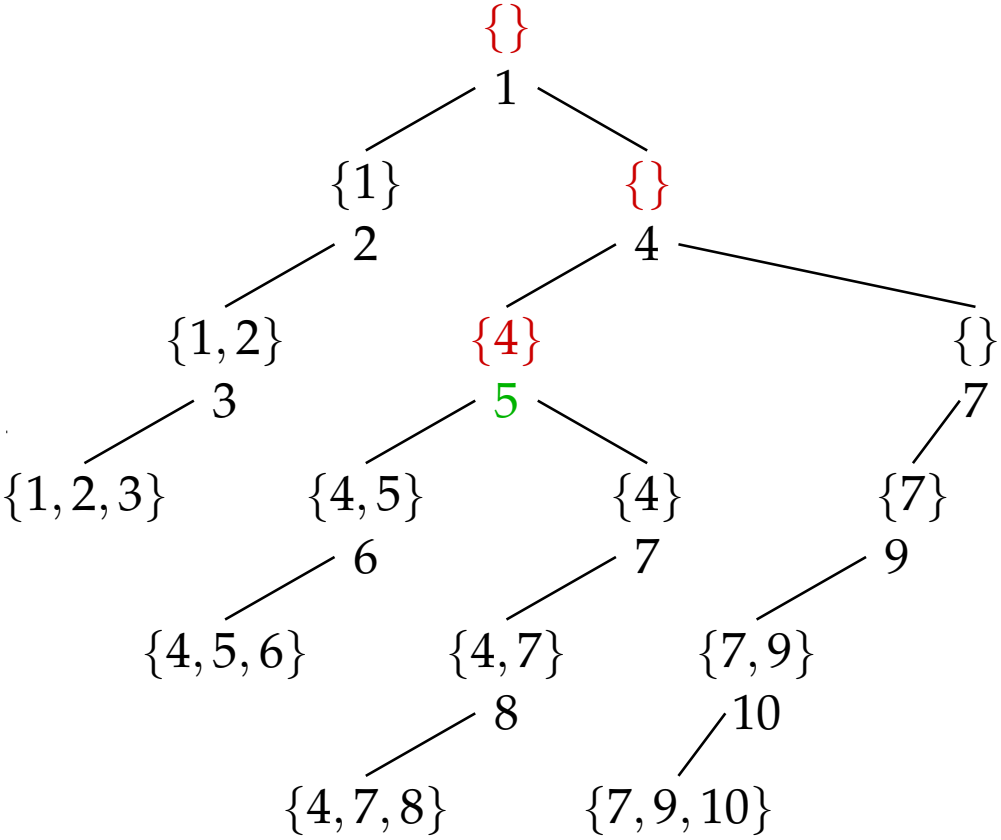
$U = \{4, 7, 1, 5\}$ $\quad G = \{3, 2, 6, 8\}$

Stateless Unfolding-based Partial-Order Reduction

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





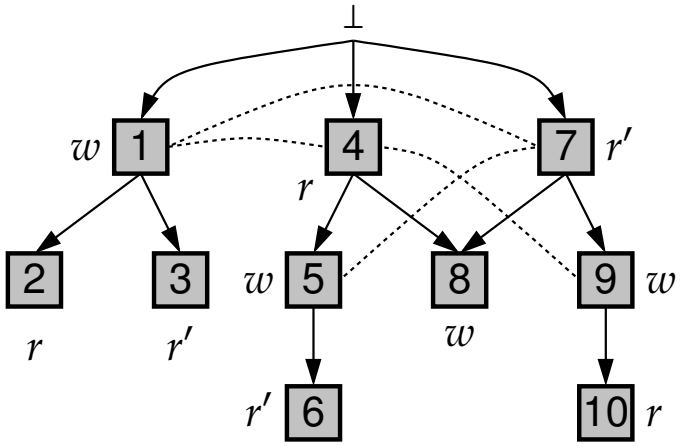


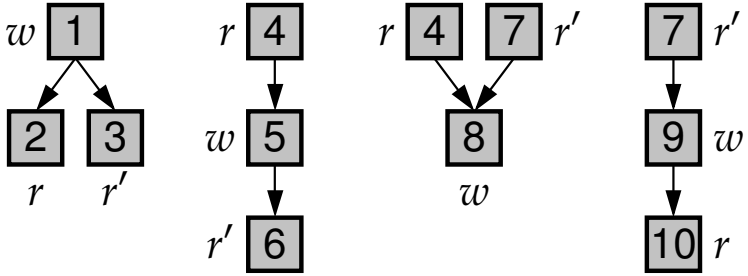
$C = \{4\}$ $\quad e = 5$ $\quad A = \{\}$

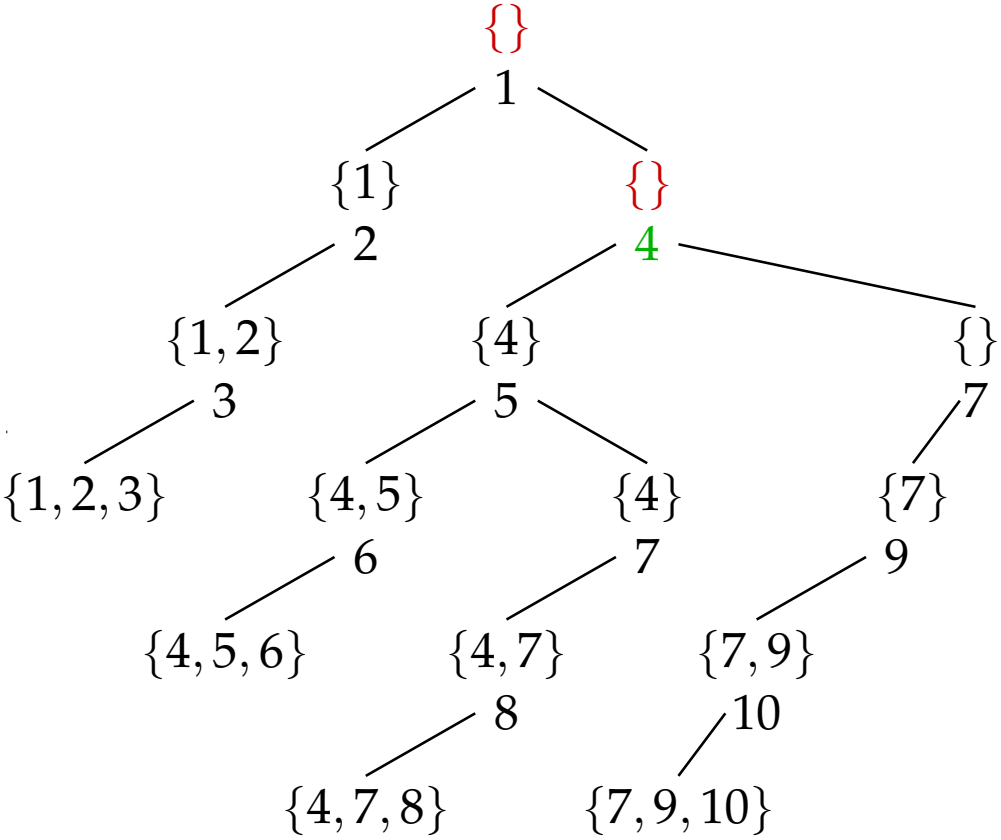
$U = \{4, 7, 1\}$ $\quad G = \{3, 2, 6, 8\}$

Stateless Unfolding-based Partial-Order Reduction

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





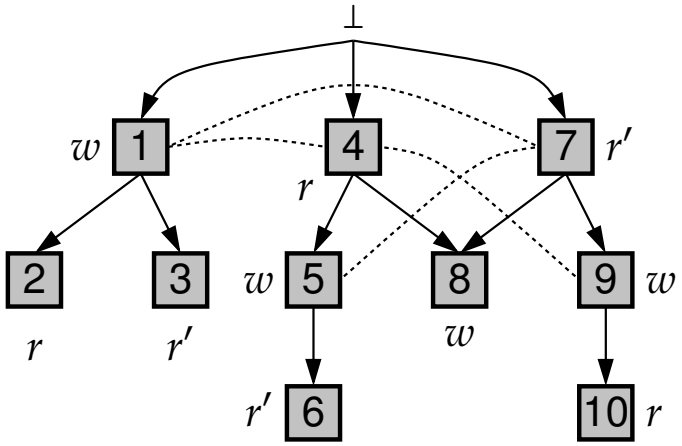


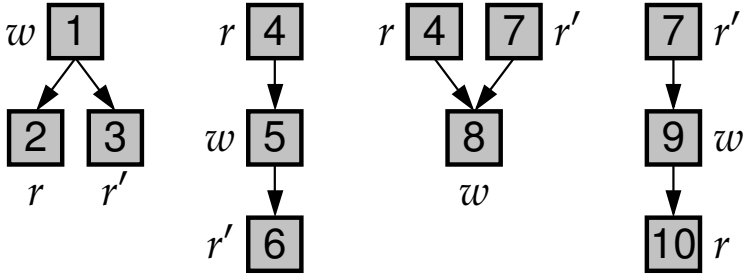
$C = \{\}$ $\quad e = 4 \quad A = \{\}$

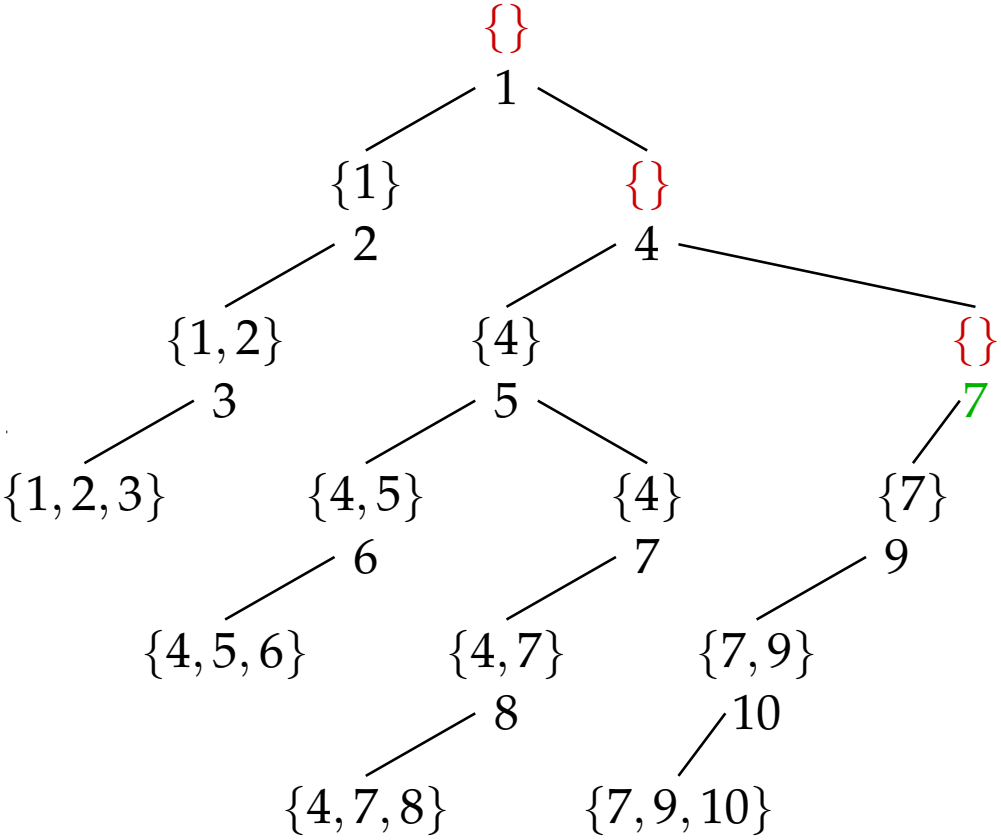
$U = \{4, 7, 1\}$ $\quad G = \{3, 2, 6, 8\}$

Stateless Unfolding-based Partial-Order Reduction

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





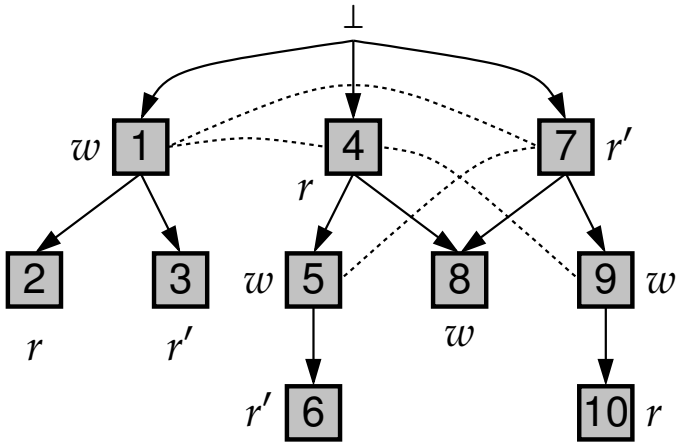


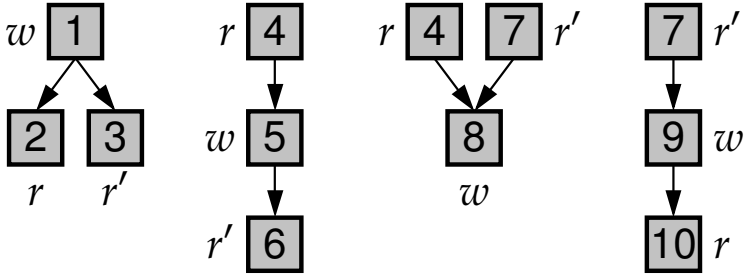
$C = \{\}$ $\quad e = 7 \quad A = \{7, 9\}$

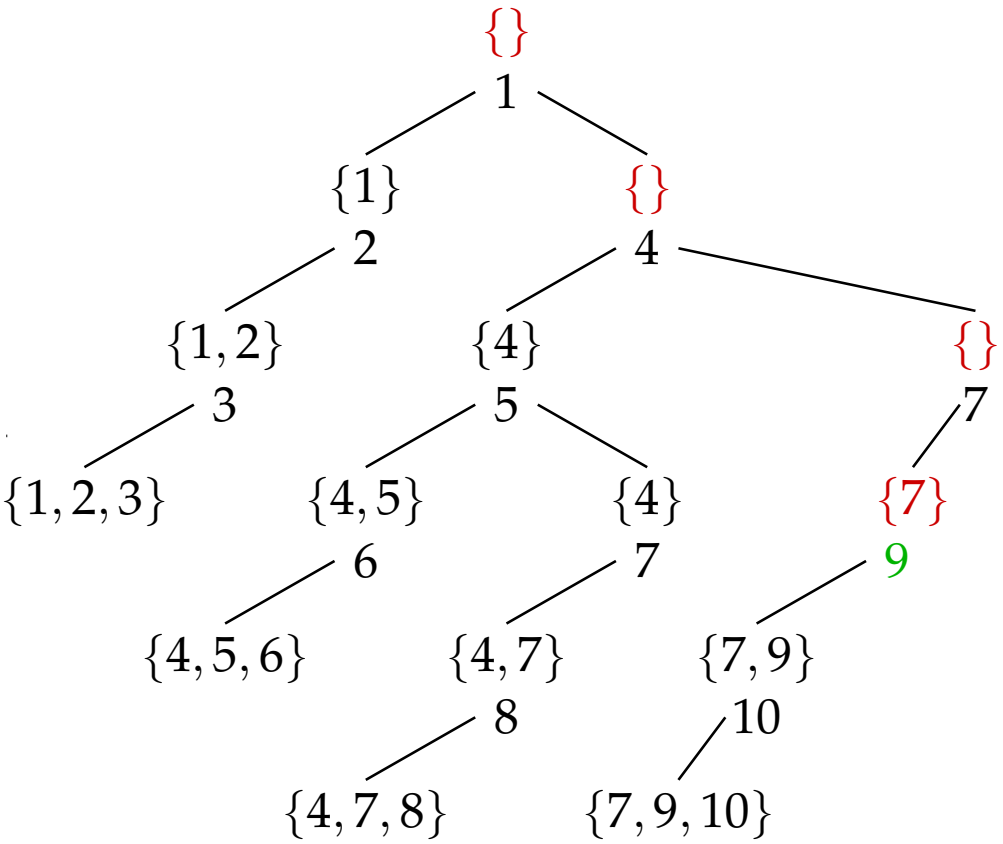
$U = \{7, 1, 4\}$ $\quad G = \{3, 2, 6, 8\}$

Stateless Unfolding-based Partial-Order Reduction

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





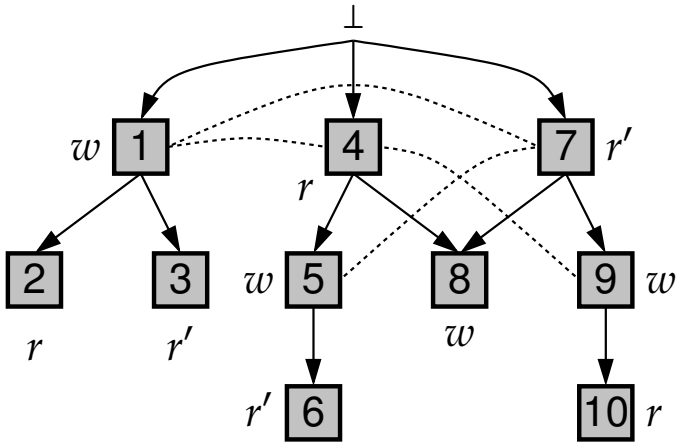


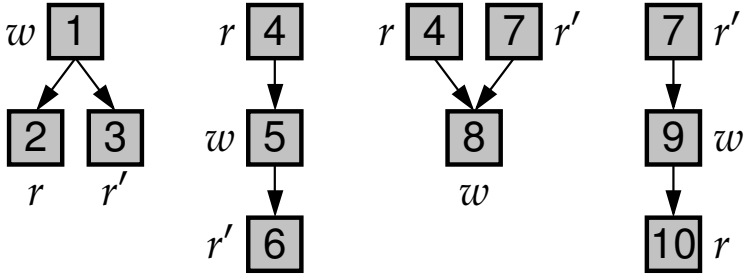
$C = \{7\}$ $\quad e = 9$ $\quad A = \{9\}$

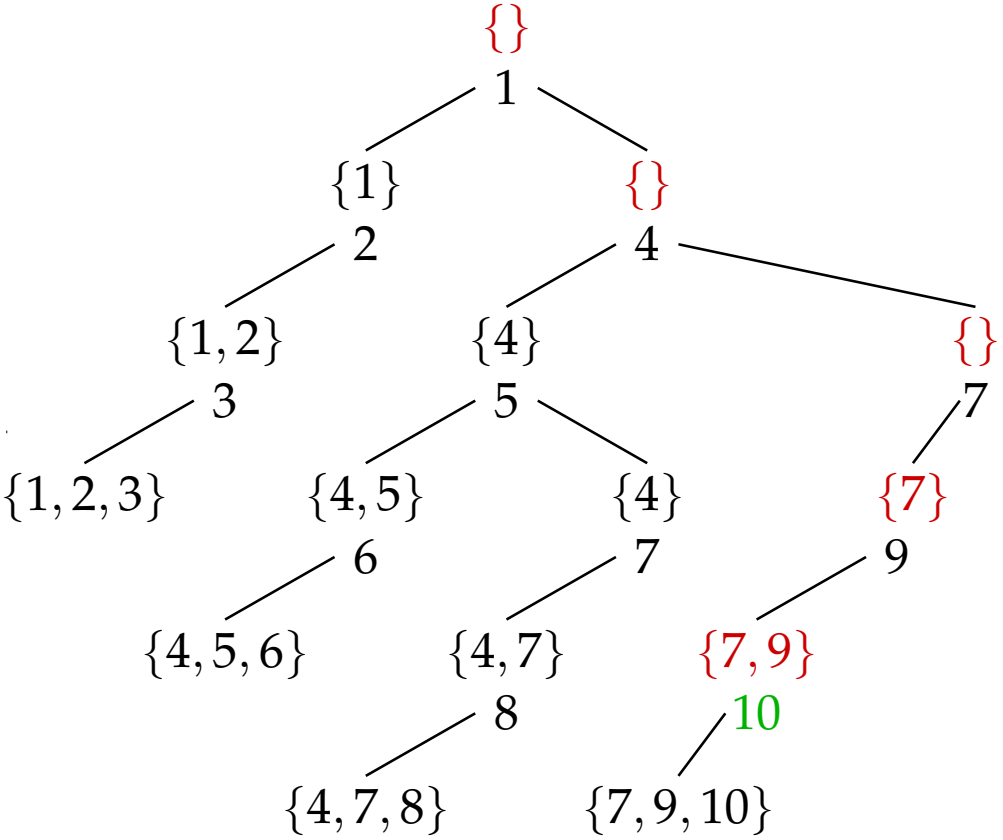
$U = \{7, 1, 4\}$ $\quad G = \{3, 2, 6, 8\}$

Stateless Unfolding-based Partial-Order Reduction

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





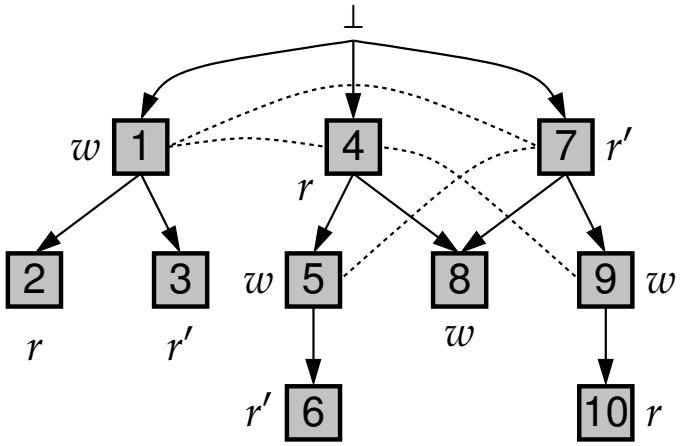


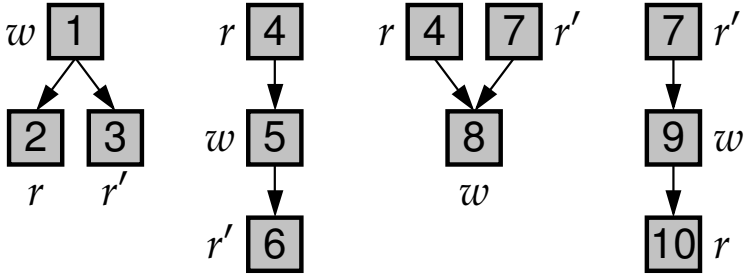
$C = \{7, 9\}$ $\quad e = 10$ $\quad A = \{\}$

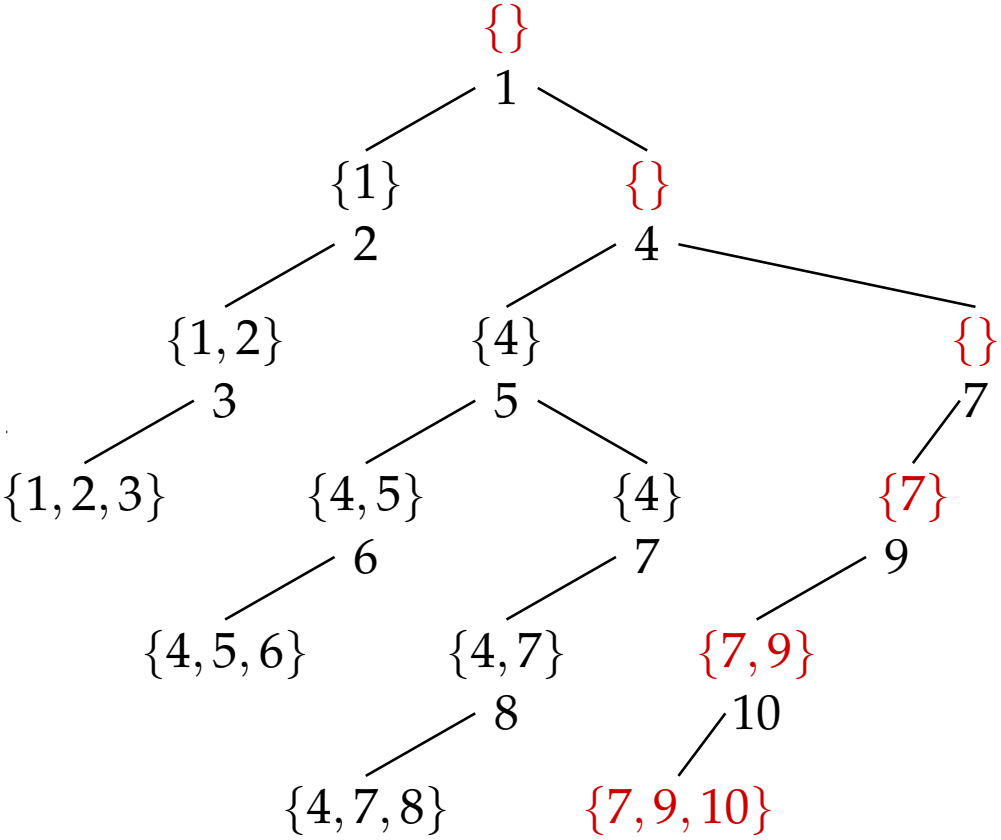
$U = \{7, 9, 1, 4\}$ $\quad G = \{3, 2, 6, 8\}$

Stateless Unfolding-based Partial-Order Reduction

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







$C = \{7, 9, 10\}$ $\quad e = n/a \quad$ $A = \{\}$

$U = \{7, 9, 10, 1, 4\}$ $\quad G = \{3, 2, 6, 8\}$

Termination, Completeness, Optimality

For terminating systems (acyclic state-space) we have:

Theorem

The previous algorithm:

1. *Always stops* *(termination)*
2. *Explores at least once every maximal configuration of* $\mathcal{U}_{M,\diamond}$ *(completeness)*
3. *Never explores twice any intermediate/maximal configuration* *(optimality)*

Next: we extend the algorithm to handle non-terminating executions

Cutoffs – Intuitions

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

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

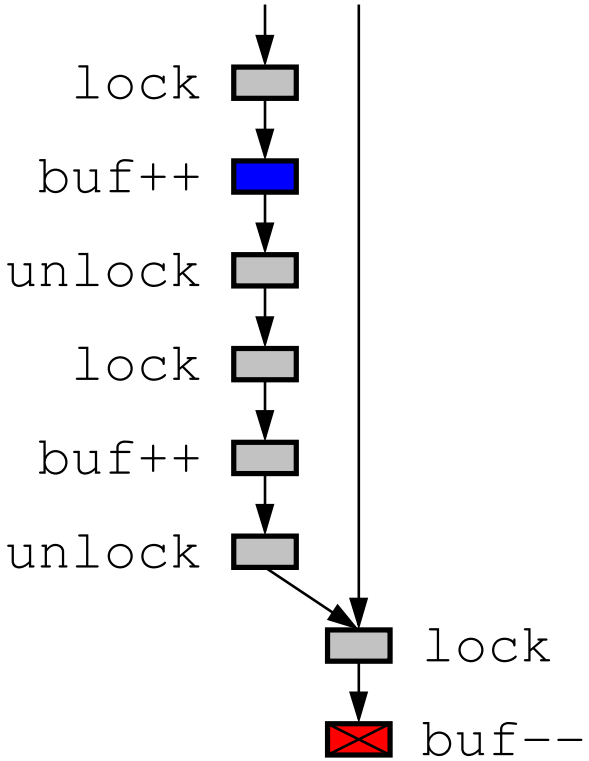
- $\overbrace{l,\ b+}^{\textit{Thread 1}}$

Cutoffs – Intuitions

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

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

$(m = 0, b = 1)$

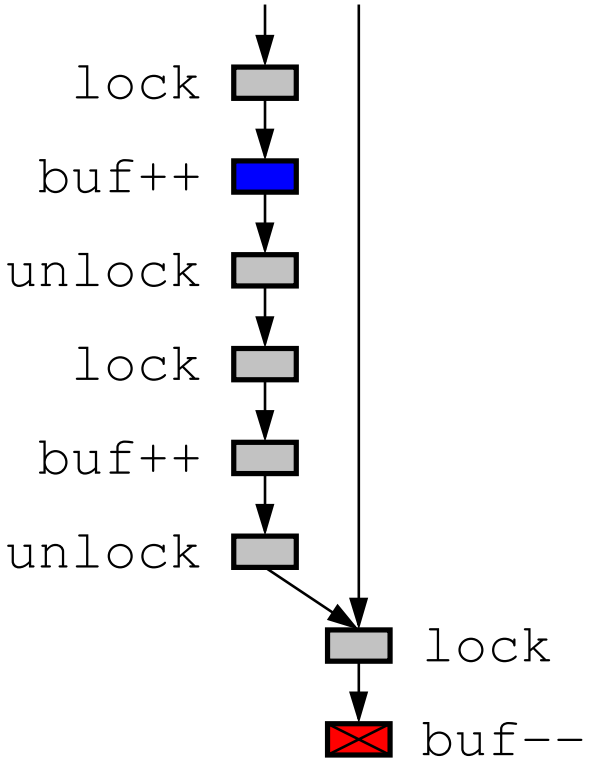


Cutoffs – Intuitions

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

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

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



Event e of $\mathcal{U}_{M,\diamond}$ is cutoff if

$\mathcal{U}_{M,\diamond}$ contains a corresponding event e' s.t.:

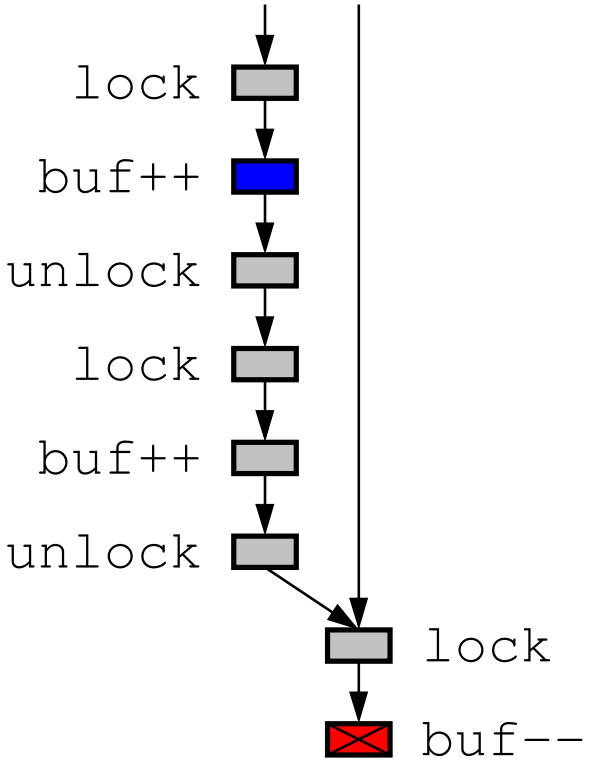
- $state([e]) = state([e'])$
- $|[e']| < |[e]|$

Cutoffs – Intuitions

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

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

- $\overbrace{l,\ b+,\ u,\ \ l,\ b+,\ u,}^{\textit{Thread 1}}\ \overbrace{l,\ b-}^{\textit{Thread 2}}$ $\quad (m = 0, b = 1)$
- $\overbrace{l,\ b+}^{\textit{Thread 1}}$
- Local configuration: $[e] := \{e' \in E : e' < e\}$



Event e of $\mathcal{U}_{M,\diamond}$ is cutoff if

$\mathcal{U}_{M,\diamond}$ contains a corresponding event e' s.t.:

- $state([e]) = state([e'])$
- $|[e']| < |[e]|$

Cutoffs – Application to our Algorithm

Problem: Preventing the algorithm from getting stuck in infinite executions

Solution: To explore event e if predicate $cutoff(e, U, G)$ evaluates to false

Definition

Predicate $cutoff(e, U, G)$ holds iff there is some $e' \in U \cup G$ s.t.

- $state([e]) = state([e'])$
- $|[e']| < |[e]|$

- $cutoff(e, U, G)$ evaluates differently on different executions (!!)

Theorem (Completeness)

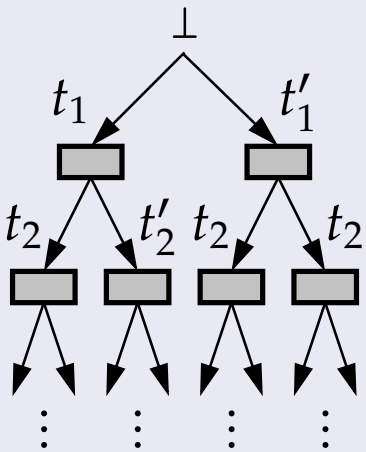
For each fireable transition t of a (possibly) non-terminating system M, the modified algorithm explores at least one t-labelled event.

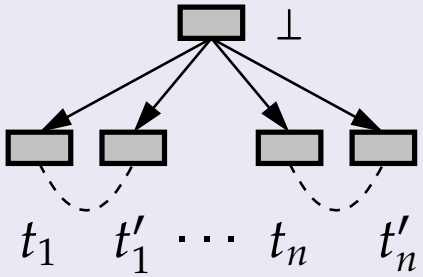
State Caching

- Garbage-collected events in G can be cleaned at discretion $\Longrightarrow$ cache memory

Cached events exploited in two ways:

1. Avoid reconstruction (causality, conflict) when reinserting in U (up to $\mathcal{O}(2^n)$ times!)





2. Improve chances of cutting off earlier the unfolding

By contrast:

- Most efficient PORs up to date are stateless (underuse memory)
- Unfolding approaches are stateful (overload memory)

Implementation and Experiments

New tool: POET (Partial-Order Exploration Tool)

- Deterministic C programs, POSIX threads
- Prototype in Haskell

Goals of the experiments:

- To investigate characteristics of average program unfoldings (depth, width)
- Frequency and impact of cutoffs
- Compare with NIDHUGG, *quasi-optimal* recent DPOR [Abdulla et al., POPL'14]

Benchmarks:

- Mainly come from the SV-COMP'14

Tool and experiments available online:

`http://www.cs.ox.ac.uk/people/marcelo.sousa/poet/`

Experiments — Acyclic State-Space

Benchmark		NIDHUGG			POET			
Name	$\|P\|$	$\|I\|$	$\|B\|$	$t(s)$	$\|E\|$	$\|E_{\text{cut}}\|$	$\|\Omega\|$	$t(s)$
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 — 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

Concluding Remarks

Results:

- Parametrized unfolding semantics exploiting provided independence relation
- Super-optimal POR algorithm, handling non-terminating executions
- State caching: compromise between stateless and stateful

Consequences:

- Identifies additional structure in the POR tree
- Indirectly maps key POR notions to well-established unfolding theory
- Makes possible comparison between unfoldings and PORs for, e.g., programs

Future work:

- Exploiting additional structure