Accelerating Predicate Abstraction for Probabilistic Automata

Dimitri Bohlender

RWTH Aachen University

September 12, 2014 / Master Thesis Presentation

Motivation

Why Model Checking?

Motivation

Why Model Checking?

- testing cannot prove absence of bugs

Motivation

Why Model Checking?

- testing cannot prove absence of bugs
- formal proof

Motivation

Why Model Checking?

- testing cannot prove absence of bugs
- formal proof

Properties

- eventually a collision free transmission occurs

Motivation

Why Model Checking?

- testing cannot prove absence of bugs
- formal proof

Properties

- eventually a collision free transmission occurs
- no collision ever occurs

Motivation

Why Model Checking?

- testing cannot prove absence of bugs
- formal proof

Why Probabilistic Model Checking?

Properties

- eventually a collision free transmission occurs
- no collision ever occurs

Motivation

Why Model Checking?

- testing cannot prove absence of bugs
- formal proof

Why Probabilistic Model Checking?

Verification of probabilistic models, e.g. network protocols

Properties

- eventually a collision free transmission occurs
- no collision ever occurs

Motivation

Why Model Checking?

- testing cannot prove absence of bugs
- formal proof

Why Probabilistic Model Checking?

Verification of probabilistic models, e.g. network protocols

Properties

- eventually a collision free transmission occurs
- no collision ever occurs
- probability for a collision is below 5%

Motivation

State Space Explosion

Even “simple” system descriptions yield huge state spaces

Motivation

State Space Explosion

Even "simple" system descriptions yield huge state spaces

$\Rightarrow$ construction & analysis often infeasible (memory & time constraints)

Motivation

State Space Explosion

Even "simple" system descriptions yield huge state spaces

$\Rightarrow$ construction & analysis often infeasible (memory & time constraints)

Observation

Model often more detailed than necessary to check property of interest

Motivation

State Space Explosion

Even “simple” system descriptions yield huge state spaces

$\Rightarrow$ construction & analysis often infeasible (memory & time constraints)

Observation

Model often more detailed than necessary to check property of interest

Approaches

Motivation

State Space Explosion

Even "simple" system descriptions yield huge state spaces

⇒ construction & analysis often infeasible (memory & time constraints)

Observation

Model often more detailed than necessary to check property of interest

Approaches

- analyse over-approximating, abstract model instead

Motivation

State Space Explosion

Even “simple” system descriptions yield huge state spaces

$\Rightarrow$ construction & analysis often infeasible (memory & time constraints)

Observation

Model often more detailed than necessary to check property of interest

Approaches

- analyse over-approximating, abstract model instead (Menu-game)

Motivation

State Space Explosion

Even "simple" system descriptions yield huge state spaces

⇒ construction & analysis often infeasible (memory & time constraints)

Observation

Model often more detailed than necessary to check property of interest

Approaches

- analyse over-approximating, abstract model instead (Menu-game)
- use space-efficient, symbolic data structures

Motivation

State Space Explosion

Even "simple" system descriptions yield huge state spaces

⇒ construction & analysis often infeasible (memory & time constraints)

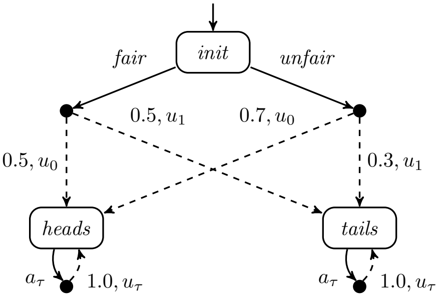
Observation

Model often more detailed than necessary to check property of interest

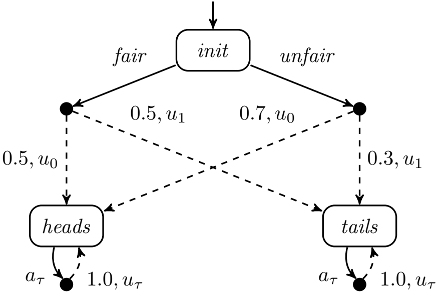
Approaches

- analyse over-approximating, abstract model instead (Menu-game)
- use space-efficient, symbolic data structures (BDD)

Probabilistic Automaton (Example)



Probabilistic Automaton (Example)

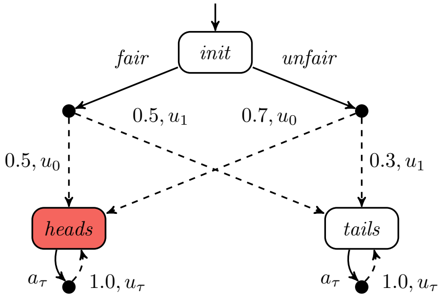


Probabilistic Reachability

- reachability depends on strategy σ of resolving non-determinism

$$Pr^{\sigma}_{\mathcal{A}}(\Diamond G)$$

Probabilistic Automaton (Example)

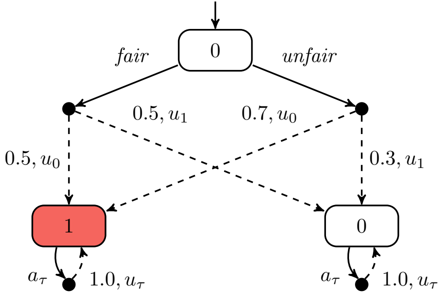


Probabilistic Reachability

- reachability depends on strategy σ of resolving non-determinism

$$Pr^{\sigma}_{\mathcal{A}}(\Diamond G)$$

Probabilistic Automaton (Example)



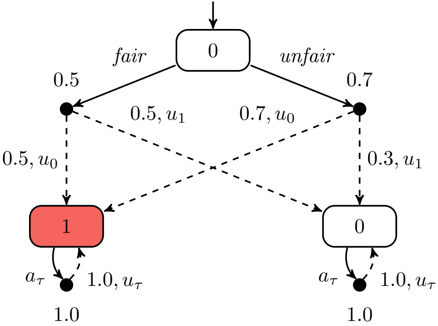
Probabilistic Reachability

- reachability depends on strategy σ of resolving non-determinism

$$Pr^{\sigma}_{\mathcal{A}}(\Diamond G)$$

- fixed point characterisation of extremal probabilities

Probabilistic Automaton (Example)



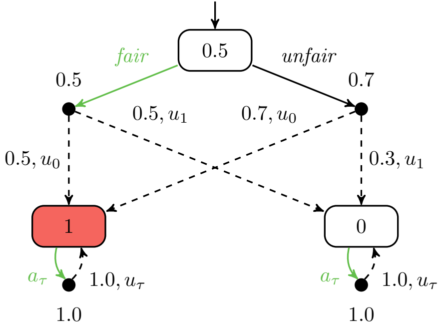
Probabilistic Reachability

- reachability depends on strategy σ of resolving non-determinism

$$Pr^{\sigma}_{\mathcal{A}}(\Diamond G)$$

- fixed point characterisation of extremal probabilities

Probabilistic Automaton (Example)



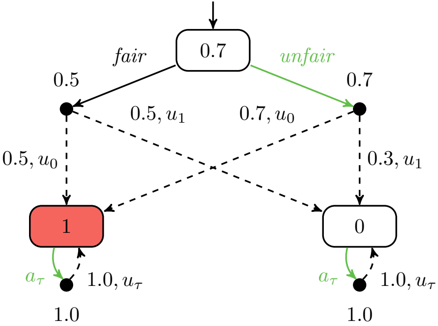
Probabilistic Reachability

- reachability depends on strategy σ of resolving non-determinism

$$Pr^{-}_{\mathcal{A}}(\Diamond G) \leq Pr^{\sigma}_{\mathcal{A}}(\Diamond G)$$

- fixed point characterisation of extremal probabilities

Probabilistic Automaton (Example)



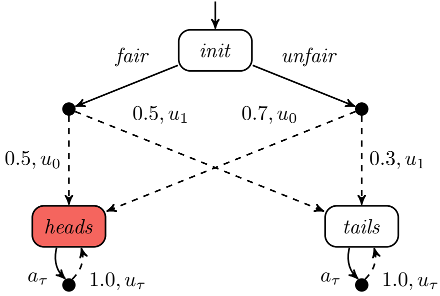
Probabilistic Reachability

- reachability depends on strategy σ of resolving non-determinism

$$Pr^{-}_{\mathcal{A}}(\Diamond G) \leq Pr^{\sigma}_{\mathcal{A}}(\Diamond G) \leq Pr^{+}_{\mathcal{A}}(\Diamond G)$$

- fixed point characterisation of extremal probabilities

Probabilistic Automaton (Example)



Probabilistic Reachability

- reachability depends on strategy σ of resolving non-determinism

$$Pr^-_{\mathcal{A}}(\Diamond G) \leq Pr^{\sigma}_{\mathcal{A}}(\Diamond G) \leq Pr^+_{\mathcal{A}}(\Diamond G)$$

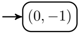
- fixed point characterisation of extremal probabilities

Probabilistic Program (Example)

```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```

Probabilistic Program (Example)

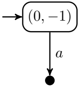
```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```



Legend: $(phase, run)$

Probabilistic Program (Example)

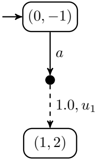
```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```



Legend: $(phase, run)$

Probabilistic Program (Example)

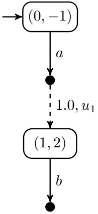
```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```



Legend: $(phase, run)$

Probabilistic Program (Example)

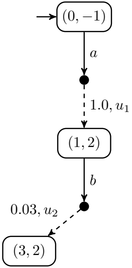
```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```



Legend: $(phase, run)$

Probabilistic Program (Example)

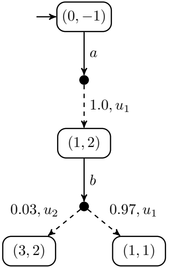
```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```



Legend: $(phase, run)$

Probabilistic Program (Example)

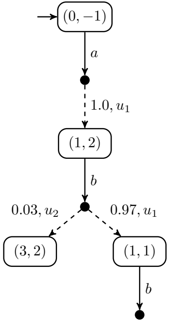
```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```



Legend: $(phase, run)$

Probabilistic Program (Example)

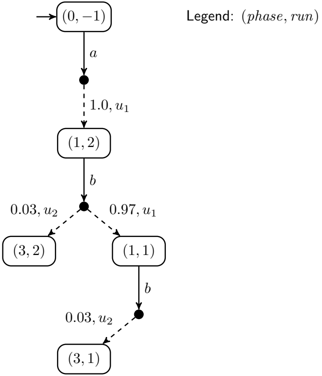
```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```



Legend: $(phase, run)$

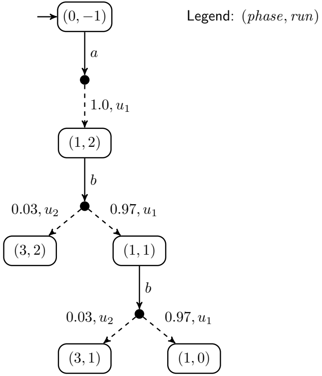
Probabilistic Program (Example)

```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```



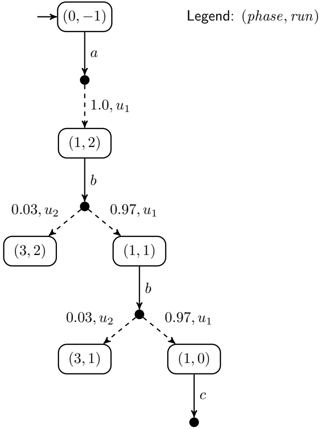
Probabilistic Program (Example)

```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```



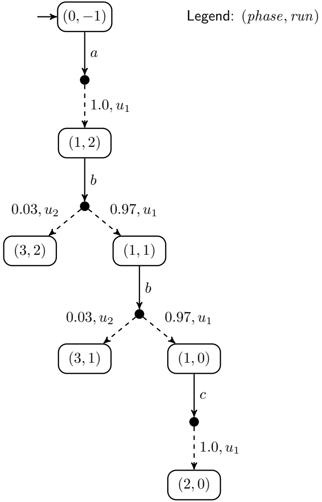
Probabilistic Program (Example)

```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```



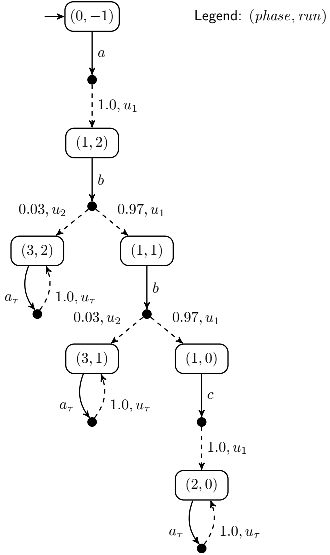
Probabilistic Program (Example)

```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```

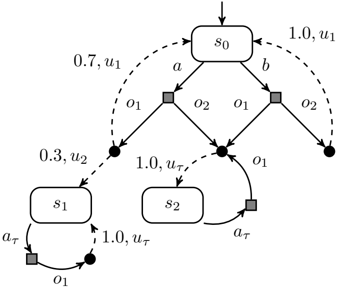


Probabilistic Program (Example)

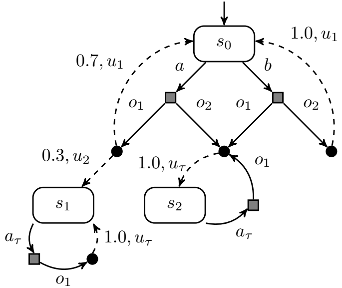
```
module simple
    // 0=init, 1=running,
    // 2=finished, 3=broken
    phase : [0..3];
    run   : int;

    [a] phase=0
        -> 1.0 :(run'=2) & (phase'=1);

    [b] phase=1 & run>0
        -> 0.97:(run'=run-1)
         + 0.03:(phase'=3);

    [c] phase=1 & run<=0
        -> 1.0 :(phase'=2);
endmodule

init
    phase=0 & run=-1;
endinit
```



Stochastic Game (Example)



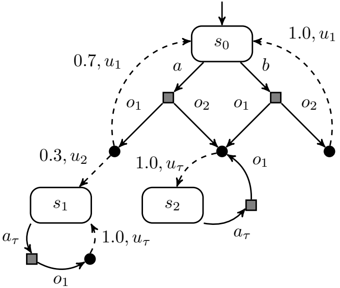
Stochastic Game (Example)



Probabilistic Reachability

Reachability probability $Pr_{\mathcal{G}}^{\sigma_1,\sigma_2}(\Diamond G)$ depends on strategy-pair (σ_1, σ_2)

Stochastic Game (Example)

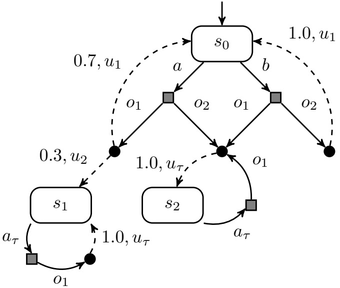


Probabilistic Reachability

Reachability probability $Pr_{\mathcal{G}}^{\sigma_1,\sigma_2}(\Diamond G)$ depends on strategy-pair (σ_1, σ_2)

Extremal probabilities

Stochastic Game (Example)



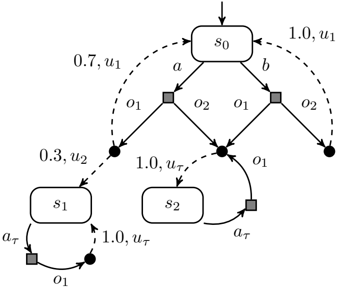
Probabilistic Reachability

Reachability probability $Pr_{\mathcal{G}}^{\sigma_1,\sigma_2}(\Diamond G)$ depends on strategy-pair (σ_1, σ_2)

Extremal probabilities

$$Pr_{\mathcal{G}}^{-,-}(\Diamond G)$$

Stochastic Game (Example)



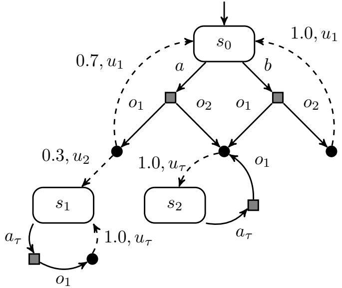
Probabilistic Reachability

Reachability probability $Pr_{\mathcal{G}}^{\sigma_1,\sigma_2}(\lozenge G)$ depends on strategy-pair (σ_1, σ_2)

Extremal probabilities

$$Pr_{\mathcal{G}}^{-,-}(\lozenge G) \qquad Pr_{\mathcal{G}}^{-,+}(\lozenge G)$$

Stochastic Game (Example)



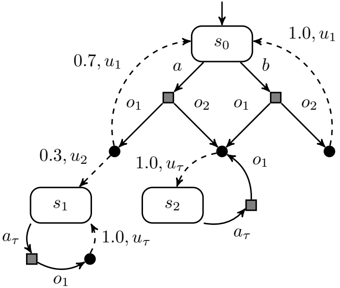
Probabilistic Reachability

Reachability probability $Pr_{\mathcal{G}}^{\sigma_1,\sigma_2}(\Diamond G)$ depends on strategy-pair (σ_1, σ_2)

Extremal probabilities

$$Pr_{\mathcal{G}}^{-,-}(\Diamond G) \qquad Pr_{\mathcal{G}}^{-,+}(\Diamond G)$$
$$Pr_{\mathcal{G}}^{+,-}(\Diamond G)$$

Stochastic Game (Example)



Probabilistic Reachability

Reachability probability $Pr_{\mathcal{G}}^{\sigma_1,\sigma_2}(\Diamond G)$ depends on strategy-pair (σ_1, σ_2)

Extremal probabilities

$$Pr_{\mathcal{G}}^{-,-}(\Diamond G) \qquad Pr_{\mathcal{G}}^{-,+}(\Diamond G)$$
$$Pr_{\mathcal{G}}^{+,-}(\Diamond G) \qquad Pr_{\mathcal{G}}^{+,+}(\Diamond G)$$

MTBDD

Observation

In practice, state spaces exhibit symmetries

MTBDD

Observation

In practice, state spaces exhibit symmetries
$\Rightarrow$ exploit by employing symbolical representation

MTBDD

Observation

In practice, state spaces exhibit symmetries
$\Rightarrow$ exploit by employing symbolical representation

Multi-Terminal Binary Decision Diagram

DAG $\mathfrak{D}$ representing a function $f_{\mathfrak{D}} : \mathbb{B}^n \to \mathbb{R}$ with finite range

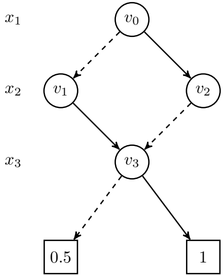
MTBDD

Observation

In practice, state spaces exhibit symmetries
$\Rightarrow$ exploit by employing symbolical representation

Multi-Terminal Binary Decision Diagram

DAG $\mathfrak{D}$ representing a function $f_{\mathfrak{D}} : \mathbb{B}^n \rightarrow \mathbb{R}$ with finite range



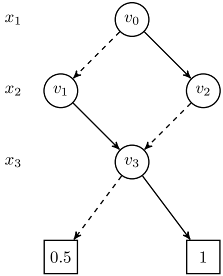
MTBDD

Observation

In practice, state spaces exhibit symmetries
$\Rightarrow$ exploit by employing symbolical representation

Multi-Terminal Binary Decision Diagram

DAG $\mathfrak{D}$ representing a function $f_{\mathfrak{D}} : \mathbb{B}^n \rightarrow \mathbb{R}$ with finite range



x_1	x_2	x_3	$f_{\mathfrak{D}}$
0	0	0	0
0	0	1	0
0	1	0	0.5
0	1	1	1
1	0	0	0.5
1	0	1	1
1	1	0	0
1	1	1	0

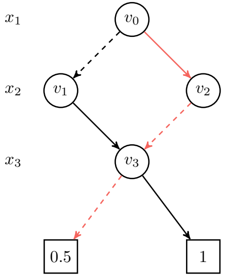
MTBDD

Observation

In practice, state spaces exhibit symmetries
$\Rightarrow$ exploit by employing symbolical representation

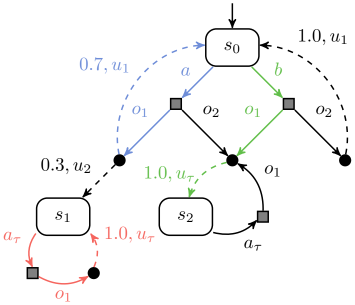
Multi-Terminal Binary Decision Diagram

DAG $\mathfrak{D}$ representing a function $f_{\mathfrak{D}} : \mathbb{B}^n \to \mathbb{R}$ with finite range

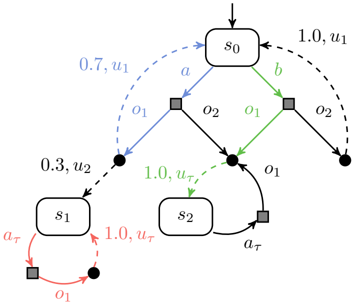


x_1	x_2	x_3	$f_{\mathfrak{D}}$
0	0	0	0
0	0	1	0
0	1	0	0.5
0	1	1	1
1	0	0	0.5
1	0	1	1
1	1	0	0
1	1	1	0

Stochastic Game as MTBDD (Example)



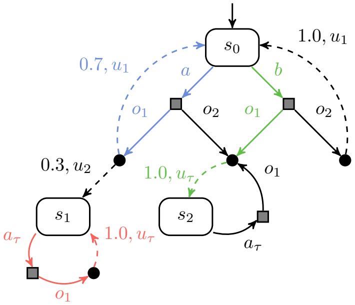
Stochastic Game as MTBDD (Example)



Encoding Excerpt

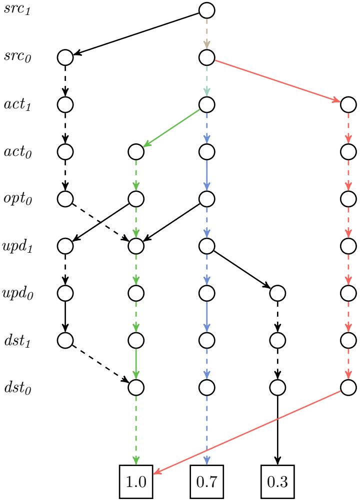
$$\delta(s_0, b, o_1, u_\tau, s_2) = 1.0$$

Stochastic Game as MTBDD (Example)



Encoding Excerpt

$$\delta(s_0, b, o_1, u_\tau, s_2) = 1.0$$



Menu-game: Concept

Partition Abstraction

Partition PA's state space S into blocks Q:

$$S = \biguplus_{B \in Q} B$$

Menu-game: Concept

Partition Abstraction

Partition PA's state space S into blocks Q:

$$S = \biguplus_{B \in Q} B$$

Non-determinism of Model & Abstraction

- merge non-determinism
- distinguish non-determinism

Menu-game: Concept

Partition Abstraction

Partition PA's state space S into blocks Q:

$$S = \biguplus_{B \in Q} B$$

Non-determinism of Model & Abstraction

- merge non-determinism $\Rightarrow$ yields PA
- distinguish non-determinism

Menu-game: Concept

Partition Abstraction

Partition PA's state space S into blocks Q:

$$S = \biguplus_{B \in Q} B$$

Non-determinism of Model & Abstraction

- merge non-determinism $\Rightarrow$ yields PA
- distinguish non-determinism $\Rightarrow$ yields a SG

Menu-game: Concept

Partition Abstraction

Partition PA's state space S into blocks Q:

$$S = \biguplus_{B \in Q} B$$

Non-determinism of Model & Abstraction

- merge non-determinism ⇒ yields PA
- distinguish non-determinism ⇒ yields a SG

Over-approximation [Wachter, 2011]

$$Pr_{\mathcal{G}}^{-,-}\left(\Diamond G^{\#}\right) \;\leq\; Pr_{\mathcal{A}}^{-}(\Diamond G) \;\leq\; Pr_{\mathcal{G}}^{-,+}\left(\Diamond G^{\#}\right)$$

Menu-game: Concept

Partition Abstraction

Partition PA's state space S into blocks Q:

$$S = \biguplus_{B \in Q} B$$

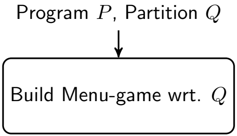
Non-determinism of Model & Abstraction

- merge non-determinism $\Rightarrow$ yields PA
- distinguish non-determinism $\Rightarrow$ yields a SG

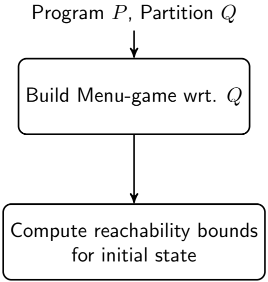
Over-approximation [Wachter, 2011]

$$\begin{array}{ccccc} Pr_{\mathcal{G}}^{-,-}(\Diamond G^{\#}) & \leq & Pr_{\mathcal{A}}^{-}(\Diamond G) & \leq & Pr_{\mathcal{G}}^{-,+}(\Diamond G^{\#}) \\ Pr_{\mathcal{G}}^{+,-}(\Diamond G^{\#}) & \leq & Pr_{\mathcal{A}}^{+}(\Diamond G) & \leq & Pr_{\mathcal{G}}^{+,+}(\Diamond G^{\#}) \end{array}$$

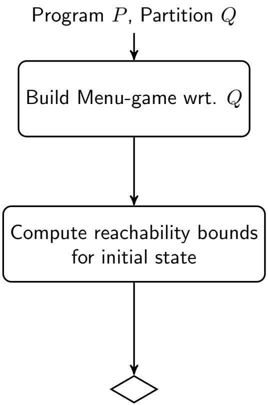
Backward Refinement Scheme



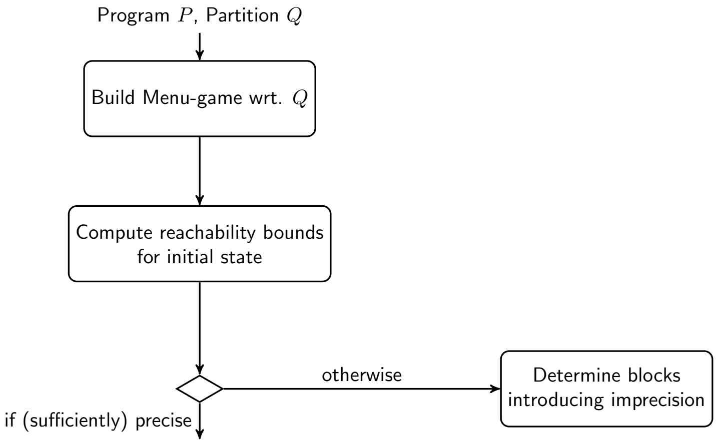
Backward Refinement Scheme



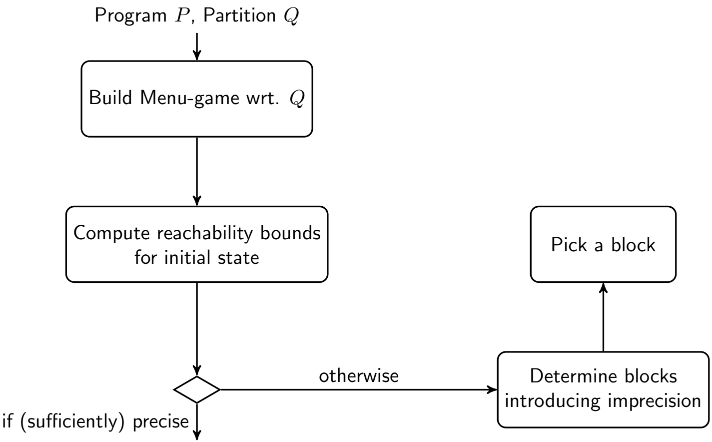
Backward Refinement Scheme



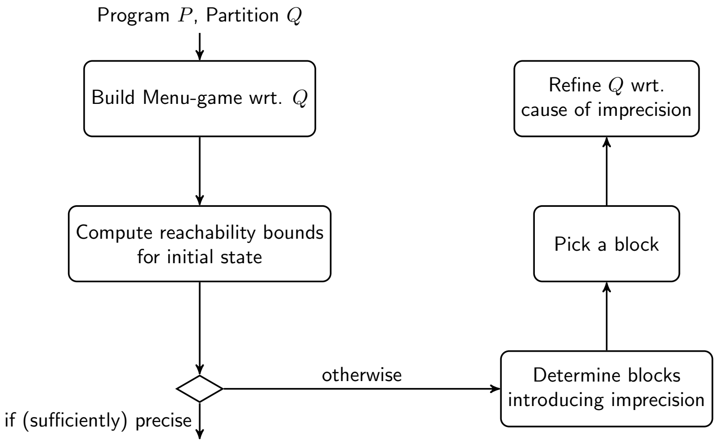
Backward Refinement Scheme



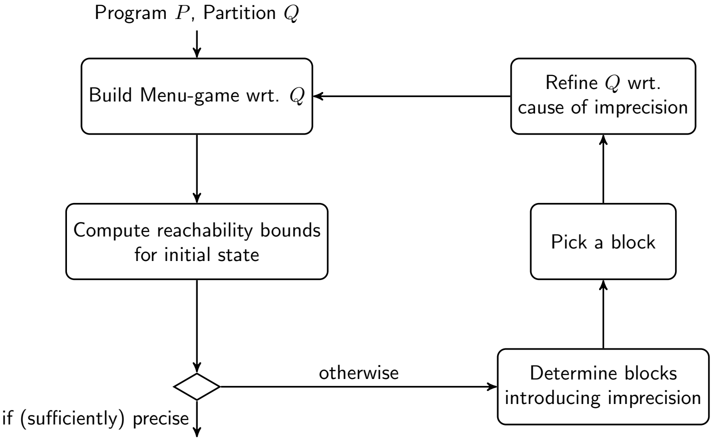
Backward Refinement Scheme



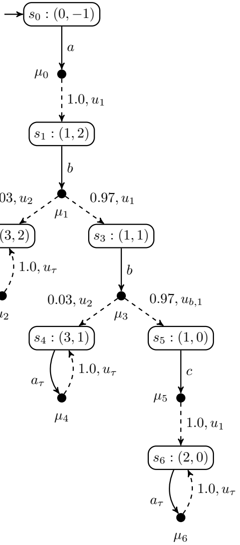
Backward Refinement Scheme



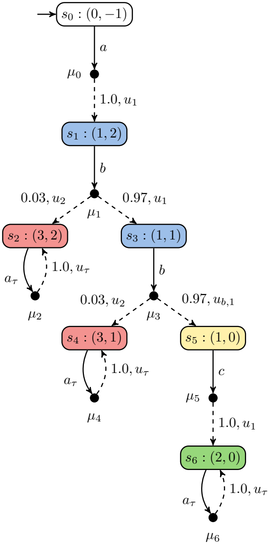
Backward Refinement Scheme



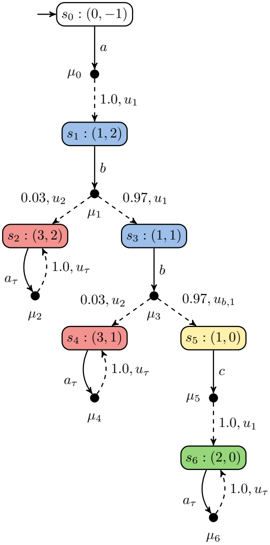
Menu-game: Construction from PA (Example)

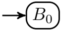


Menu-game: Construction from PA (Example)

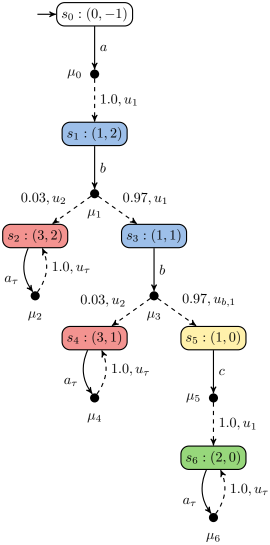


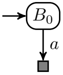
Menu-game: Construction from PA (Example)



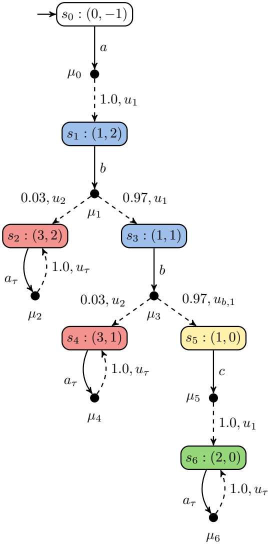


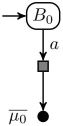
Menu-game: Construction from PA (Example)



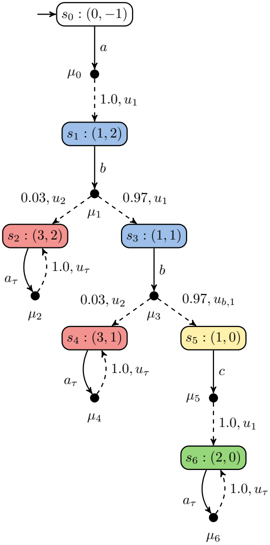


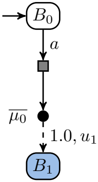
Menu-game: Construction from PA (Example)



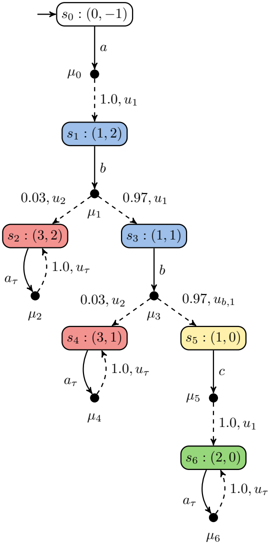


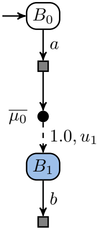
Menu-game: Construction from PA (Example)



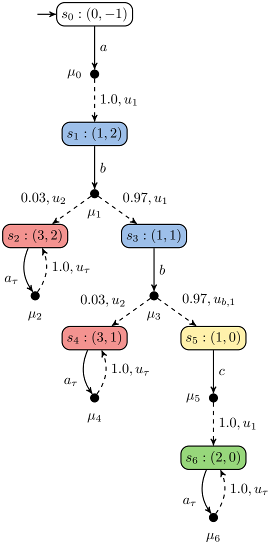


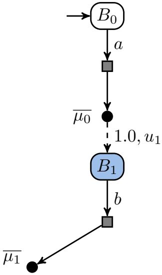
Menu-game: Construction from PA (Example)



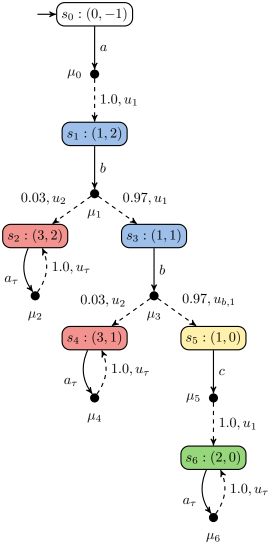


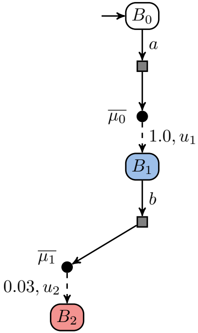
Menu-game: Construction from PA (Example)



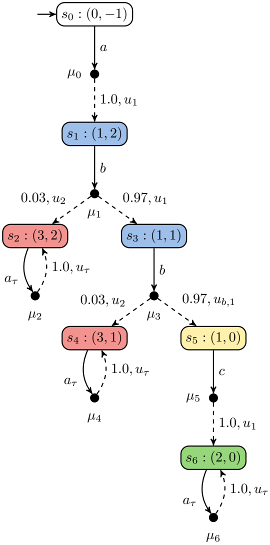


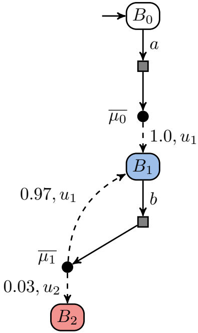
Menu-game: Construction from PA (Example)



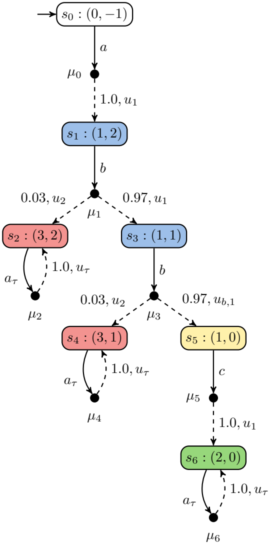


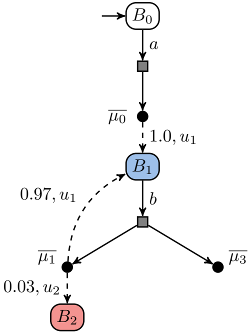
Menu-game: Construction from PA (Example)



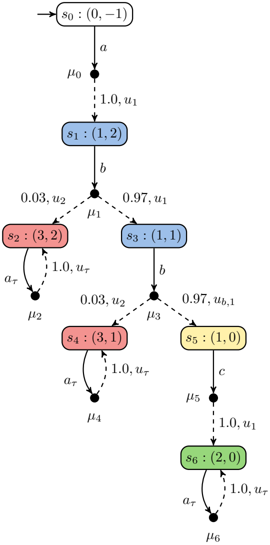


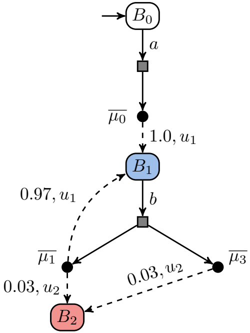
Menu-game: Construction from PA (Example)



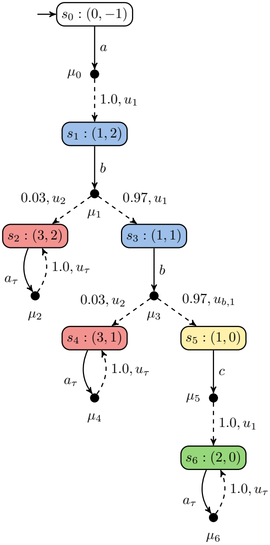


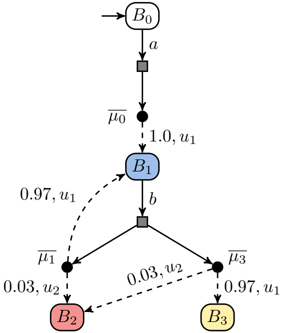
Menu-game: Construction from PA (Example)



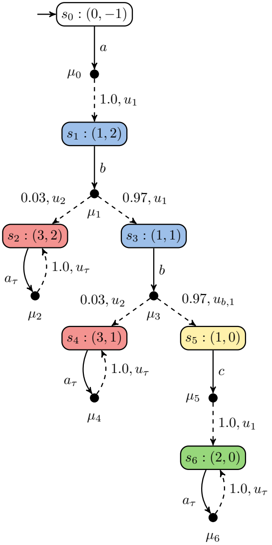


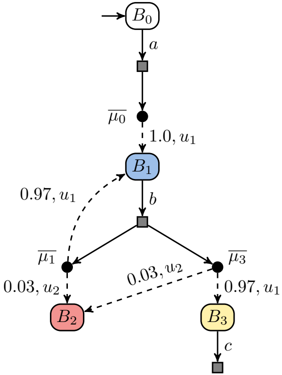
Menu-game: Construction from PA (Example)



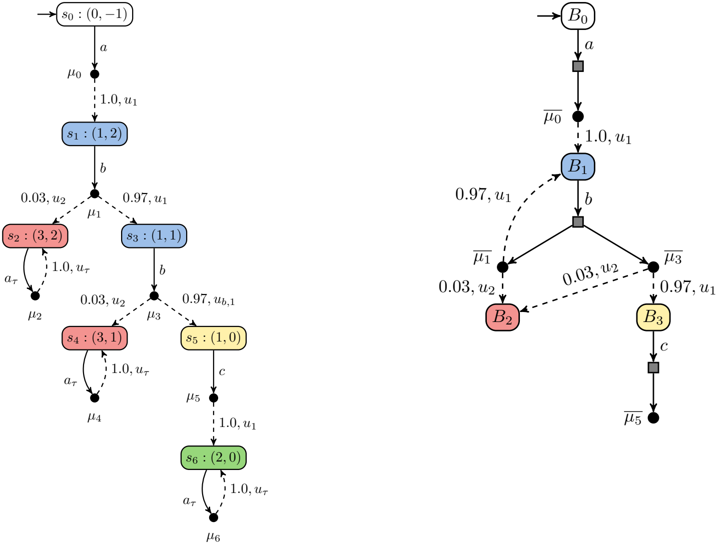


Menu-game: Construction from PA (Example)

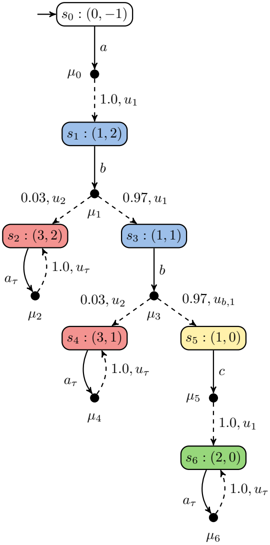


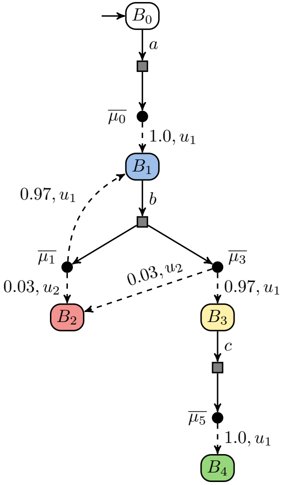


Menu-game: Construction from PA (Example)

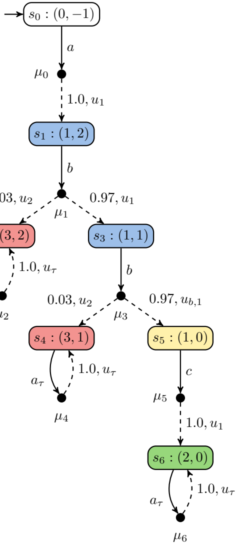


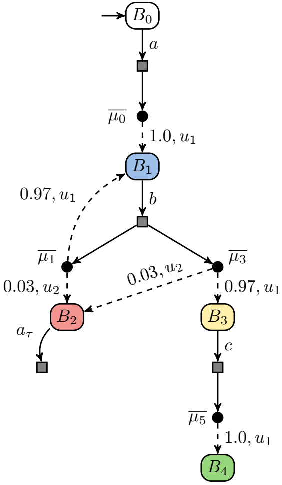
Menu-game: Construction from PA (Example)



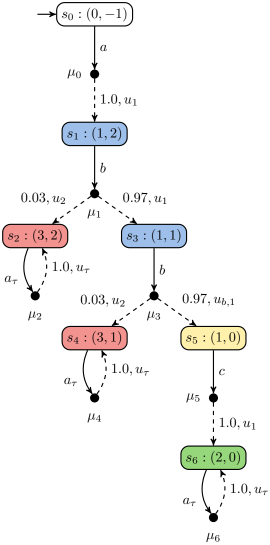


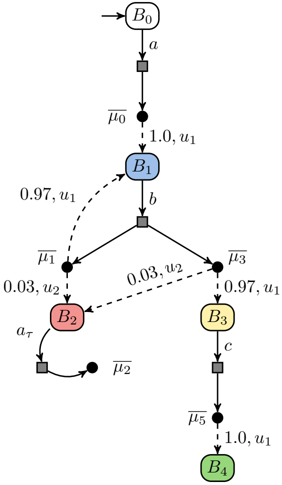
Menu-game: Construction from PA (Example)



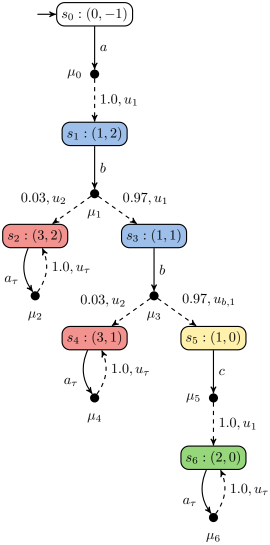


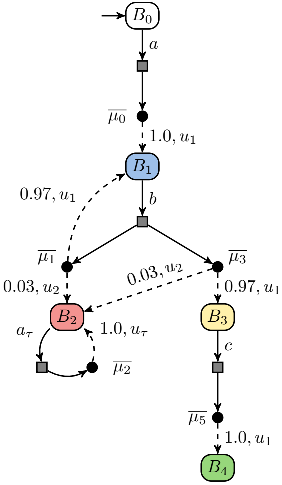
Menu-game: Construction from PA (Example)



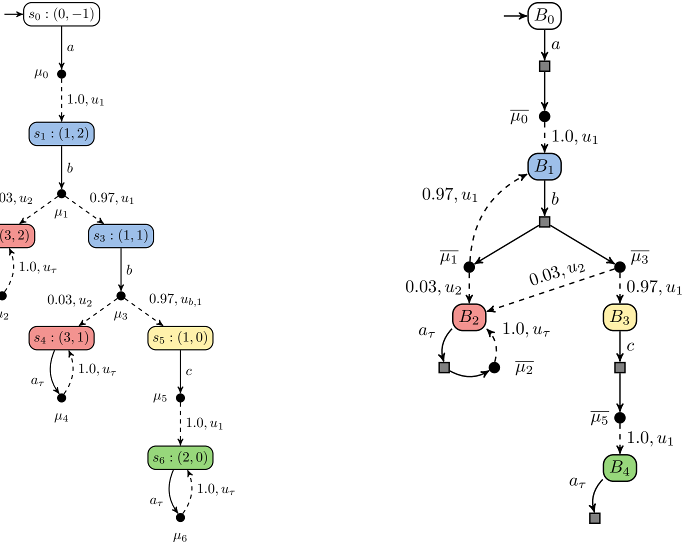


Menu-game: Construction from PA (Example)

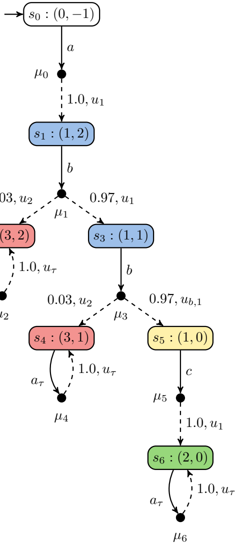


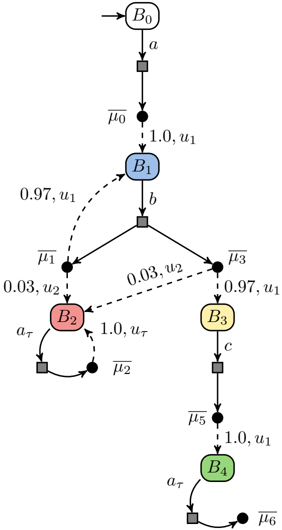


Menu-game: Construction from PA (Example)

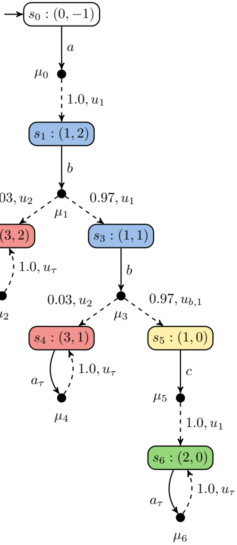


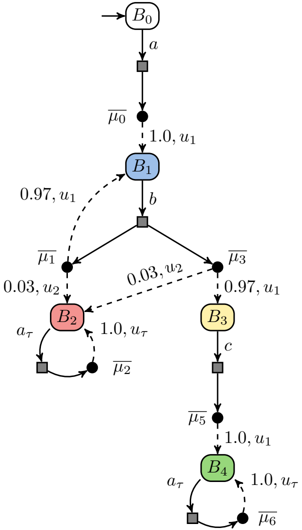
Menu-game: Construction from PA (Example)



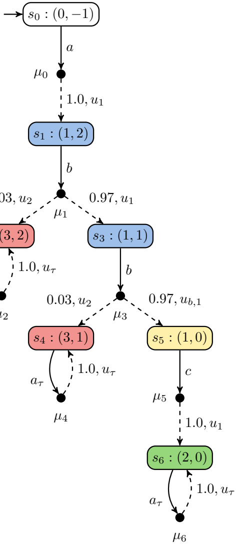


Menu-game: Construction from PA (Example)





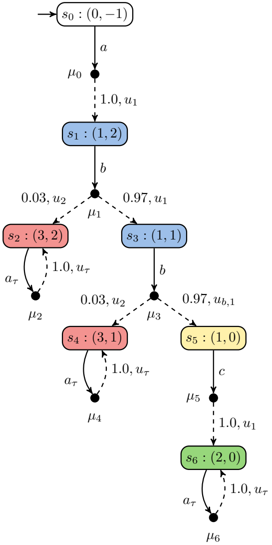
Menu-game: Predicate Abstraction



Predicate

Boolean expression over a program's variables

Menu-game: Predicate Abstraction



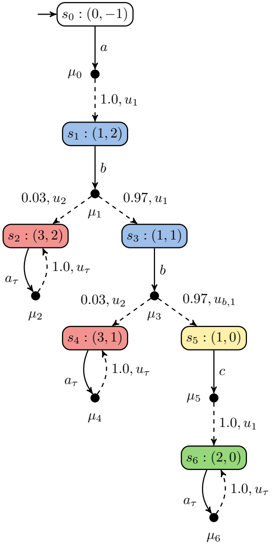
Predicate

Boolean expression over a program's variables

Predicates induce partitioning

$$\mathcal{P} = \{phase = 0, phase = 1, phase = 2, phase = 3, run > 0\}$$

Menu-game: Predicate Abstraction



Predicate

Boolean expression over a program's variables

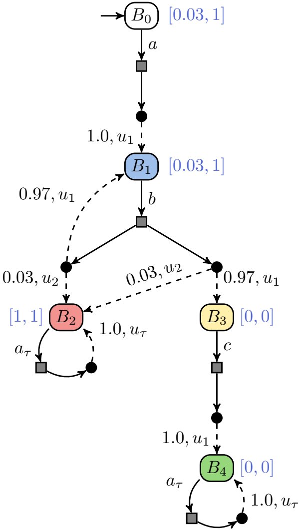
Predicates induce partitioning

$$\mathcal{P} = \{phase = 0, phase = 1, phase = 2, phase = 3, run > 0\}$$

induces the partition:

- $phase = 0$
- $phase = 1, run > 0$
- $phase = 1$
- $phase = 2, run > 0$
- $phase = 3, run > 0$

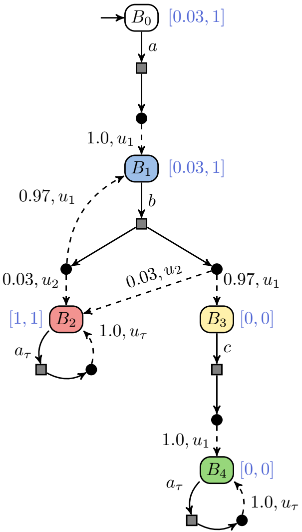
Refinement



Idea

Split *pivot blocks*, which introduce imprecision

Refinement



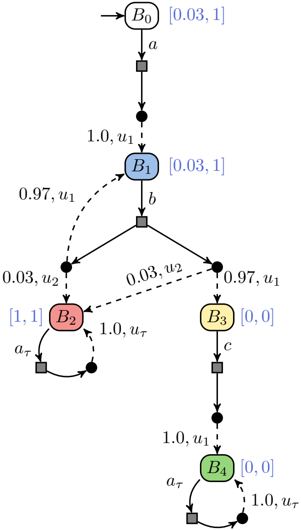
Idea

Split *pivot blocks*, which introduce imprecision

Observations

- deviation alone does not indicate a block being pivot

Refinement



Idea

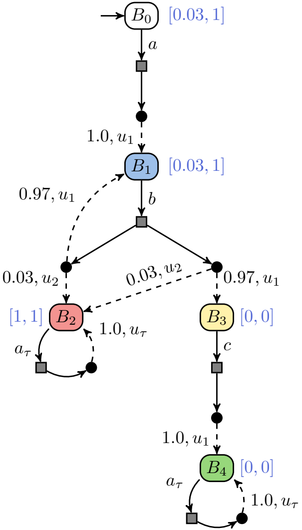
Split *pivot blocks*, which introduce imprecision

Observations

- deviation alone does not indicate a block being pivot

$\Rightarrow$ differing player 2 strategies do

Refinement



Idea

Split *pivot blocks*, which introduce imprecision

Observations

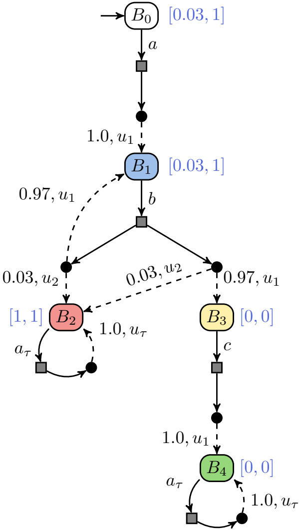
- deviation alone does not indicate a block being pivot

⇒ differing player 2 strategies do

Refinement Predicates

- derived from update leading to different blocks

Refinement



Idea

Split *pivot blocks*, which introduce imprecision

Observations

- deviation alone does not indicate a block being pivot

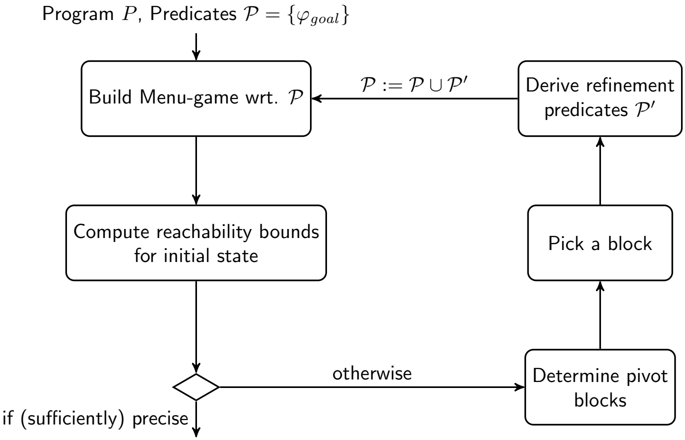
⇒ differing player 2 strategies do

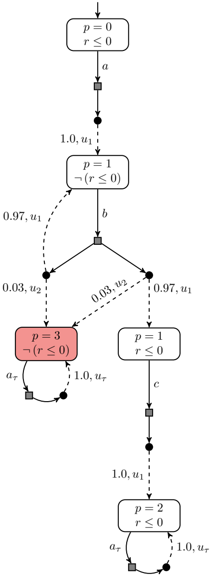
Refinement Predicates

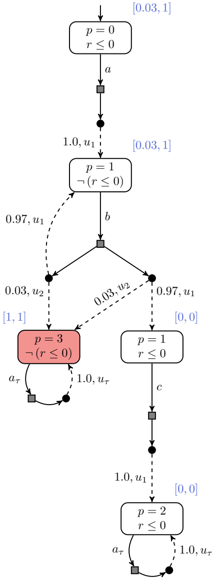
- derived from update leading to different blocks

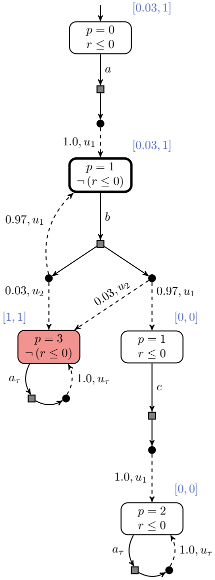
⇒ splitting corresponding behaviours

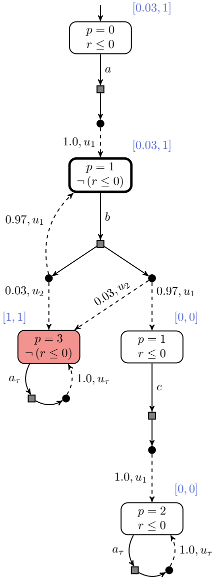
Reminder

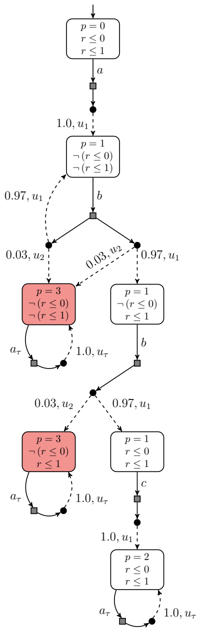

Program P, Predicates $\mathcal{P} = \{\varphi_{goal}\}$
Build Menu-game wrt. $\mathcal{P}$
$\mathcal{P} := \mathcal{P} \cup \mathcal{P}'$
Derive refinement predicates $\mathcal{P}'$
Compute reachability bounds for initial state
Pick a block
otherwise
Determine pivot blocks
if (sufficiently) precise

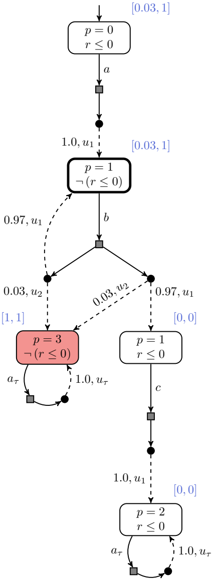

$p = 0$
$r \leq 0$
a
$1.0, u_1$
$p = 1$
$\neg(r \leq 0)$
$0.97, u_1$
b
$0.03, u_2$
$0.03, u_2$
$0.97, u_1$
$p = 3$
$\neg(r \leq 0)$
a_τ
$1.0, u_\tau$
$p = 1$
$r \leq 0$
c
$1.0, u_1$
$p = 2$
$r \leq 0$
a_τ
$1.0, u_\tau$

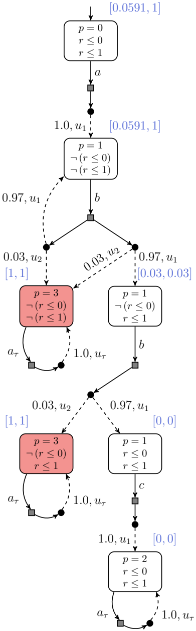

[0.03, 1]
p = 0
r ≤ 0
a
1.0, u1
[0.03, 1]
p = 1
¬(r ≤ 0)
0.97, u1
b
0.03, u2
0.03, u2
0.97, u1
[1, 1]
[0, 0]
p = 3
¬(r ≤ 0)
p = 1
r ≤ 0
aτ
1.0, uτ
c
1.0, u1
[0, 0]
p = 2
r ≤ 0
aτ
1.0, uτ

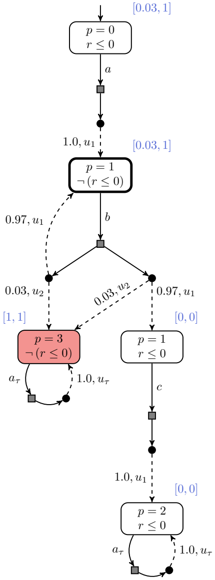

[0.03, 1]
p = 0
r ≤ 0
a
1.0, u1
[0.03, 1]
p = 1
¬(r ≤ 0)
0.97, u1
b
0.03, u2
0.03, u2
0.97, u1
[1, 1]
[0, 0]
p = 3
¬(r ≤ 0)
p = 1
r ≤ 0
aτ
1.0, uτ
c
1.0, u1
[0, 0]
p = 2
r ≤ 0
aτ
1.0, uτ


[0.03, 1]
$p = 0$
$r \leq 0$
a
$1.0, u_1$
[0.03, 1]
$p = 1$
$\neg(r \leq 0)$
b
$0.97, u_1$
$0.03, u_2$
$0.03, u_2$
$0.97, u_1$
[1, 1]
[0, 0]
$p = 3$
$\neg(r \leq 0)$
$p = 1$
$r \leq 0$
a_τ
$1.0, u_\tau$
c
$1.0, u_1$
[0, 0]
$p = 2$
$r \leq 0$
a_τ
$1.0, u_\tau$

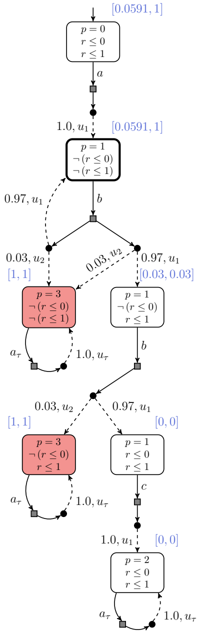

$p = 0$
$r \leq 0$
$r \leq 1$
a
$1.0, u_1$
$p = 1$
$\neg(r \leq 0)$
$\neg(r \leq 1)$
$0.97, u_1$
b
$0.03, u_2$
$0.03, u_2$
$0.97, u_1$
$p = 3$
$\neg(r \leq 0)$
$\neg(r \leq 1)$
$p = 1$
$\neg(r \leq 0)$
$r \leq 1$
a_τ
$1.0, u_\tau$
b
$0.03, u_2$
$0.97, u_1$
$p = 3$
$\neg(r \leq 0)$
$r \leq 1$
$p = 1$
$r \leq 0$
$r \leq 1$
a_τ
$1.0, u_\tau$
c
$1.0, u_1$
$p = 2$
$r \leq 0$
$r \leq 1$
a_τ
$1.0, u_\tau$


[0.03, 1]
p = 0
r ≤ 0
a
1.0, u1
[0.03, 1]
p = 1
¬(r ≤ 0)
b
0.97, u1
0.03, u2
0.03, u2
0.97, u1
[1, 1]
[0, 0]
p = 3
¬(r ≤ 0)
p = 1
r ≤ 0
aτ
1.0, uτ
c
1.0, u1
[0, 0]
p = 2
r ≤ 0
aτ
1.0, uτ

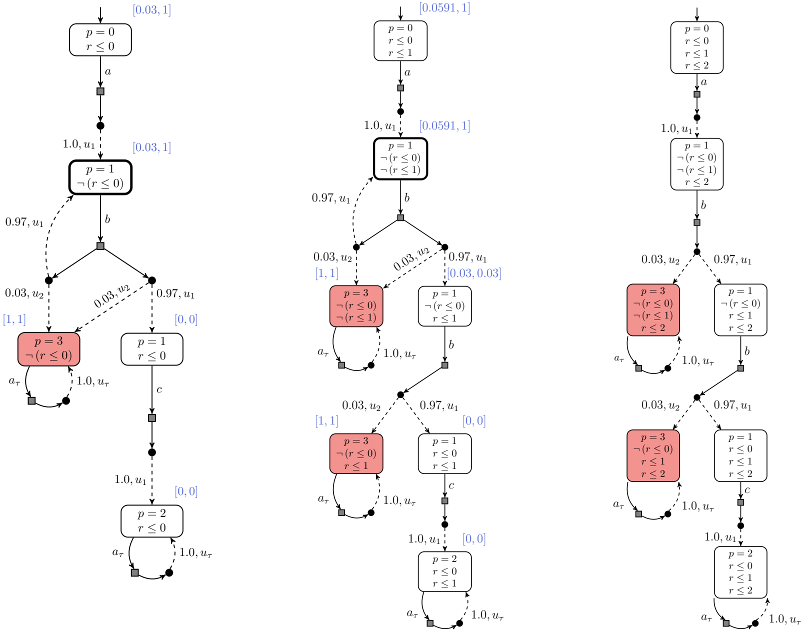

[0.0591, 1]
p = 0
r ≤ 0
r ≤ 1
a
1.0, u1
[0.0591, 1]
p = 1
¬(r ≤ 0)
¬(r ≤ 1)
0.97, u1
b
0.03, u2
0.03, u2
0.97, u1
[1, 1]
[0.03, 0.03]
p = 3
¬(r ≤ 0)
¬(r ≤ 1)
p = 1
¬(r ≤ 0)
r ≤ 1
aτ
1.0, uτ
b
0.03, u2
0.97, u1
[1, 1]
[0, 0]
p = 3
¬(r ≤ 0)
r ≤ 1
p = 1
r ≤ 0
r ≤ 1
aτ
1.0, uτ
c
1.0, u1
[0, 0]
p = 2
r ≤ 0
r ≤ 1
aτ
1.0, uτ

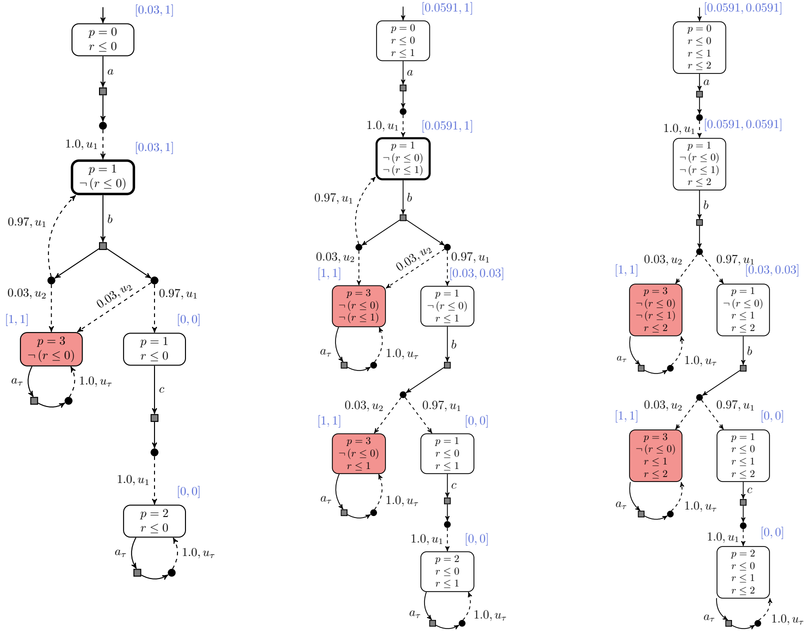


[0.03, 1]
p = 0
r ≤ 0
a
1.0, u1
[0.03, 1]
p = 1
¬(r ≤ 0)
0.97, u1
b
0.03, u2
0.03, u2
0.97, u1
[1, 1]
[0, 0]
p = 3
¬(r ≤ 0)
p = 1
r ≤ 0
aτ
1.0, uτ
c
1.0, u1
[0, 0]
p = 2
r ≤ 0
aτ
1.0, uτ

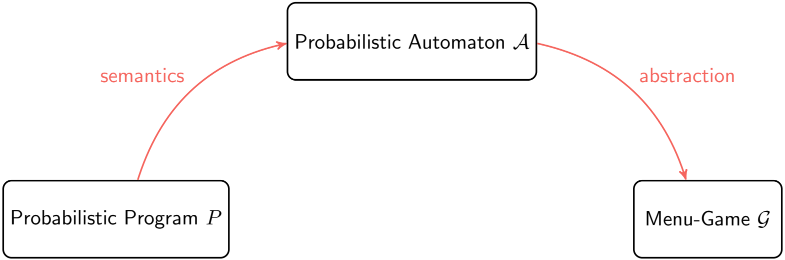


[0.0591, 1]
p = 0
r ≤ 0
r ≤ 1
a
1.0, u1
[0.0591, 1]
p = 1
¬(r ≤ 0)
¬(r ≤ 1)
0.97, u1
b
0.03, u2
0.03, u2
0.97, u1
[1, 1]
[0.03, 0.03]
p = 3
¬(r ≤ 0)
¬(r ≤ 1)
p = 1
¬(r ≤ 0)
r ≤ 1
aτ
1.0, uτ
b
0.03, u2
0.97, u1
[1, 1]
[0, 0]
p = 3
¬(r ≤ 0)
r ≤ 1
p = 1
r ≤ 0
r ≤ 1
aτ
1.0, uτ
c
1.0, u1
[0, 0]
p = 2
r ≤ 0
r ≤ 1
aτ
1.0, uτ

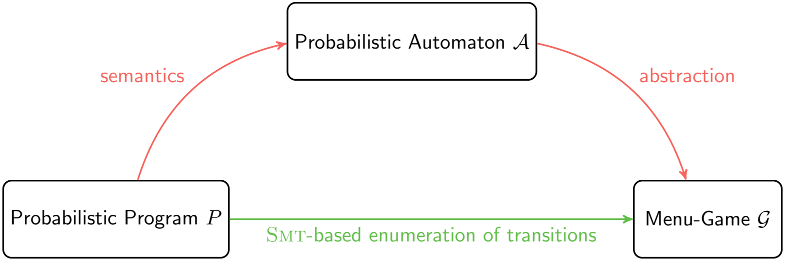

[0.03, 1]
$p = 0$
$r \leq 0$
a
$1.0, u_1$
[0.03, 1]
$p = 1$
$\neg(r \leq 0)$
$0.97, u_1$
b
$0.03, u_2$
$0.03, u_2$
$0.97, u_1$
[1, 1]
[0, 0]
$p = 3$
$\neg(r \leq 0)$
$p = 1$
$r \leq 0$
a_τ
$1.0, u_\tau$
c
$1.0, u_1$
[0, 0]
$p = 2$
$r \leq 0$
a_τ
$1.0, u_\tau$
[0.0591, 1]
$p = 0$
$r \leq 0$
$r \leq 1$
a
$1.0, u_1$
[0.0591, 1]
$p = 1$
$\neg(r \leq 0)$
$\neg(r \leq 1)$
$0.97, u_1$
b
$0.03, u_2$
$0.03, u_2$
$0.97, u_1$
[1, 1]
[0.03, 0.03]
$p = 3$
$\neg(r \leq 0)$
$\neg(r \leq 1)$
$p = 1$
$\neg(r \leq 0)$
$r \leq 1$
a_τ
$1.0, u_\tau$
b
$0.03, u_2$
$0.97, u_1$
[1, 1]
[0, 0]
$p = 3$
$\neg(r \leq 0)$
$r \leq 1$
$p = 1$
$r \leq 0$
$r \leq 1$
a_τ
$1.0, u_\tau$
c
$1.0, u_1$
[0, 0]
$p = 2$
$r \leq 0$
$r \leq 1$
a_τ
$1.0, u_\tau$
$p = 0$
$r \leq 0$
$r \leq 1$
$r \leq 2$
a
$1.0, u_1$
$p = 1$
$\neg(r \leq 0)$
$\neg(r \leq 1)$
$r \leq 2$
b
$0.03, u_2$
$0.97, u_1$
$p = 3$
$\neg(r \leq 0)$
$\neg(r \leq 1)$
$r \leq 2$
$p = 1$
$\neg(r \leq 0)$
$r \leq 1$
$r \leq 2$
a_τ
$1.0, u_\tau$
b
$0.03, u_2$
$0.97, u_1$
$p = 3$
$\neg(r \leq 0)$
$r \leq 1$
$r \leq 2$
$p = 1$
$r \leq 0$
$r \leq 1$
$r \leq 2$
a_τ
$1.0, u_\tau$
c
$1.0, u_1$
$p = 2$
$r \leq 0$
$r \leq 1$
$r \leq 2$
a_τ
$1.0, u_\tau$


[0.03, 1]
$p = 0$, $r \leq 0$
a
$1.0, u_1$
[0.03, 1]
$p = 1$, $\neg(r \leq 0)$
b
$0.97, u_1$
$0.03, u_2$
$0.03, u_2$
$0.97, u_1$
[1, 1]
[0, 0]
$p = 3$, $\neg(r \leq 0)$
$p = 1$, $r \leq 0$
a_τ
$1.0, u_\tau$
c
$1.0, u_1$
[0, 0]
$p = 2$, $r \leq 0$
a_τ
$1.0, u_\tau$
[0.0591, 1]
$p = 0$, $r \leq 0$, $r \leq 1$
a
$1.0, u_1$
[0.0591, 1]
$p = 1$, $\neg(r \leq 0)$, $\neg(r \leq 1)$
$0.97, u_1$
b
$0.03, u_2$
$0.03, u_2$
$0.97, u_1$
[1, 1]
[0.03, 0.03]
$p = 3$, $\neg(r \leq 0)$, $\neg(r \leq 1)$
$p = 1$, $\neg(r \leq 0)$, $r \leq 1$
a_τ
$1.0, u_\tau$
b
$0.03, u_2$
$0.97, u_1$
[1, 1]
[0, 0]
$p = 3$, $\neg(r \leq 0)$, $r \leq 1$
$p = 1$, $r \leq 0$, $r \leq 1$
a_τ
$1.0, u_\tau$
c
$1.0, u_1$
[0, 0]
$p = 2$, $r \leq 0$, $r \leq 1$
a_τ
$1.0, u_\tau$
[0.0591, 0.0591]
$p = 0$, $r \leq 0$, $r \leq 1$, $r \leq 2$
a
$1.0, u_1$
[0.0591, 0.0591]
$p = 1$, $\neg(r \leq 0)$, $\neg(r \leq 1)$, $r \leq 2$
b
$0.03, u_2$
$0.97, u_1$
[1, 1]
[0.03, 0.03]
$p = 3$, $\neg(r \leq 0)$, $\neg(r \leq 1)$, $r \leq 2$
$p = 1$, $\neg(r \leq 0)$, $r \leq 1$, $r \leq 2$
a_τ
$1.0, u_\tau$
b
$0.03, u_2$
$0.97, u_1$
[1, 1]
[0, 0]
$p = 3$, $\neg(r \leq 0)$, $r \leq 1$, $r \leq 2$
$p = 1$, $r \leq 0$, $r \leq 1$, $r \leq 2$
a_τ
$1.0, u_\tau$
c
$1.0, u_1$
[0, 0]
$p = 2$, $r \leq 0$, $r \leq 1$, $r \leq 2$
a_τ
$1.0, u_\tau$

Motivating SMT-based Construction



Motivating SMT-based Construction



SMT-based Construction (Example)

Consider

$$[a]\ x > 0 \rightarrow 0.7 : (x' = x + 1) + 0.3 : (y' = x)$$

$$\mathcal{P} = \{x \text{ is odd}, y \text{ is odd}\}$$

SMT-based Construction (Example)

Consider

$$[a]\ x > 0 \rightarrow 0.7 : (x' = x + 1) + 0.3 : (y' = x)$$

$$\mathcal{P} = \{x \text{ is odd}, y \text{ is odd}\}$$

Abstract Transition Constraint

$x > 0$

SMT-based Construction (Example)

Consider

$$[a]\ x > 0 \rightarrow 0.7 : (x' = x + 1) + 0.3 : (y' = x)$$

$$\mathcal{P} = \{x \text{ is odd}, y \text{ is odd}\}$$

Abstract Transition Constraint

$$x > 0$$
$$\wedge\ (b_0^{src} \Leftrightarrow x \text{ is odd}) \wedge (b_1^{src} \Leftrightarrow y \text{ is odd})$$

SMT-based Construction (Example)

Consider

$$[a]\ x > 0 \rightarrow 0.7 : (x' = x + 1) + 0.3 : (y' = x)$$

$$\mathcal{P} = \{x \text{ is odd}, y \text{ is odd}\}$$

Abstract Transition Constraint

$$\begin{aligned} & x > 0 \\ \wedge\ & (b_0^{src} \Leftrightarrow x \text{ is odd}) \wedge (b_1^{src} \Leftrightarrow y \text{ is odd}) \\ \wedge\ & (b_0^{dst_1} \Leftrightarrow x + 1 \text{ is odd}) \wedge (b_1^{dst_1} \Leftrightarrow y \text{ is odd}) \end{aligned}$$

SMT-based Construction (Example)

Consider

$$[a]\ x > 0 \rightarrow 0.7 : (x' = x + 1) + 0.3 : (y' = x)$$

$$\mathcal{P} = \{x \text{ is odd}, y \text{ is odd}\}$$

Abstract Transition Constraint

$$\begin{aligned} & x > 0 \\ \wedge\ & (b_0^{src} \Leftrightarrow x \text{ is odd}) \wedge (b_1^{src} \Leftrightarrow y \text{ is odd}) \\ \wedge\ & (b_0^{dst_1} \Leftrightarrow x + 1 \text{ is odd}) \wedge (b_1^{dst_1} \Leftrightarrow y \text{ is odd}) \\ \wedge\ & (b_0^{dst_2} \Leftrightarrow x \text{ is odd}) \wedge (b_1^{dst_2} \Leftrightarrow x \text{ is odd}) \end{aligned}$$

SMT-based Construction (Example)

Consider

$$[a]\ x > 0 \rightarrow 0.7 : (x' = x + 1) + 0.3 : (y' = x)$$

$$\mathcal{P} = \{x \text{ is odd}, y \text{ is odd}\}$$

Abstract Transition Constraint

$$\begin{aligned} & x > 0 \\ \wedge\ & (b_0^{src} \Leftrightarrow x \text{ is odd}) \wedge (b_1^{src} \Leftrightarrow y \text{ is odd}) \\ \wedge\ & (b_0^{dst_1} \Leftrightarrow x + 1 \text{ is odd}) \wedge (b_1^{dst_1} \Leftrightarrow y \text{ is odd}) \\ \wedge\ & (b_0^{dst_2} \Leftrightarrow x \text{ is odd}) \wedge (b_1^{dst_2} \Leftrightarrow x \text{ is odd}) \end{aligned}$$

Interpretation

- $(b_0^{src}, b_1^{src}) = (1, 0)$
- $(b_0^{dst_1}, b_1^{dst_1}) = (0, 0)$
- $(b_0^{dst_2}, b_1^{dst_2}) = (1, 1)$

SMT-based Construction (Example)

Consider

$$[a]\ x > 0 \rightarrow 0.7 : (x' = x + 1) + 0.3 : (y' = x)$$

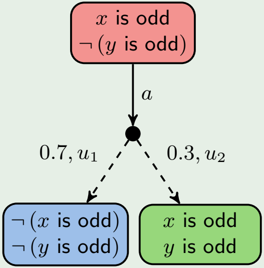
$$\mathcal{P} = \{x \text{ is odd}, y \text{ is odd}\}$$

Abstract Transition Constraint

$$x > 0$$
$$\wedge\ (b_0^{src} \Leftrightarrow x \text{ is odd}) \wedge (b_1^{src} \Leftrightarrow y \text{ is odd})$$
$$\wedge\ (b_0^{dst_1} \Leftrightarrow x + 1 \text{ is odd}) \wedge (b_1^{dst_1} \Leftrightarrow y \text{ is odd})$$
$$\wedge\ (b_0^{dst_2} \Leftrightarrow x \text{ is odd}) \wedge (b_1^{dst_2} \Leftrightarrow x \text{ is odd})$$

Interpretation

- $(b_0^{src}, b_1^{src}) = (1, 0)$
- $(b_0^{dst_1}, b_1^{dst_1}) = (0, 0)$
- $(b_0^{dst_2}, b_1^{dst_2}) = (1, 1)$



Relevant Predicates

Observation

Commands are often only related to a subset of all predicates

Relevant Predicates

Observation

Commands are often only related to a subset of all predicates

$\mathcal{P}_{u_j}^{src}$ indicate the (in)validity of predicates in the successor B_j

Relevant Predicates

Observation

Commands are often only related to a subset of all predicates

$\mathcal{P}^{src}_{u_j}$ indicate the (in)validity of predicates in the successor B_j, e.g. share variable with assignment

Relevant Predicates

Observation

Commands are often only related to a subset of all predicates

$\mathcal{P}_{u_j}^{src}$ indicate the (in)validity of predicates in the successor B_j, e.g. share variable with assignment

$\mathcal{P}_{u_j}^{dst}$ whose validity in successor blocks may be affected by u_j

Relevant Predicates

Observation

Commands are often only related to a subset of all predicates

- $\mathcal{P}_{u_j}^{src}$ indicate the (in)validity of predicates in the successor B_j, e.g. share variable with assignment
- $\mathcal{P}_{u_j}^{dst}$ whose validity in successor blocks may be affected by u_j, e.g. contain assignment variable.

Relevant Predicates

Observation

Commands are often only related to a subset of all predicates

$\mathcal{P}_{u_j}^{src}$ indicate the (in)validity of predicates in the successor B_j, e.g. share variable with assignment

$\mathcal{P}_{u_j}^{dst}$ whose validity in successor blocks may be affected by u_j, e.g. contain assignment variable.

$\Rightarrow$ irrelevant destination predicates retain their value

Relevant Predicates

Observation

Commands are often only related to a subset of all predicates

$\mathcal{P}_{u_j}^{src}$ indicate the (in)validity of predicates in the successor B_j, e.g. share variable with assignment

$\mathcal{P}_{u_j}^{dst}$ whose validity in successor blocks may be affected by u_j, e.g. contain assignment variable.

$\Rightarrow$ irrelevant destination predicates retain their value

Simplify Transition Constraint

$$[a]\ x > 0 \rightarrow 0.7 : (x' = x + 1) + 0.3 : (y' = x)$$

$$\begin{aligned} x > 0 &\wedge (b_0^{src} \Leftrightarrow x \text{ is odd}) \wedge (b_1^{src} \Leftrightarrow y \text{ is odd}) \\ &\wedge (b_0^{dst_1} \Leftrightarrow x+1 \text{ is odd}) \wedge (b_1^{dst_1} \Leftrightarrow y \text{ is odd}) \\ &\wedge (b_0^{dst_2} \Leftrightarrow x \text{ is odd}) \wedge (b_1^{dst_2} \Leftrightarrow x \text{ is odd}) \end{aligned}$$

Relevant Predicates

Observation

Commands are often only related to a subset of all predicates

$\mathcal{P}^{src}_{u_j}$ indicate the (in)validity of predicates in the successor B_j, e.g. share variable with assignment

$\mathcal{P}^{dst}_{u_j}$ whose validity in successor blocks may be affected by u_j, e.g. contain assignment variable.

$\Rightarrow$ irrelevant destination predicates retain their value

Simplify Transition Constraint

$$[a]\ x > 0 \rightarrow 0.7 : (x' = x + 1) + 0.3 : (y' = x)$$

$$x > 0 \wedge (b_0^{src} \Leftrightarrow x \text{ is odd}) \wedge \cancel{(b_1^{src} \Leftrightarrow y \text{ is odd})}$$
$$\wedge\ (b_0^{dst_1} \Leftrightarrow x + 1 \text{ is odd}) \wedge (b_1^{dst_1} \Leftrightarrow y \text{ is odd})$$
$$\wedge\ (b_0^{dst_2} \Leftrightarrow x \text{ is odd}) \wedge (b_1^{dst_2} \Leftrightarrow x \text{ is odd})$$

Relevant Predicates

Observation

Commands are often only related to a subset of all predicates

- $\mathcal{P}_{u_j}^{src}$ indicate the (in)validity of predicates in the successor B_j, e.g. share variable with assignment
- $\mathcal{P}_{u_j}^{dst}$ whose validity in successor blocks may be affected by u_j, e.g. contain assignment variable.

$\Rightarrow$ irrelevant destination predicates retain their value

Simplify Transition Constraint

$$[a]\ x > 0 \rightarrow 0.7 : (x' = x + 1) + 0.3 : (y' = x)$$

$$x > 0 \wedge (b_0^{src} \Leftrightarrow x \text{ is odd}) \wedge \cancel{(b_1^{src} \Leftrightarrow y \text{ is odd})}$$
$$\wedge\ (b_0^{dst_1} \Leftrightarrow x + 1 \text{ is odd}) \wedge \cancel{(b_1^{dst_1} \Leftrightarrow y \text{ is odd})}$$
$$\wedge\ (b_0^{dst_2} \Leftrightarrow x \text{ is odd}) \wedge (b_1^{dst_2} \Leftrightarrow x \text{ is odd})$$

Relevant Predicates

Observation

Commands are often only related to a subset of all predicates

- $\mathcal{P}_{u_j}^{src}$ indicate the (in)validity of predicates in the successor B_j, e.g. share variable with assignment
- $\mathcal{P}_{u_j}^{dst}$ whose validity in successor blocks may be affected by u_j, e.g. contain assignment variable.

$\Rightarrow$ irrelevant destination predicates retain their value

Simplify Transition Constraint

$$[a]\ x > 0 \rightarrow 0.7 : (x' = x + 1) + 0.3 : (y' = x)$$

$$\begin{aligned} x > 0 \wedge (b_0^{src} \Leftrightarrow x \text{ is odd}) \wedge \cancel{(b_1^{src} \Leftrightarrow y \text{ is odd})} \\ \wedge (b_0^{dst_1} \Leftrightarrow x + 1 \text{ is odd}) \wedge \cancel{(b_1^{dst_1} \Leftrightarrow y \text{ is odd})} \\ \wedge \cancel{(b_0^{dst_2} \Leftrightarrow x \text{ is odd})} \wedge (b_1^{dst_2} \Leftrightarrow x \text{ is odd}) \end{aligned}$$

Relevant Predicates

Observation

Commands are often only related to a subset of all predicates

- $\mathcal{P}_{u_j}^{src}$ indicate the (in)validity of predicates in the successor B_j, e.g. share variable with assignment
- $\mathcal{P}_{u_j}^{dst}$ whose validity in successor blocks may be affected by u_j, e.g. contain assignment variable.

$\Rightarrow$ irrelevant destination predicates retain their value

Simplify Transition Constraint

$$[a]\ x > 0 \rightarrow 0.7 : (x' = x + 1) + 0.3 : (y' = x)$$

$x > 0 \wedge (b_0^{src} \Leftrightarrow x$ is odd$) \wedge$ ~~$(b_1^{src} \Leftrightarrow y$ is odd$)$~~

$\wedge\ (b_0^{dst_1} \Leftrightarrow x + 1$ is odd$) \wedge$ ~~$(b_1^{dst_1} \Leftrightarrow y$ is odd$)$~~

$\wedge$ ~~$(b_0^{dst_2} \Leftrightarrow x$ is odd$)$~~ $\wedge\ (b_1^{dst_2} \Leftrightarrow x$ is odd$)$

$\Rightarrow$ extend solutions with $b_1^{src} \Leftrightarrow b_1^{dst_1}$ and $b_0^{src} \Leftrightarrow b_0^{dst_2}$

Reachable Blocks as Constraint

Observation

Refinement can only split blocks but never introduce “new” ones

Reachable Blocks as Constraint

Observation

Refinement can only split blocks but never introduce “new” ones

$\Rightarrow$ new transition constraint solutions extend the old ones

Reachable Blocks as Constraint

Observation

Refinement can only split blocks but never introduce “new” ones
$\Rightarrow$ new transition constraint solutions extend the old ones

New Solutions are Extensions

Let the old solution have only three source blocks:

$$(b_0^{src}, b_1^{src}, b_2^{src}) \in \{(0,0,1), (0,1,1), (1,0,0)\}$$

Reachable Blocks as Constraint

Observation

Refinement can only split blocks but never introduce “new” ones
$\Rightarrow$ new transition constraint solutions extend the old ones

New Solutions are Extensions

Let the old solution have only three source blocks:

$$(b_0^{src}, b_1^{src}, b_2^{src}) \in \{(0,0,1), (0,1,1), (1,0,0)\}$$

Solutions $(b_0^{src}, b_1^{src}, b_2^{src}, b_3^{src})$ of refined constraint must extend those, i.e. be in

$$\{(0,0,1), (0,1,1), (1,0,0)\} \times \{0,1\}$$

Reachable Blocks as Constraint

Observation

Refinement can only split blocks but never introduce “new” ones
$\Rightarrow$ new transition constraint solutions extend the old ones

New Solutions are Extensions

Let the old solution have only three source blocks:

$$(b_0^{src}, b_1^{src}, b_2^{src}) \in \{(0,0,1), (0,1,1), (1,0,0)\}$$

Solutions $(b_0^{src}, b_1^{src}, b_2^{src}, b_3^{src})$ of refined constraint must extend those, i.e. be in

$$\{(0,0,1), (0,1,1), (1,0,0)\} \times \{0,1\}$$

Idea

Extend transition constraint with reachable blocks constraints

Other optimisations

Variables' Ranges $\Rightarrow$ extend constraint with variables' domains

Exploit Incrementality

Predicate Decomposition

Unrelated Commands

Other optimisations

Variables' Ranges $\Rightarrow$ extend constraint with variables' domains

- e.g. for $x \in \{0, 1, 2\}$ add $0 \leq x \land x \leq 2$

Exploit Incrementality

Predicate Decomposition

Unrelated Commands

Other optimisations

Variables' Ranges

$\Rightarrow$ extend constraint with variables' domains

- e.g. for $x \in \{0, 1, 2\}$ add $0 \leq x \wedge x \leq 2$

Exploit Incrementality

- incremental checking faster than starting from scratch

Predicate Decomposition

Unrelated Commands

Other optimisations

Variables' Ranges

$\Rightarrow$ extend constraint with variables' domains

- e.g. for $x \in \{0, 1, 2\}$ add $0 \leq x \wedge x \leq 2$

Exploit Incrementality

- incremental checking faster than starting from scratch
- transition constraint grows monotonously

Predicate Decomposition

Unrelated Commands

Other optimisations

Variables' Ranges

⇒ extend constraint with variables' domains

- e.g. for $x \in \{0, 1, 2\}$ add $0 \leq x \wedge x \leq 2$

Exploit Incrementality

- incremental checking faster than starting from scratch
- transition constraint grows monotonously

⇒ one SMT-solver instance for each command

Predicate Decomposition

Unrelated Commands

Other optimisations

Variables' Ranges

$\Rightarrow$ extend constraint with variables' domains

- e.g. for $x \in \{0, 1, 2\}$ add $0 \leq x \wedge x \leq 2$

Exploit Incrementality

- incremental checking faster than starting from scratch
- transition constraint grows monotonously

$\Rightarrow$ one SMT-solver instance for each command

Predicate Decomposition

$\Rightarrow$ split spuriously coupled variables

Unrelated Commands

Other optimisations

Variables' Ranges

$\Rightarrow$ extend constraint with variables' domains

- e.g. for $x \in \{0, 1, 2\}$ add $0 \leq x \wedge x \leq 2$

Exploit Incrementality

- incremental checking faster than starting from scratch
- transition constraint grows monotonously

$\Rightarrow$ one SMT-solver instance for each command

Predicate Decomposition

$\Rightarrow$ split spuriously coupled variables

- e.g. $\{x = 1 \wedge y > 0\}$ becomes $\{x = 1, y > 0\}$

Unrelated Commands

Other optimisations

Variables' Ranges

$\Rightarrow$ extend constraint with variables' domains

- e.g. for $x \in \{0, 1, 2\}$ add $0 \leq x \land x \leq 2$

Exploit Incrementality

- incremental checking faster than starting from scratch
- transition constraint grows monotonously

$\Rightarrow$ one SMT-solver instance for each command

Predicate Decomposition

$\Rightarrow$ split spuriously coupled variables

- e.g. $\{x = 1 \land y > 0\}$ becomes $\{x = 1, y > 0\}$

Unrelated Commands

- new predicate often not relevant for all commands

Other optimisations

Variables' Ranges

$\Rightarrow$ extend constraint with variables' domains

- e.g. for $x \in \{0, 1, 2\}$ add $0 \leq x \wedge x \leq 2$

Exploit Incrementality

- incremental checking faster than starting from scratch
- transition constraint grows monotonously

$\Rightarrow$ one SMT-solver instance for each command

Predicate Decomposition

$\Rightarrow$ split spuriously coupled variables

- e.g. $\{x = 1 \wedge y > 0\}$ becomes $\{x = 1, y > 0\}$

Unrelated Commands

- new predicate often not relevant for all commands

$\Rightarrow$ reuse previous solutions

Pre-computation

Observations

- value iteration may not yield reachability probability to be exactly 1

Pre-computation

Observations

- value iteration may not yield reachability probability to be exactly 1
- convergence to 0 or 1 may be slow

Pre-computation

Observations

- value iteration may not yield reachability probability to be exactly 1
- convergence to 0 or 1 may be slow

Idea

Extend pre-computation algorithms for PA to Menu-games

Pre-computation

Observations

- value iteration may not yield reachability probability to be exactly 1
- convergence to 0 or 1 may be slow

Idea

Extend pre-computation algorithms for PA to Menu-games

"σ_1"	"σ_2"	PROB0	PROB1
$-$	$-$	EE	AA
$-$	$+$		
$+$	$-$		
$+$	$+$		

Pre-computation

Observations

- value iteration may not yield reachability probability to be exactly 1
- convergence to 0 or 1 may be slow

Idea

Extend pre-computation algorithms for PA to Menu-games

"σ_1"	"σ_2"	PROB0	PROB1
$-$	$-$	EE	AA
$-$	$+$	EA	AE
$+$	$-$		
$+$	$+$		

Pre-computation

Observations

- value iteration may not yield reachability probability to be exactly 1
- convergence to 0 or 1 may be slow

Idea

Extend pre-computation algorithms for PA to Menu-games

"σ_1"	"σ_2"	Prob0	Prob1
$-$	$-$	EE	AA
$-$	$+$	EA	AE
$+$	$-$	AE	EA
$+$	$+$		

Pre-computation

Observations

- value iteration may not yield reachability probability to be exactly 1
- convergence to 0 or 1 may be slow

Idea

Extend pre-computation algorithms for PA to Menu-games

"σ_1"	"σ_2"	PROB0	PROB1
$-$	$-$	EE	AA
$-$	$+$	EA	AE
$+$	$-$	AE	EA
$+$	$+$	AA	EE

Pre-computation

Observations

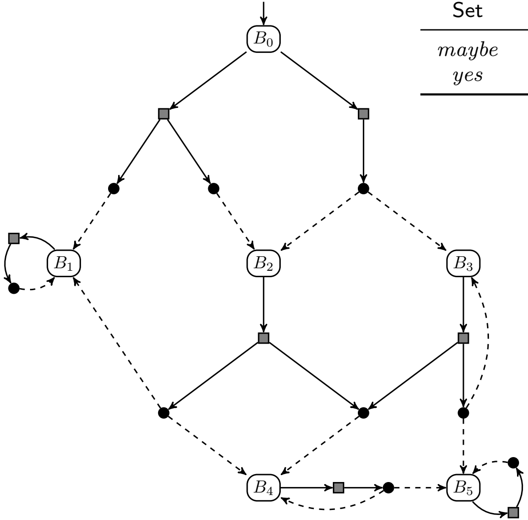
- value iteration may not yield reachability probability to be exactly 1
- convergence to 0 or 1 may be slow

Idea

Extend pre-computation algorithms for PA to Menu-games

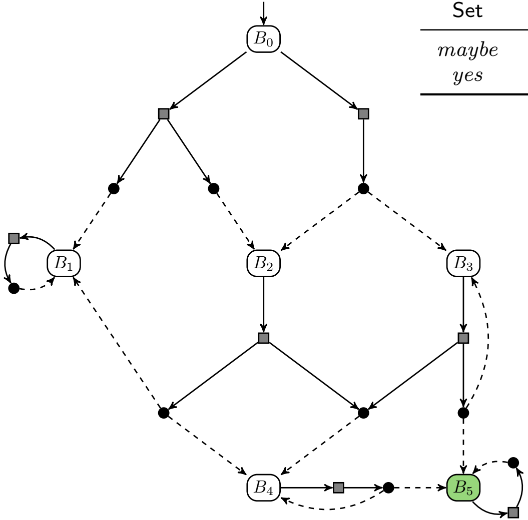
"σ_1"	"σ_2"	Prob0	Prob1
$-$	$-$	EE	AA
$-$	$+$	EA	AE
$+$	$-$	AE	EA
$+$	$+$	AA	EE

PROB1EA (Example)



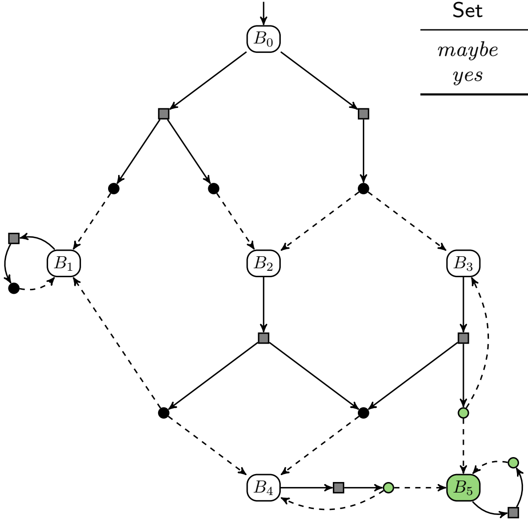
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_5

PROB1EA (Example)



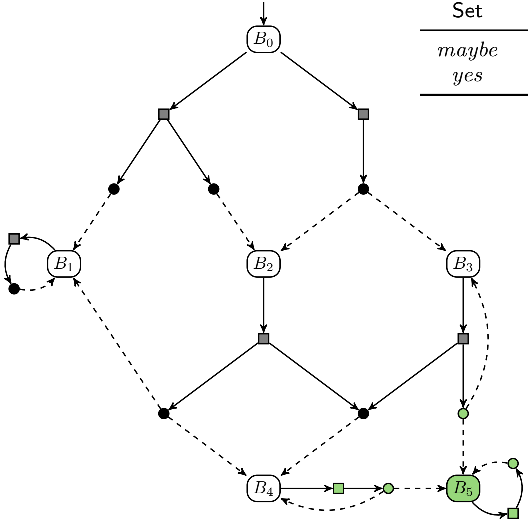
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_5

PROB1EA (Example)



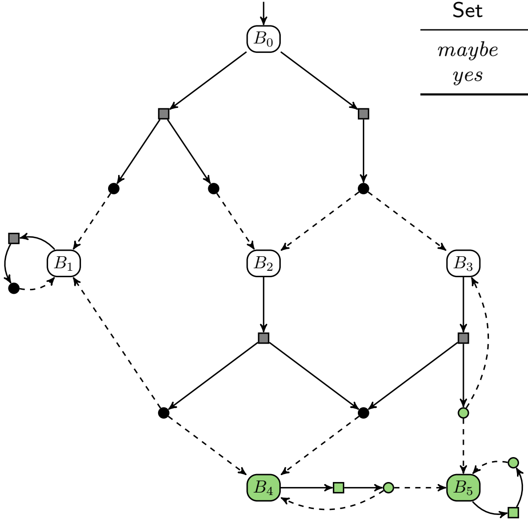
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_5

Prob1EA (Example)



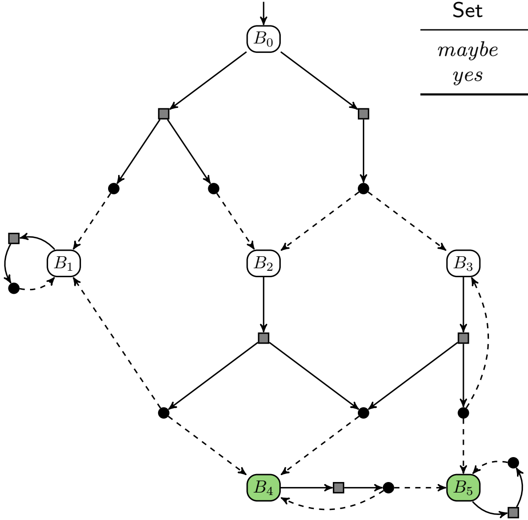
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_5

PROB1EA (Example)



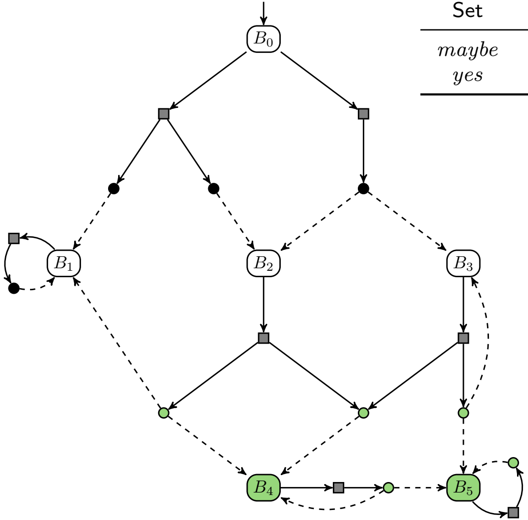
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_5

PROB1EA (Example)



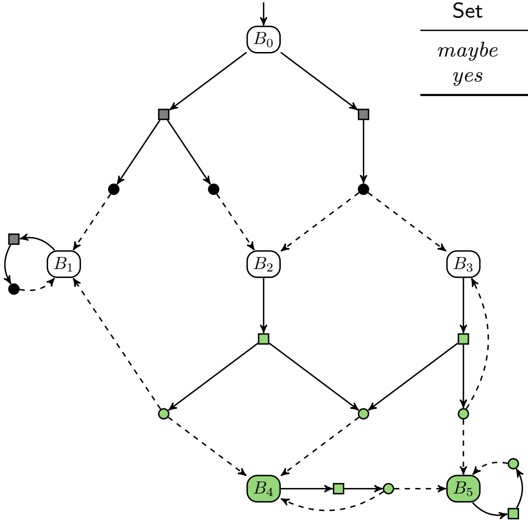
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_4, B_5

PROB1EA (Example)



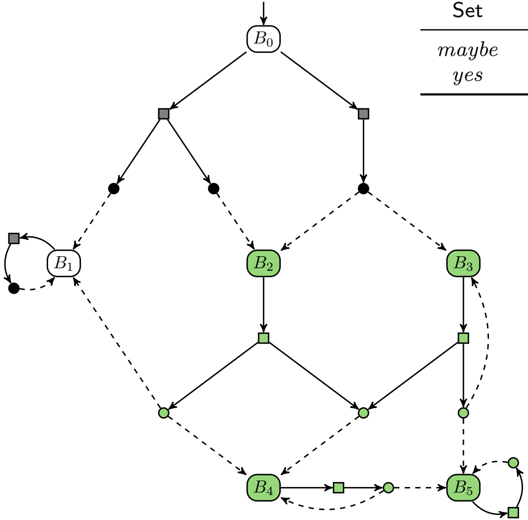
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_4, B_5

PROB1EA (Example)



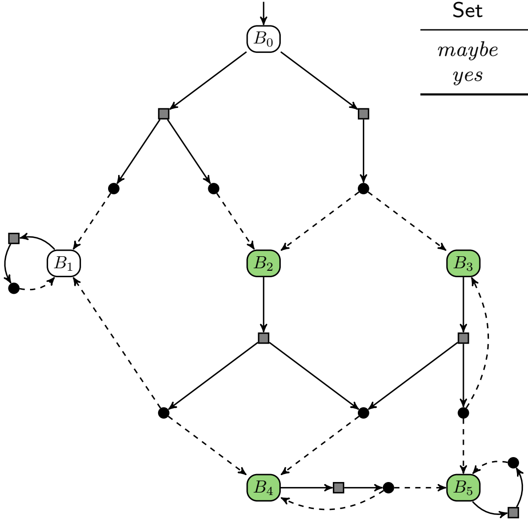
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_4, B_5

PROB1EA (Example)



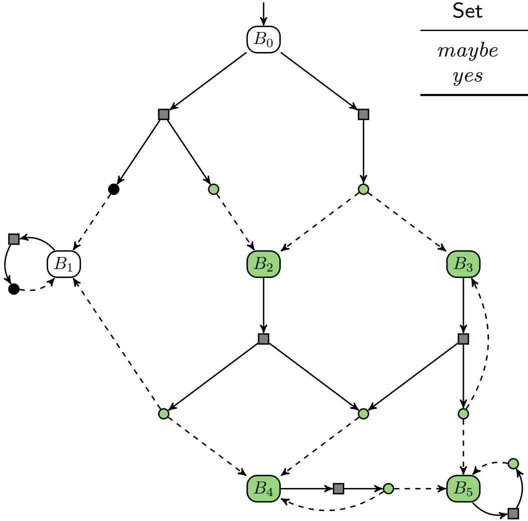
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_4, B_5

PROB1EA (Example)



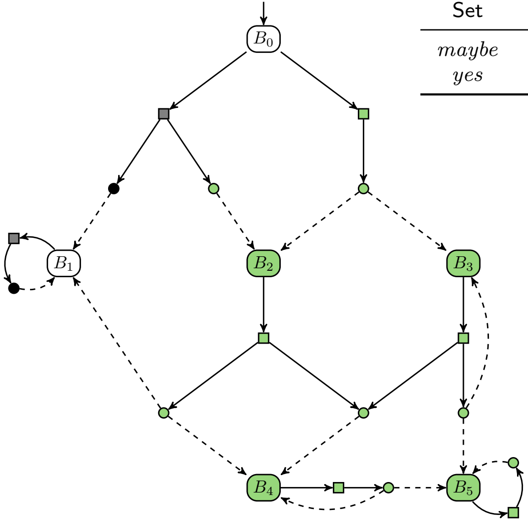
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_2, B_3, B_4, B_5

PROB1EA (Example)



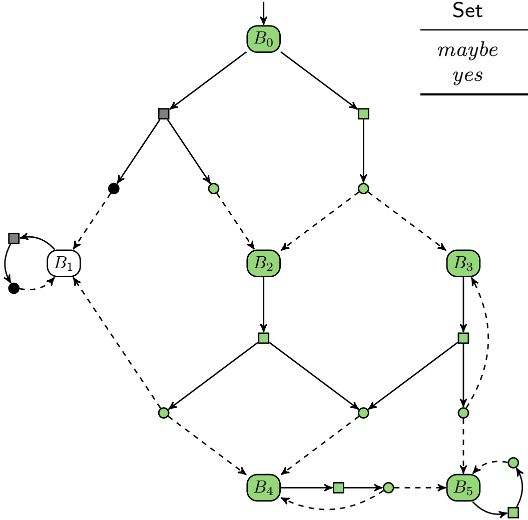
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_2, B_3, B_4, B_5

PROB1EA (Example)



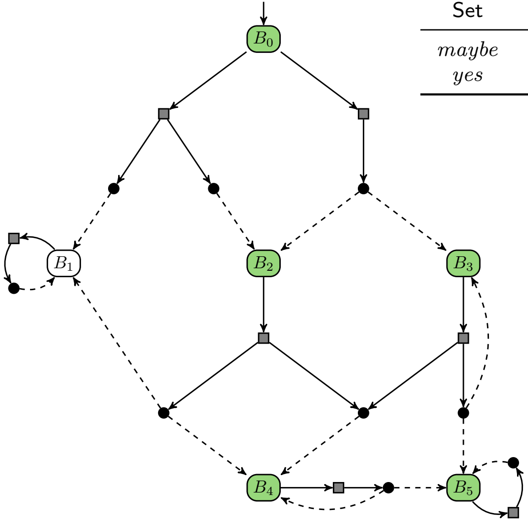
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_2, B_3, B_4, B_5

PROB1EA (Example)



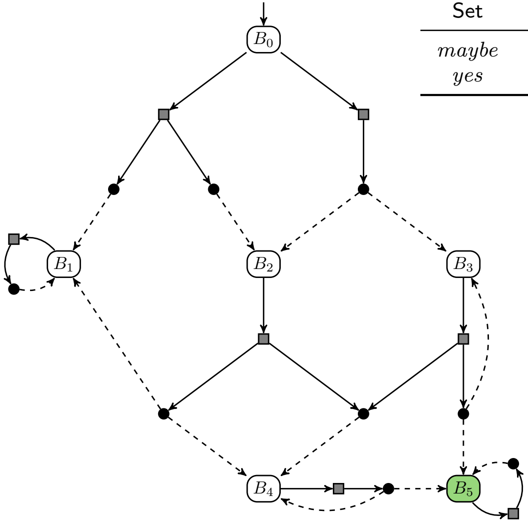
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_2, B_3, B_4, B_5

Prob1EA (Example)



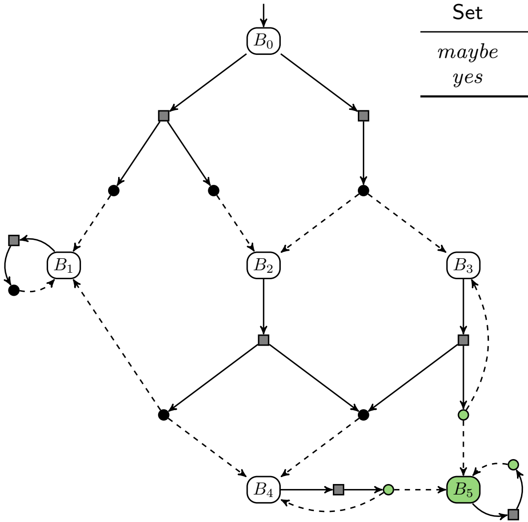
Set	Blocks
maybe	$B_0, B_1, B_2, B_3, B_4, B_5$
yes	B_0, B_2, B_3, B_4, B_5

Prob1EA (Example)



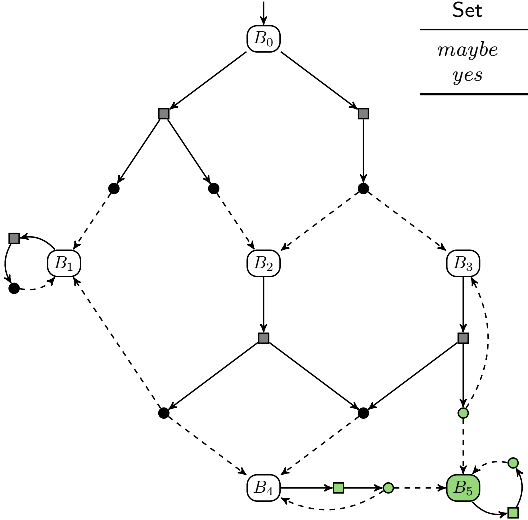
Set	Blocks
maybe	B_0, B_2, B_3, B_4, B_5
yes	B_5

PROB1EA (Example)



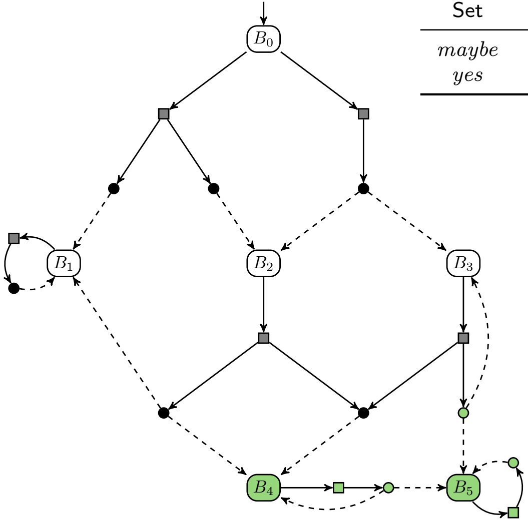
Set	Blocks
maybe	B_0, B_2, B_3, B_4, B_5
yes	B_5

PROB1EA (Example)



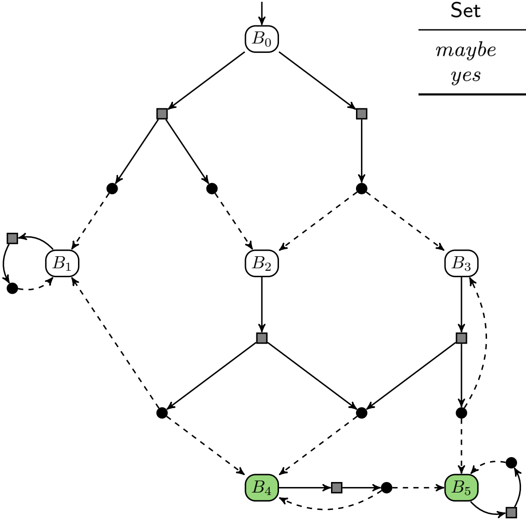
Set	Blocks
maybe	B_0, B_2, B_3, B_4, B_5
yes	B_5

PROB1EA (Example)



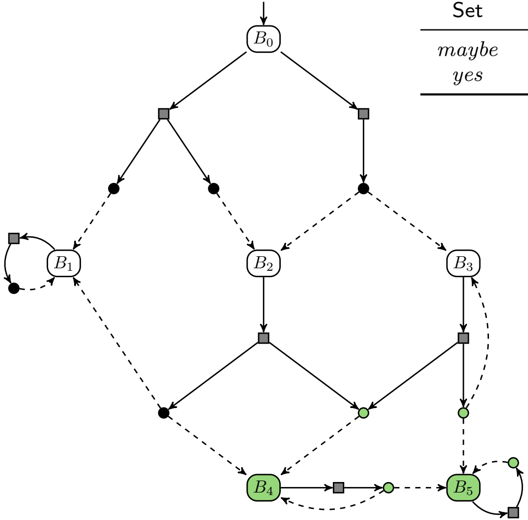
Set	Blocks
maybe	B_0, B_2, B_3, B_4, B_5
yes	B_5

PROB1EA (Example)



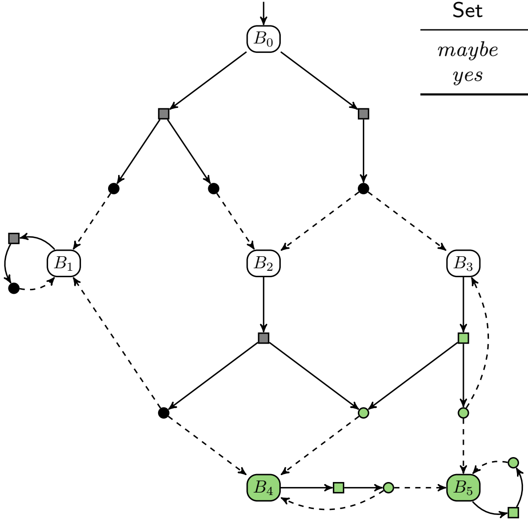
Set	Blocks
maybe	B_0, B_2, B_3, B_4, B_5
yes	B_4, B_5

PROB1EA (Example)



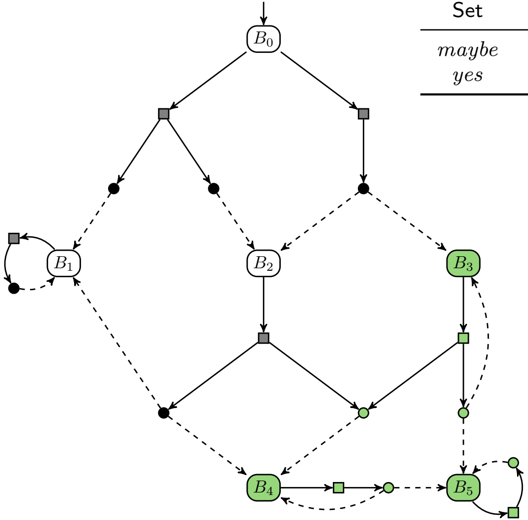
Set	Blocks
maybe	B_0, B_2, B_3, B_4, B_5
yes	B_4, B_5

PROB1EA (Example)



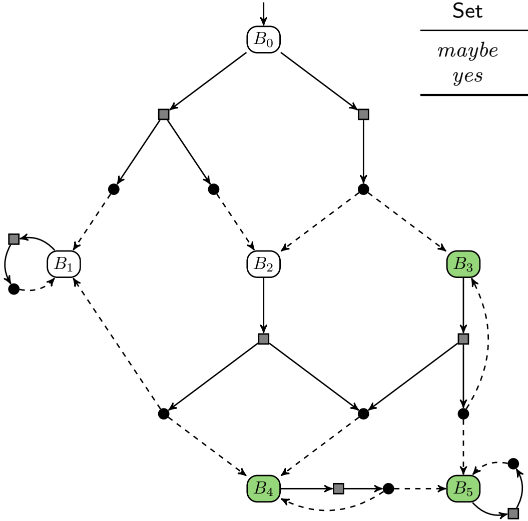
Set	Blocks
maybe	B_0, B_2, B_3, B_4, B_5
yes	B_4, B_5

PROB1EA (Example)



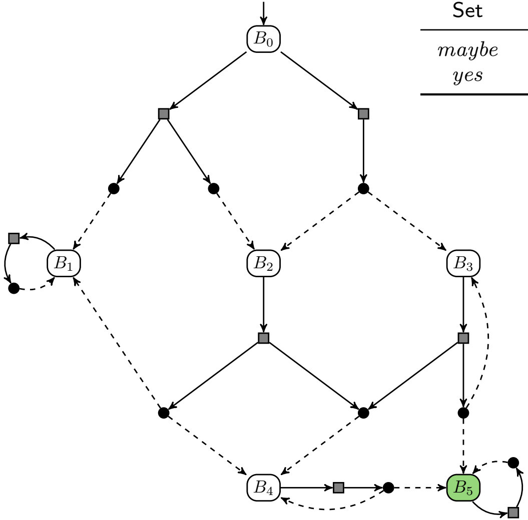
Set	Blocks
maybe	B_0, B_2, B_3, B_4, B_5
yes	B_4, B_5

PROB1EA (Example)



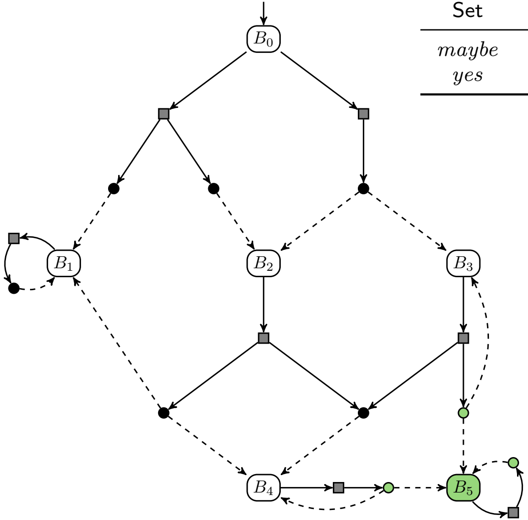
Set	Blocks
maybe	B_0, B_2, B_3, B_4, B_5
yes	B_3, B_4, B_5

PROB1EA (Example)



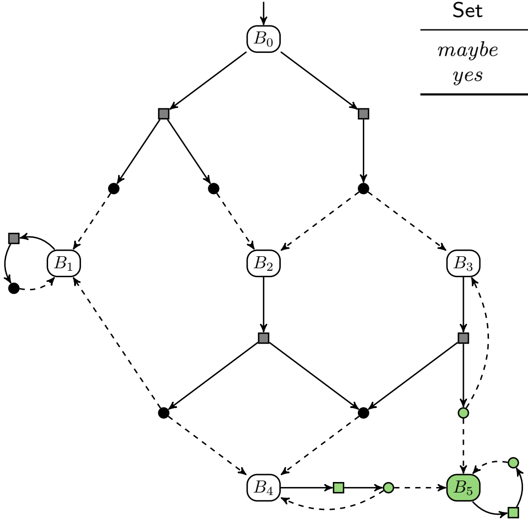
Set	Blocks
maybe	B_3, B_4, B_5
yes	B_5

PROB1EA (Example)



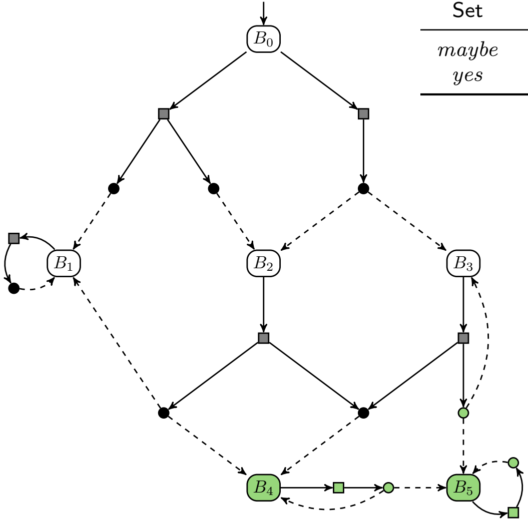
Set	Blocks
maybe	B_3, B_4, B_5
yes	B_5

PROB1EA (Example)



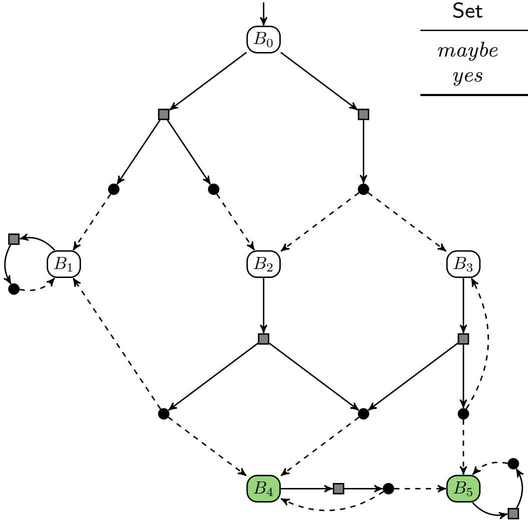
Set	Blocks
maybe	B_3, B_4, B_5
yes	B_5

PROB1EA (Example)



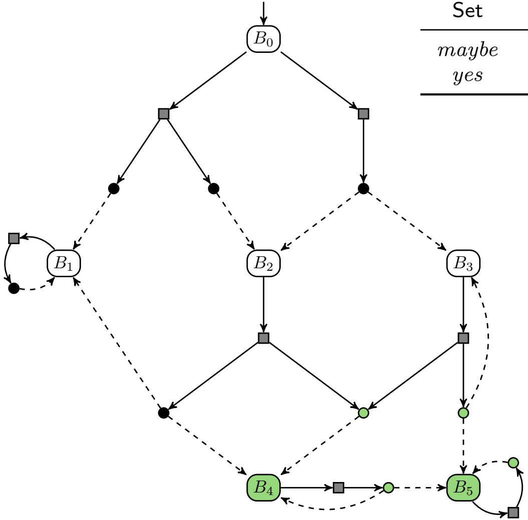
Set	Blocks
$maybe$	B_3, B_4, B_5
yes	B_5

Prob1EA (Example)



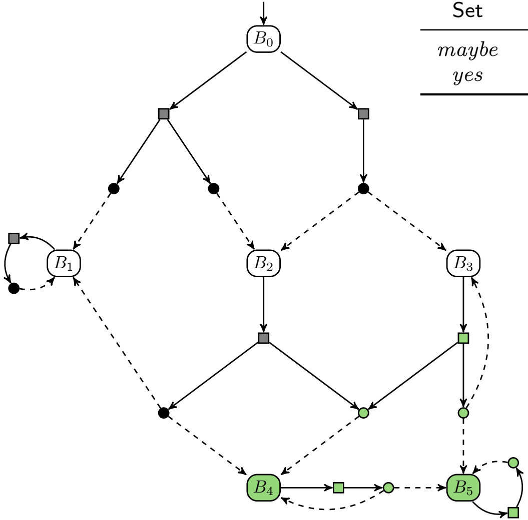
Set	Blocks
maybe	B_3, B_4, B_5
yes	B_4, B_5

PROB1EA (Example)



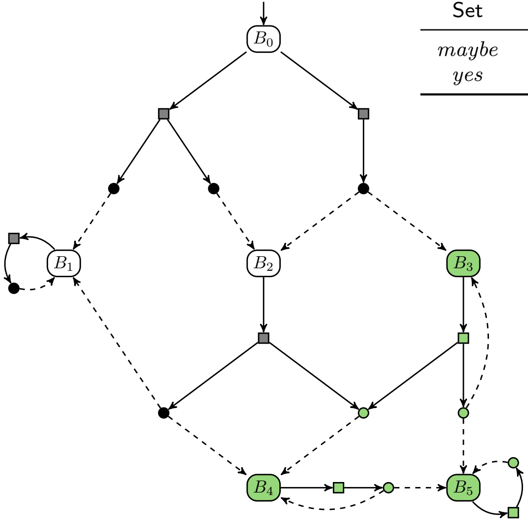
Set	Blocks
maybe	B_3, B_4, B_5
yes	B_4, B_5

PROB1EA (Example)



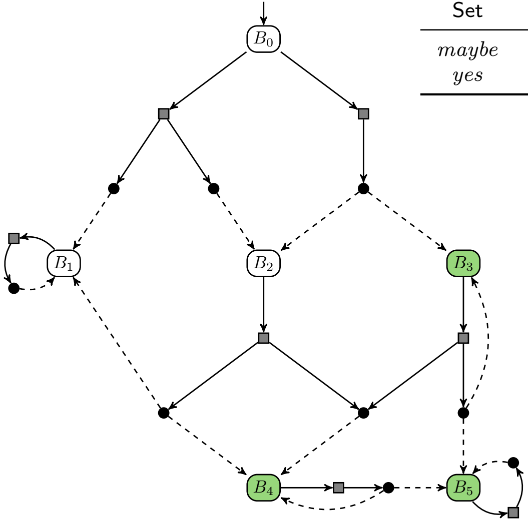
Set	Blocks
maybe	B_3, B_4, B_5
yes	B_4, B_5

Prob1EA (Example)



Set	Blocks
maybe	B_3, B_4, B_5
yes	B_4, B_5

PROB1EA (Example)



Set	Blocks
maybe	B_3, B_4, B_5
yes	B_3, B_4, B_5

Other tweaks

Reuse reachability

- avoid starting value iteration from scratch

Remove goal transitions

Other tweaks

Reuse reachability

- avoid starting value iteration from scratch

⇒ reuse previous refinement iteration results (where applicable)

Remove goal transitions

Other tweaks

Reuse reachability

- avoid starting value iteration from scratch

⇒ reuse previous refinement iteration results (where applicable)

Remove goal transitions

- focus on probabilistic reachability

Other tweaks

Reuse reachability

- avoid starting value iteration from scratch

$\Rightarrow$ reuse previous refinement iteration results (where applicable)

Remove goal transitions

- focus on probabilistic reachability
- irrelevant what happens once goal is reached

Other tweaks

Reuse reachability

- avoid starting value iteration from scratch

⇒ reuse previous refinement iteration results (where applicable)

Remove goal transitions

- focus on probabilistic reachability
- irrelevant what happens once goal is reached

⇒ remove outgoing transitions of goal blocks

Experiments

Prototypical Implementation

- uses STORM's parser, expressions and can use explicit value iteration

Experiments

Prototypical Implementation

- uses STORM's parser, expressions and can use explicit value iteration
- uses Z3 as SMT-solver and CUDD as MTBDD-library

Experiments

Prototypical Implementation

- uses STORM's parser, expressions and can use explicit value iteration
- uses Z3 as SMT-solver and CUDD as MTBDD-library
- obstacles:
  - corner cases of backward refinement not documented

Experiments

Prototypical Implementation

- uses STORM's parser, expressions and can use explicit value iteration
- uses Z3 as SMT-solver and CUDD as MTBDD-library
- obstacles:
  - corner cases of backward refinement not documented
  - vague (to not existent) description of PASS's implementation details

Experiments

Prototypical Implementation

- uses STORM's parser, expressions and can use explicit value iteration
- uses Z3 as SMT-solver and CUDD as MTBDD-library
- obstacles:
  - corner cases of backward refinement not documented
  - vague (to not existent) description of PASS's implementation details
  - standard MTBDD operations not sufficient for strategy computation

Experiments

Prototypical Implementation

- uses STORM's parser, expressions and can use explicit value iteration
- uses Z3 as SMT-solver and CUDD as MTBDD-library
- obstacles:
  - corner cases of backward refinement not documented
  - vague (to not existent) description of PASS's implementation details
  - standard MTBDD operations not sufficient for strategy computation
  - ...

Experiments

Prototypical Implementation

- uses STORM's parser, expressions and can use explicit value iteration
- uses Z3 as SMT-solver and CUDD as MTBDD-library
- obstacles:
  - corner cases of backward refinement not documented
  - vague (to not existent) description of PASS's implementation details
  - standard MTBDD operations not sufficient for strategy computation
  - ...
- final implementation has ≈ 6000 lines of code (18.000 committed)

Experiments

Prototypical Implementation

- uses STORM's parser, expressions and can use explicit value iteration
- uses Z3 as SMT-solver and CUDD as MTBDD-library
- obstacles:
  - corner cases of backward refinement not documented
  - vague (to not existent) description of PASS's implementation details
  - standard MTBDD operations not sufficient for strategy computation
  - ...
- final implementation has ≈ 6000 lines of code (18.000 committed)

Case Studies

- 4 case studies (focus on two here)

Experiments

Prototypical Implementation

- uses STORM's parser, expressions and can use explicit value iteration
- uses Z3 as SMT-solver and CUDD as MTBDD-library
- obstacles:
  - corner cases of backward refinement not documented
  - vague (to not existent) description of PASS's implementation details
  - standard MTBDD operations not sufficient for strategy computation
  - ...
- final implementation has ≈ 6000 lines of code (18.000 committed)

Case Studies

- 4 case studies (focus on two here)
- measured impact of optimisations on run time and game sizes

Experiments

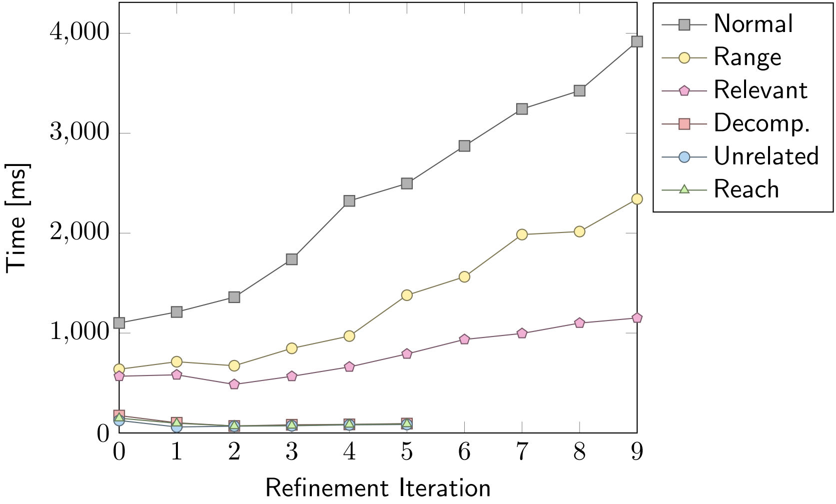
Prototypical Implementation

- uses STORM's parser, expressions and can use explicit value iteration
- uses Z3 as SMT-solver and CUDD as MTBDD-library
- obstacles:
  - corner cases of backward refinement not documented
  - vague (to not existent) description of PASS's implementation details
  - standard MTBDD operations not sufficient for strategy computation
  - ...
- final implementation has ≈ 6000 lines of code $(18.000$ committed)

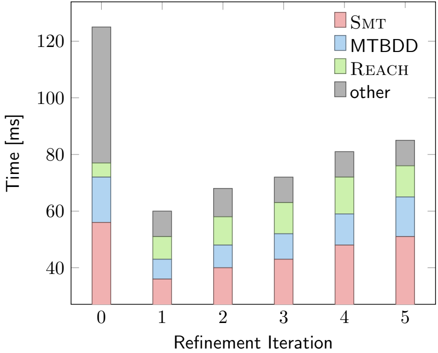
Case Studies

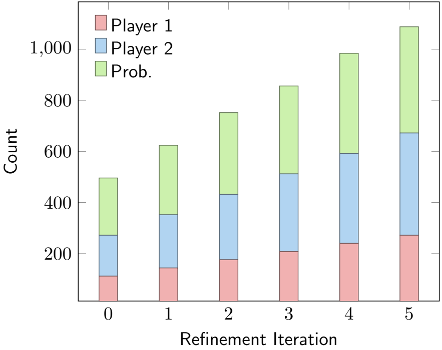
- 4 case studies (focus on two here)
- measured impact of optimisations on run time and game sizes
- evaluated symbolic vs. explicit analysis (memory usage & run time)

Consensus (Abstraction)

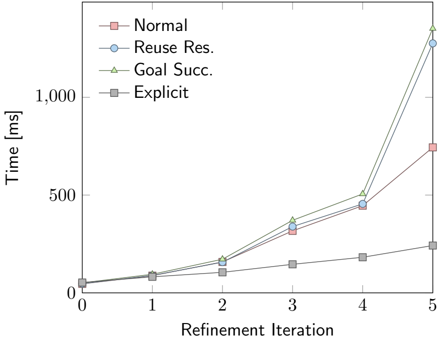


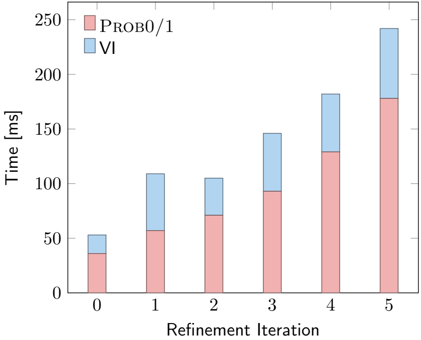
Consensus (Abstraction)



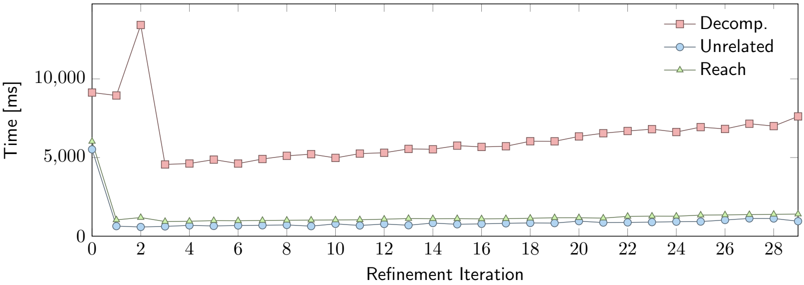


Consensus (Analysis)

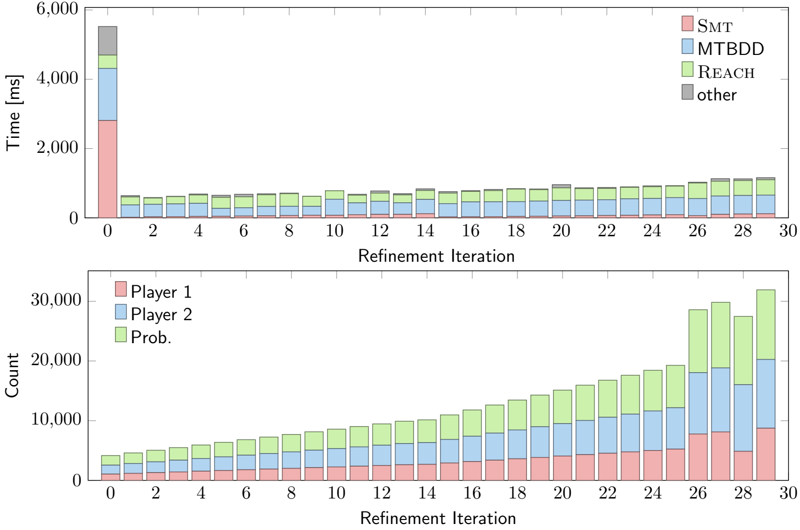




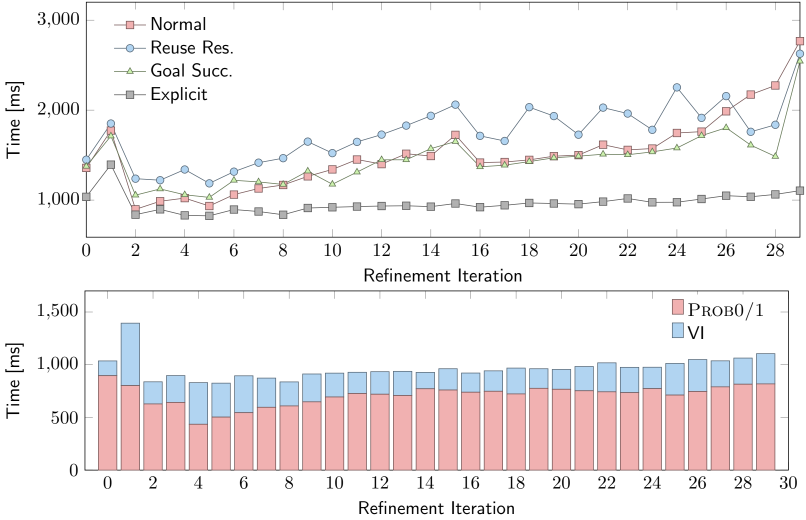
WLAN (Abstraction)



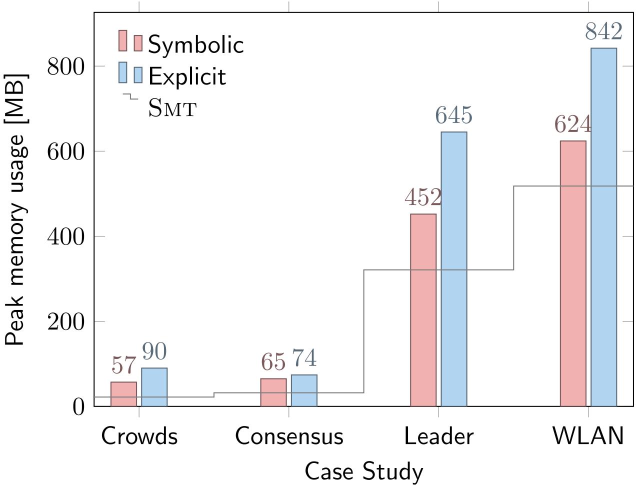
WLAN (Abstraction)



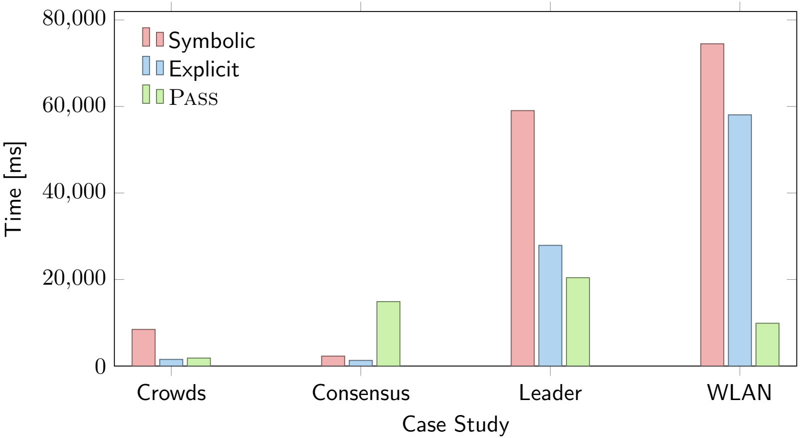
WLAN (Analysis)



Memory usage



Symbolic vs. Explicit vs. PASS



Conclusion

Summary

- Menu-game as over-approximation of a PA

Conclusion

Summary

- Menu-game as over-approximation of a PA
- several optimisations for both abstraction and analysis

Conclusion

Summary

- Menu-game as over-approximation of a PA
- several optimisations for both abstraction and analysis
- results:
  - proposed optimisations are crucial

Conclusion

Summary

- Menu-game as over-approximation of a PA
- several optimisations for both abstraction and analysis
- results:
  - proposed optimisations are crucial
  - symbolical approach slower but needs significantly less memory

Conclusion

Summary

- Menu-game as over-approximation of a PA
- several optimisations for both abstraction and analysis
- results:
  - proposed optimisations are crucial
  - symbolical approach slower but needs significantly less memory
  - MTBDD operations are the bottleneck

Conclusion

Summary

- Menu-game as over-approximation of a PA
- several optimisations for both abstraction and analysis
- results:
  - proposed optimisations are crucial
  - symbolical approach slower but needs significantly less memory
  - MTBDD operations are the bottleneck
  - comparable to PASS

Conclusion

Summary

- Menu-game as over-approximation of a PA
- several optimisations for both abstraction and analysis
- results:
  - proposed optimisations are crucial
  - symbolical approach slower but needs significantly less memory
  - MTBDD operations are the bottleneck
  - comparable to PASS

Future work

- topological symbolic value iteration

Conclusion

Summary

- Menu-game as over-approximation of a PA
- several optimisations for both abstraction and analysis
- results:
  - proposed optimisations are crucial
  - symbolical approach slower but needs significantly less memory
  - MTBDD operations are the bottleneck
  - comparable to PASS

Future work

- topological symbolic value iteration
- parallelisation

Conclusion

Summary

- Menu-game as over-approximation of a PA
- several optimisations for both abstraction and analysis
- results:
  - proposed optimisations are crucial
  - symbolical approach slower but needs significantly less memory
  - MTBDD operations are the bottleneck
  - comparable to PASS

Future work

- topological symbolic value iteration
- parallelisation
- restrictive refinement predicates (remove spurious blocks)

Conclusion

Summary

- Menu-game as over-approximation of a PA
- several optimisations for both abstraction and analysis
- results:
  - proposed optimisations are crucial
  - symbolical approach slower but needs significantly less memory
  - MTBDD operations are the bottleneck
  - comparable to PASS

Future work

- topological symbolic value iteration
- parallelisation
- restrictive refinement predicates (remove spurious blocks)
- local refinement

Conclusion

Summary

- Menu-game as over-approximation of a PA
- several optimisations for both abstraction and analysis
- results:
  - proposed optimisations are crucial
  - symbolical approach slower but needs significantly less memory
  - MTBDD operations are the bottleneck
  - comparable to PASS

Future work

- topological symbolic value iteration
- parallelisation
- restrictive refinement predicates (remove spurious blocks)
- local refinement
- exploit modularity of probabilistic programs

Thanks for your attention!

Interested in details? Suggestions?