Abstraction Refinement for Probabilistic Software

Sascha Kurowski

6. Juli 2016

Outline

ANSI-C
program

Gambler's ruin

```
int main() {
1: int c = ndet(2) + 1;
2: while (0 < c && c < 3) {
3:   c += coin(0.5) ? 1:-1;
   }
4: assert(c > 0);
}
```

RWTH AACHEN UNIVERSITY

Gambler's ruin

```
int main() {
1: int c = ndet(2) + 1;
2: while (0 < c && c < 3) {
3:   c += coin(0.5) ? 1:-1;
   }
4: assert(c > 0);
}
```

Non-deterministic choice: **int** ndet(**int** n)

- No information about outcome
- Returns any value between 0 and n
- E.g. user input or underspecified function

Gambler's ruin

```
int main() {
1: int c = ndet(2) + 1;
2: while (0 < c && c < 3) {
3:   c += coin(0.5) ? 1:−1;
   }
4: assert(c > 0);
}
```

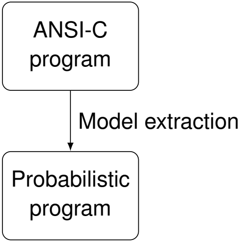
Probabilistic choices: **int** coin(**float** p)

- Likelyhood of each possible outcome known
- Returns 1 with probability p and 0 with probability $1 - p$
- E.g. randomization or network communication

Gambler's ruin

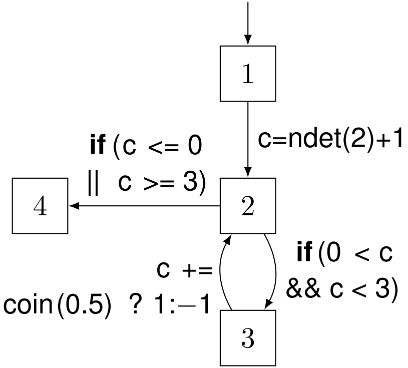
```
int main() {
1: int c = ndet(2) + 1;
2: while (0 < c && c < 3) {
3:   c += coin(0.5) ? 1:-1;
   }
4: assert(c > 0);
}
```

Outline

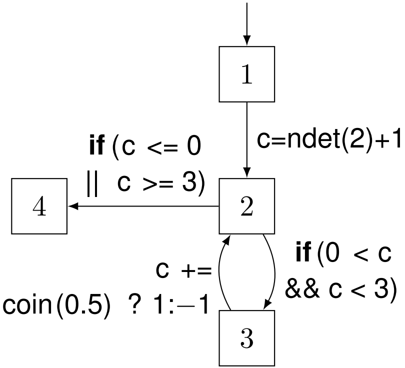


Gambler's ruin

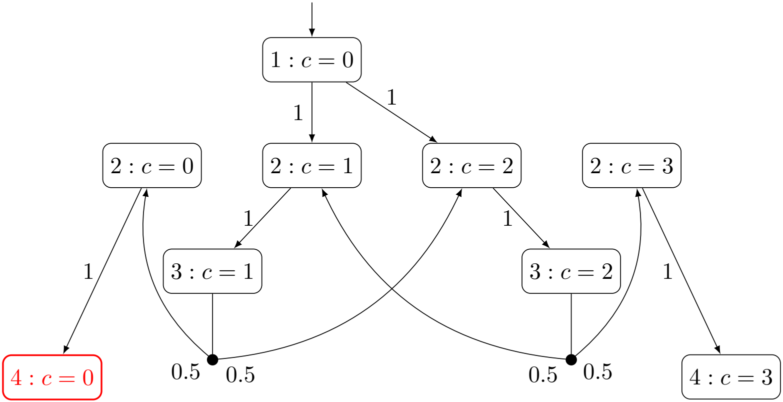
```
int main() {
1: int c = ndet(2) + 1;
2: while (0 < c && c < 3) {
3:   c += coin(0.5) ? 1:−1;
   }
4: assert(c > 0);
}
```



Semantics of probabilistic programs

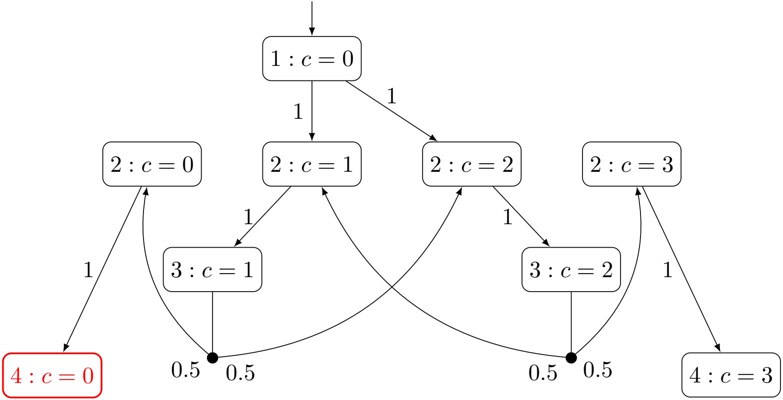


Semantics of probabilistic programs



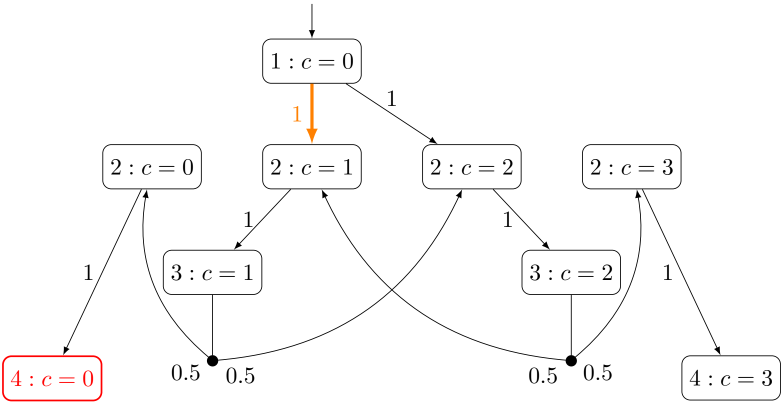
Schedulers

- Schedulers resolve non-determinism by mapping each state to a non-deterministic choice



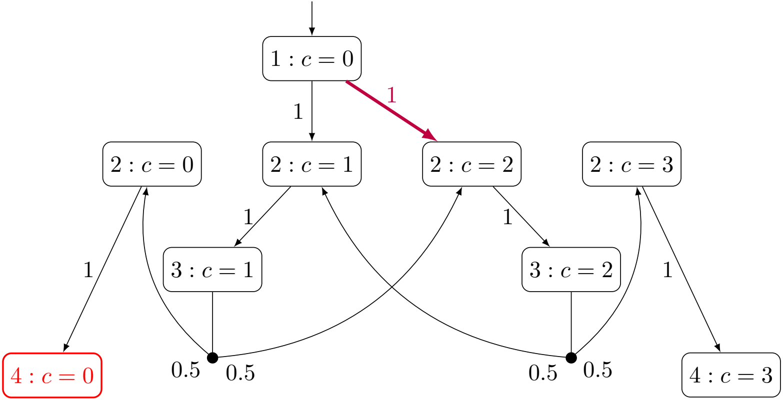
Schedulers

- Schedulers resolve non-determinism by mapping each state to a non-deterministic choice



Schedulers

- Schedulers resolve non-determinism by mapping each state to a non-deterministic choice



Probabilities

Problem

- Calculating *the* probability of a state is impossible
- Instead, there is a separate probability for each scheduler

Idea

- Define properties over minimal and maximal probabilities
- I.e. consider the *best-case* and the *worst-case* scheduler

RWTH AACHEN UNIVERSITY

Probabilities

Probability of $\lozenge$red

- Scheduler $c = 1$: $\frac{1}{2}^1 + \frac{1}{2}^3 + \frac{1}{2}^5 + \cdots = \frac{2}{3}$

RWTH AACHEN UNIVERSITY

Probabilities

Probability of $\Diamond$red

- Scheduler $c = 1$: $\frac{1}{2}^1 + \frac{1}{2}^3 + \frac{1}{2}^5 + \cdots = \frac{2}{3}$
- Scheduler $c = 2$: $\frac{1}{2} \times (\frac{1}{2}^1 + \frac{1}{2}^3 + \frac{1}{2}^5 + \dots) = \frac{1}{3}$

RWTH AACHEN UNIVERSITY

Probabilities

Probability of $\Diamond$red

- Scheduler $c = 1$: $\frac{1}{2}^1 + \frac{1}{2}^3 + \frac{1}{2}^5 + \cdots = \frac{2}{3}$
  Maximal probability
- Scheduler $c = 2$: $\frac{1}{2} \times (\frac{1}{2}^1 + \frac{1}{2}^3 + \frac{1}{2}^5 + \ldots) = \frac{1}{3}$
  Minimal probability

Probabilities

Probability of $\lozenge$red

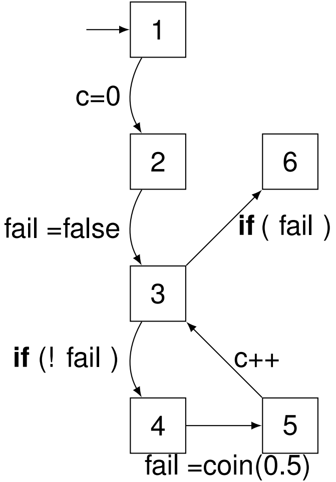
- Scheduler $c = 1$: $\frac{1}{2}^1 + \frac{1}{2}^3 + \frac{1}{2}^5 + \dots = \frac{2}{3}$
- Scheduler $c = 2$: $\frac{1}{2} \times (\frac{1}{2}^1 + \frac{1}{2}^3 + \frac{1}{2}^5 + \dots) = \frac{1}{3}$

- Probabilities can be calculated using a linear optimisation problem

State space explosion

- MDP semantics often have too many (or even infinitely many) states

```
int main() {
1:   int c = 0;
2:   bool fail = false;
3:   while(!fail) {
4:     fail = coin(0.5);
5:     c++;
6:   }
}
```



RWTH AACHEN UNIVERSITY

State space explosion

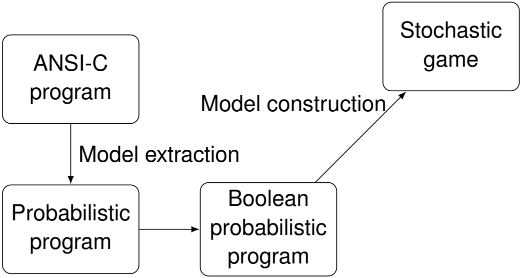
Problem

$$|\mathcal{M}| \approx \underbrace{2^{32} + \cdots + 2^{32}}_{\text{Number of 32-bit integer variables}} \times \text{Number of statements}$$

Idea

Reduce the value ranges of variables

Outline



ANSI-C program
Model extraction
Probabilistic program
Boolean probabilistic program
Model construction
Stochastic game

Predicate abstraction

Solution

Replace original variables with predicates (boolean expressions)

Predicate abstraction

Solution

Replace original variables with predicates (boolean expressions)

```
int main() {
  int i = 0;
  while(true) {
    assert(i != 443);
    if(i < 10) {
      i = i + 1;
    }
    else i = i + 2;
  }
}
```

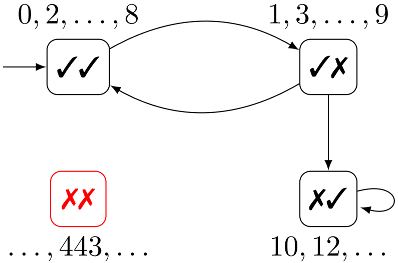
Predicate abstraction

Solution

Replace original variables with predicates (boolean expressions)

```
int main() {
  int i = 0;
  while(true) {
    assert(i != 443);
    if(i < 10) {
      i = i + 1;
    }
    else i = i + 2;
  }
}
```

Predicates: $i < 10, i \bmod 2 = 0$



Min. and max. $Prob(\Diamond$✗✗$) = 0$

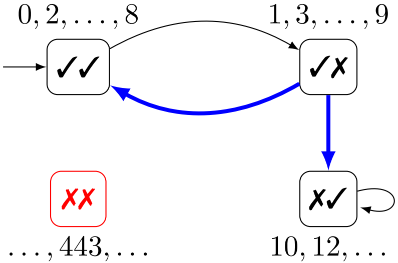
Predicate abstraction

Solution

Replace original variables with predicates (boolean expressions)

```
int main() {
  int i = 0;
  while(true) {
    assert(i != 443);
    if(i < 10) {
      i = i + 1;
    }
    else i = i + 2;
  }
}
```

Predicates: $i < 10, i \bmod 2 = 0$



Min. and max. $Prob(\Diamond ✗✗) = 0$

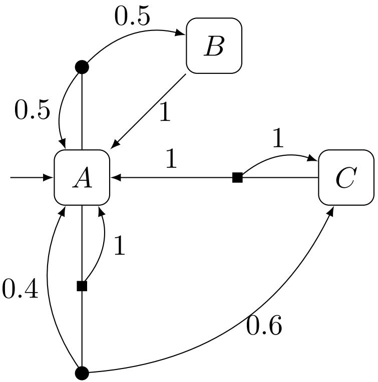
RWTH AACHEN UNIVERSITY

Two-player stochastic game

Definition

A two-player stochastic game is a tuple $\hat{\mathcal{M}} = (\hat{S}, \hat{s}_{init}, \hat{\mathbf{P}})$ with

- countable, non-empty set of states $\hat{S}$
- initial state $\hat{s}_{init} \in \hat{S}$
- transition function $\hat{\mathbf{P}} : \hat{S} \to \mathcal{P}(\mathcal{P}(Dist_{\hat{S}}))$

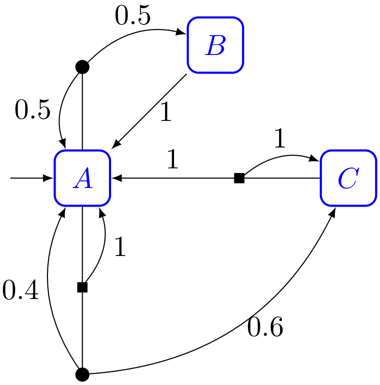


Two-player stochastic game

Definition

A two-player stochastic game is a tuple $\hat{\mathcal{M}} = (\hat{S}, \hat{s}_{init}, \hat{\mathbf{P}})$ with

- countable, non-empty set of states $\hat{S}$
- initial state $\hat{s}_{init} \in \hat{S}$
- transition function $\hat{\mathbf{P}} : \hat{S} \to \mathcal{P}(\mathcal{P}(Dist_{\hat{S}}))$



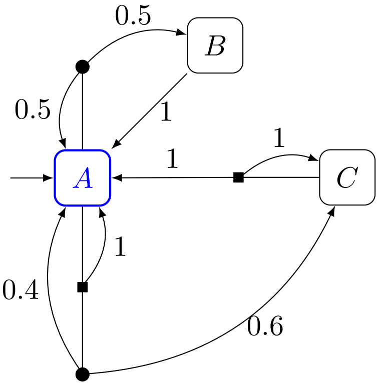
RWTH AACHEN UNIVERSITY

Two-player stochastic game

Definition

A two-player stochastic game is a tuple $\hat{\mathcal{M}} = (\hat{S}, \hat{s}_{init}, \hat{\mathbf{P}})$ with

- countable, non-empty set of states $\hat{S}$
- initial state $\hat{s}_{init} \in \hat{S}$
- transition function $\hat{\mathbf{P}} : \hat{S} \to \mathcal{P}(\mathcal{P}(Dist_{\hat{S}}))$

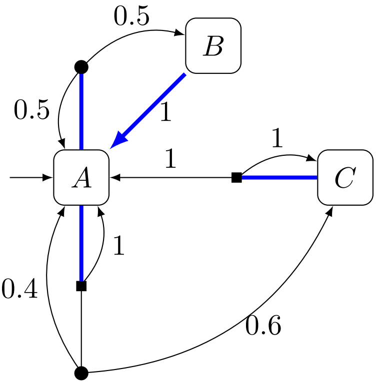


Two-player stochastic game

Definition

A two-player stochastic game is a tuple $\hat{\mathcal{M}} = (\hat{S}, \hat{s}_{init}, \hat{\mathbf{P}})$ with

- countable, non-empty set of states $\hat{S}$
- initial state $\hat{s}_{init} \in \hat{S}$
- transition function $\hat{\mathbf{P}} : \hat{S} \to \mathcal{P}(\mathcal{P}(Dist_{\hat{S}}))$

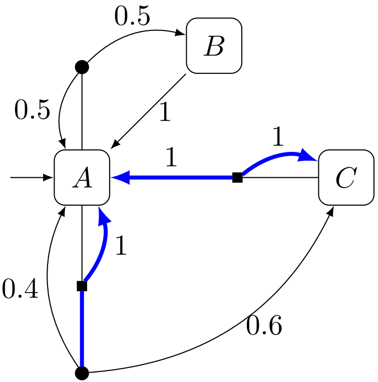


Two-player stochastic game

Definition

A two-player stochastic game is a tuple $\hat{\mathcal{M}} = (\hat{S}, \hat{s}_{init}, \hat{\mathbf{P}})$ with

- countable, non-empty set of states $\hat{S}$
- initial state $\hat{s}_{init} \in \hat{S}$
- transition function $\hat{\mathbf{P}} : \hat{S} \to \mathcal{P}(\mathcal{P}(Dist_{\hat{S}}))$

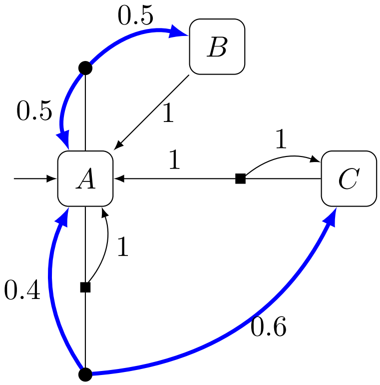


Two-player stochastic game

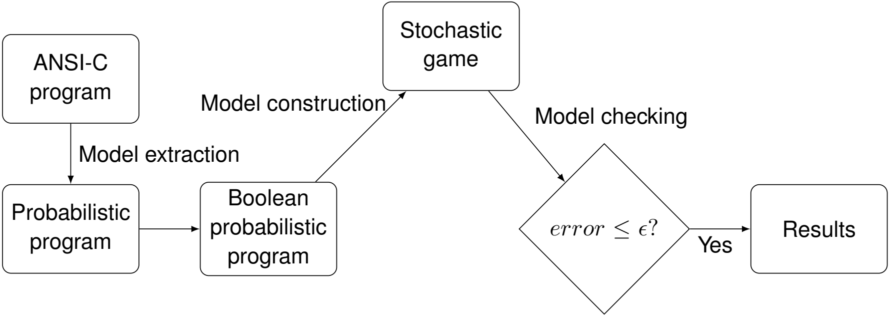
Definition

A two-player stochastic game is a tuple $\hat{\mathcal{M}} = (\hat{S}, \hat{s}_{init}, \hat{\mathbf{P}})$ with

- countable, non-empty set of states $\hat{S}$
- initial state $\hat{s}_{init} \in \hat{S}$
- transition function $\hat{\mathbf{P}} : \hat{S} \to \mathcal{P}(\mathcal{P}(Dist_{\hat{S}}))$

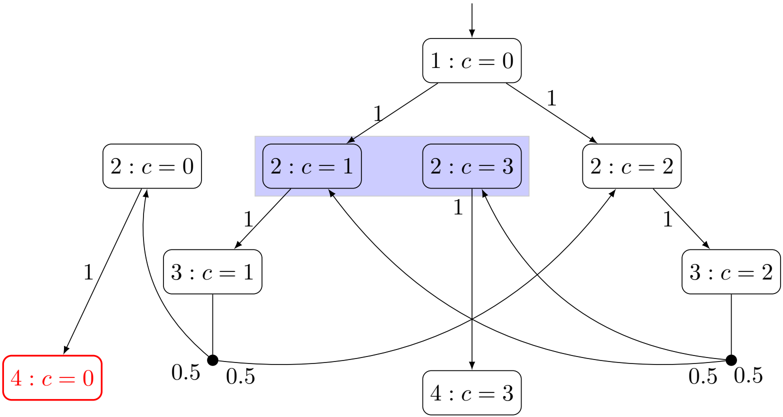


Outline

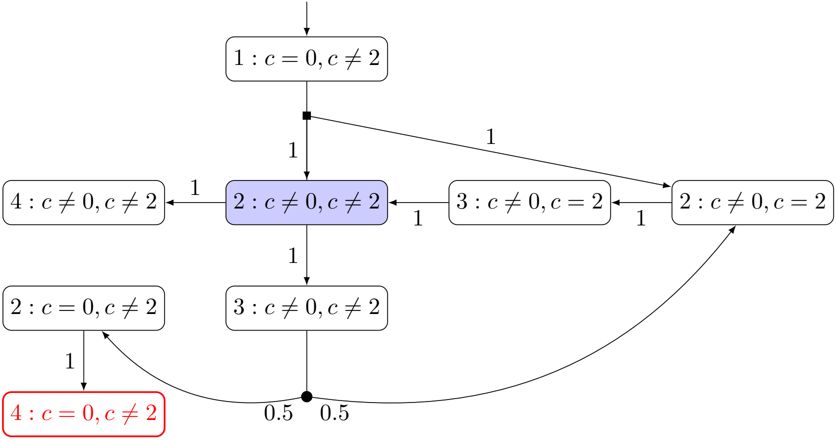

ANSI-C program
Model extraction
Probabilistic program
Boolean probabilistic program
Model construction
Stochastic game
Model checking
$error \leq \epsilon?$
Yes
Results

Abstraction of Gambler's ruin

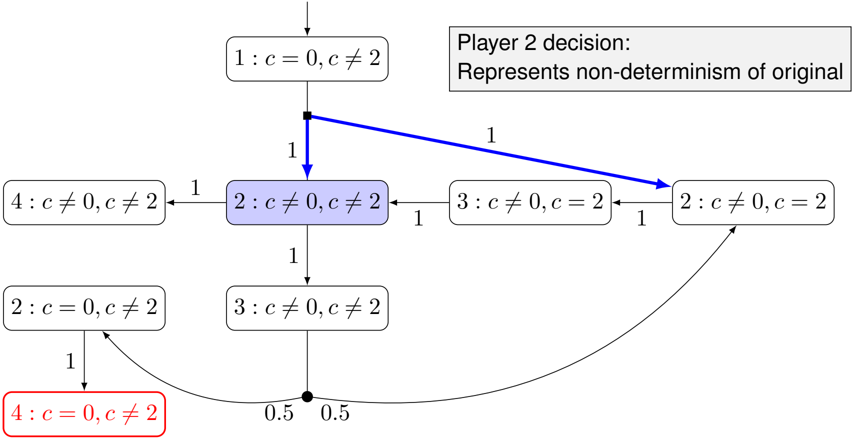
Predicates: $c = 0, c = 2$



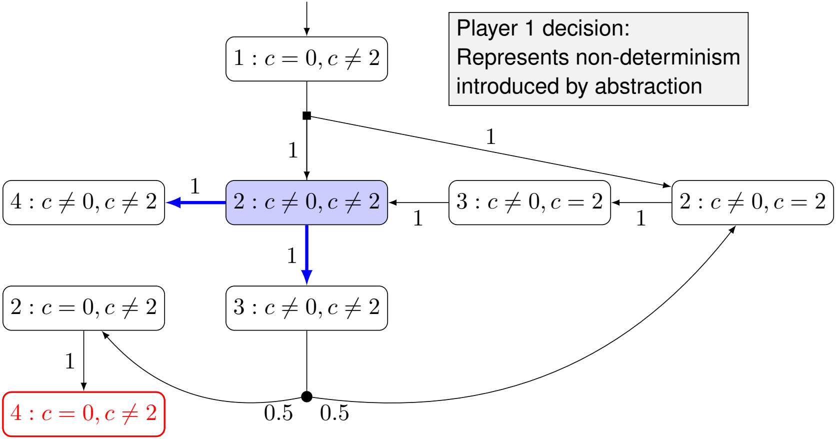
Abstraction of Gambler's ruin



Abstraction of Gambler's ruin

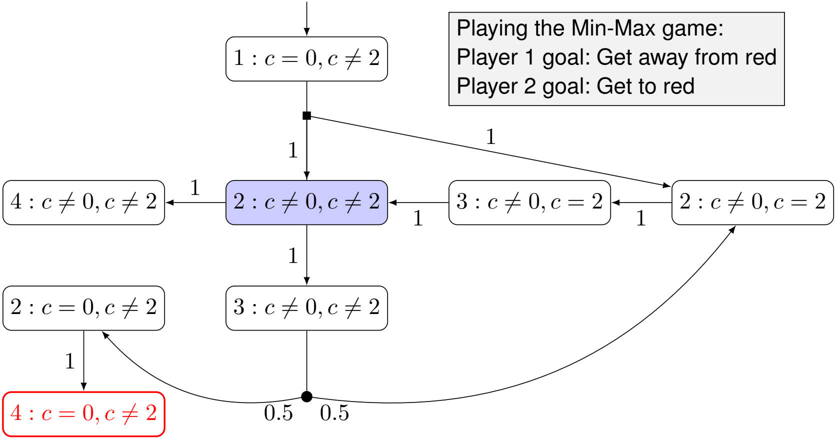


Abstraction of Gambler's ruin



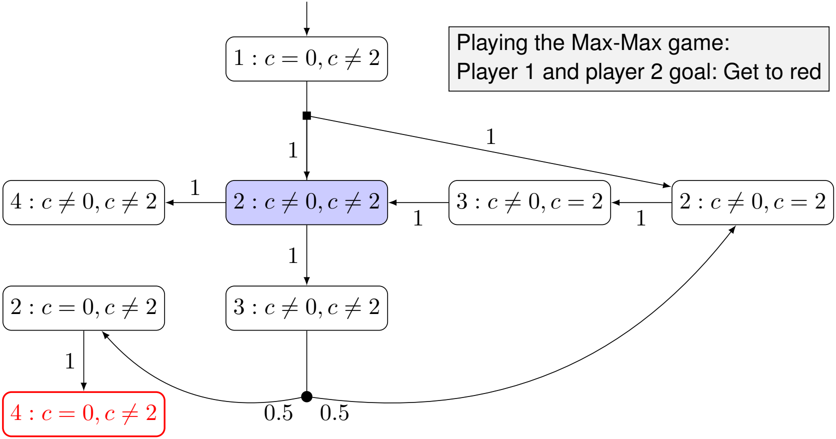
Abstraction of Gambler's ruin

Playing the Min-Max game:
Player 1 goal: Get away from red
Player 2 goal: Get to red



Abstraction of Gambler's ruin

Playing the Max-Max game:
Player 1 and player 2 goal: Get to red



RWTH AACHEN UNIVERSITY

Model checking

Theorem

Probabilities in abstraction give bounds on original probabilities

$$\mathbf{p}^{Min-Min}(\hat{\mathcal{F}}) \leq \mathbf{p}^{Min}(\mathcal{F}) \leq \mathbf{p}^{Max-Min}(\hat{\mathcal{F}})$$

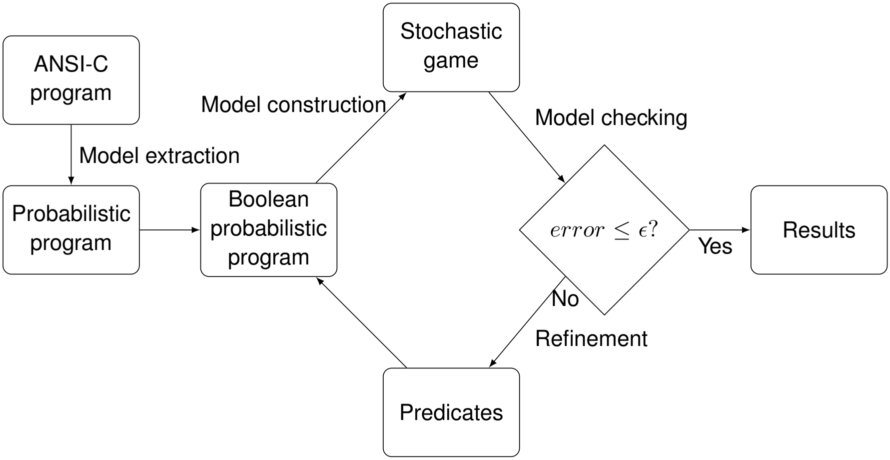
$$\mathbf{p}^{Min-Max}(\hat{\mathcal{F}}) \leq \mathbf{p}^{Max}(\mathcal{F}) \leq \mathbf{p}^{Max-Max}(\hat{\mathcal{F}})$$

$$\mathbf{p}^{Min-Max}(\Diamond \text{red}) = 0 \leq \mathbf{p}^{Max}(\Diamond \text{red}) \leq 1 = \mathbf{p}^{Max-Max}(\Diamond \text{red})$$

Problem

No meaningful information: error $\epsilon = 1 \implies$ Imprecise abstraction

Outline



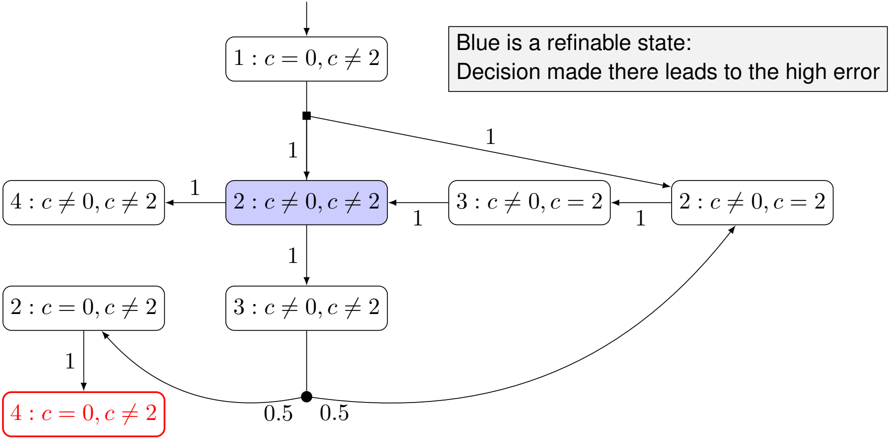
ANSI-C program
Model extraction
Probabilistic program
Boolean probabilistic program
Model construction
Stochastic game
Model checking
$error \leq \epsilon?$
Yes
Results
No
Refinement
Predicates

Abstraction refinement

Solution: Abstraction refinement

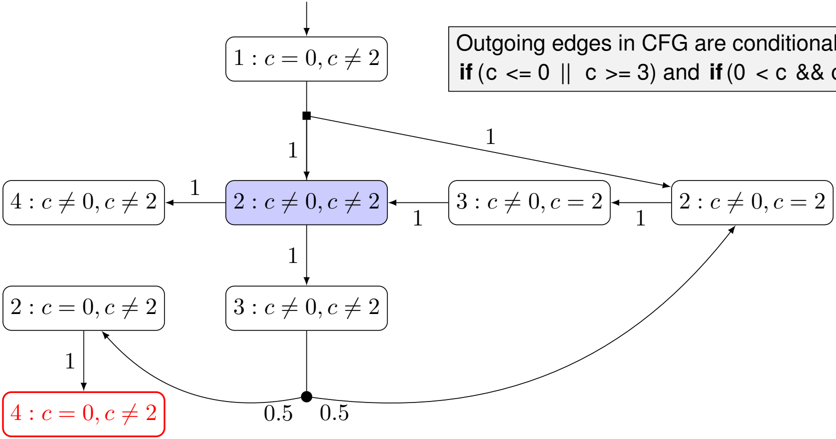
- Identify refinable state with distinct choices in min and max case, e.g.
  - state with highest error
    (difference between minimal and maximal probability)
  - state nearest to initial state
- Case distinction on the label of the outgoing edges

Abstraction of Gambler's ruin



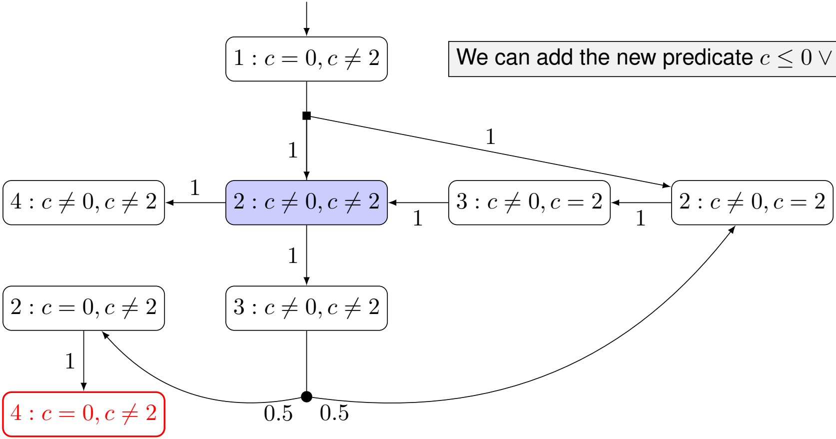
Abstraction of Gambler's ruin

Outgoing edges in CFG are conditionals
if (c <= 0 || c >= 3) and **if** (0 < c && c < 3)



Abstraction of Gambler's ruin

We can add the new predicate $c \leq 0 \vee c \geq 3$

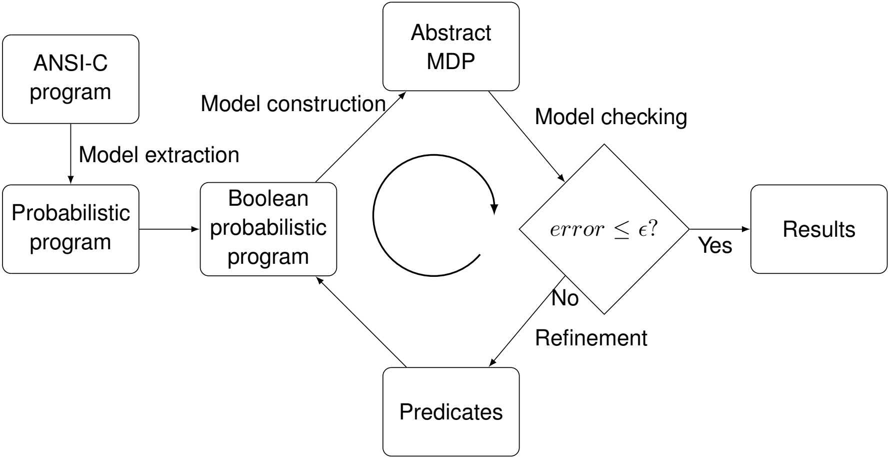


Abstraction of Gambler's ruin

- Model checking the abstraction gives exact results (error $\epsilon = 0$)
- Refined abstraction is isomorphic to the original program
  Using the predicates we can separate all possible values of c
- So here we haven't gained much
- But the method is very efficient for real-world examples

RWTH AACHEN UNIVERSITY

Abstraction-refinement loop



Thank you for your attention

Questions?