Abstraction Refinement for Probabilistic Software

Sascha Kurowski

Outline

ANSI-C program
```
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);
}
```
Gamblers’s ruin

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

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

- Likelihood of each possible outcome known
- Returns 1 with probability $p$ and 0 with probability $1 - p$
- E.g. randomization or network communication
int main() {
  int c = ndet(2) + 1;
  while (0 < c && c < 3) {
    c += coin(0.5) ? 1 : -1;
  }
  assert(c > 0);
}
ANSI-C program

Model extraction

Probabilistic program
```c
int main() {
    int c = ndet(2) + 1;
    while (0 < c && c < 3) {
        c += coin(0.5) ? 1 : -1;
    }
    assert(c > 0);
}
```
Semantics of probabilistic programs

c = n\text{det}(2) + 1

\begin{align*}
\text{if } (0 < c &\land c < 3) \\
\text{c += coin(0.5) ? 1:} &-1 \\
\text{if } (0 < c &\land c < 3)
\end{align*}
Semantics of probabilistic programs

Probabilistic programs
Schedulers resolve non-determinism by mapping each state to a non-deterministic choice.
Schedulers resolve non-determinism by mapping each state to a non-deterministic choice.
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
Probability of ♦red

- **Scheduler** $c = 1$: $\frac{1}{2} + \frac{1}{2} + \frac{3}{2} + \frac{5}{2} + \cdots = \frac{2}{3}$
Probability of ♦red

- **Scheduler** $c = 1$: $\frac{1}{2} + \frac{1}{2^3} + \frac{1}{2^5} + \cdots = \frac{2}{3}$
- **Scheduler** $c = 2$: $\frac{1}{2} \times (\frac{1}{2^1} + \frac{1}{2^3} + \frac{1}{2^5} + \cdots) = \frac{1}{3}$
Probability of ♦red

- **Scheduler** $c = 1$: \[ \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{3}{2} + \frac{1}{2} \times \frac{5}{2} + \cdots = \frac{2}{3} \]
  Maximal probability

- **Scheduler** $c = 2$: \[ \frac{1}{2} \times \left( \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{3}{2} + \frac{1}{2} \times \frac{5}{2} + \cdots \right) = \frac{1}{3} \]
  Minimal probability
Probability of ♦ red

- **Scheduler \( c = 1 \):** \( \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{3}{2} + \frac{1}{2} \times \frac{5}{2} + \cdots = \frac{2}{3} \)
- **Scheduler \( c = 2 \):** \( \frac{1}{2} \times \left( \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{3}{2} + \frac{1}{2} \times \frac{5}{2} + \cdots \right) = \frac{1}{3} \)

- Probabilities can be calculated using a linear optimisation problem.
MDP semantics often have too many (or even infinitely many) states

```plaintext
int main() {
    int c = 0;
    bool fail = false;
    while (!fail) {
        fail = coin(0.5);
        c++;
    }
}
```
State space explosion

Problem

\[ |M| \approx 2^{32} + \cdots + 2^{32} \times \text{Number of statements} \]

Number of 32-bit integer variables

Idea

Reduce the value ranges of variables
Predicate abstraction

Solution

Replace original variables with predicates (boolean expressions)
Solution

Replace original variables with predicates (boolean expressions)

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

Replace original variables with predicates (boolean expressions)

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

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

0, 2, ..., 8

1, 3, ..., 9

Min. and max. $Prob(\diamond \Box)$ = 0
Solution

Replace original variables with predicates (boolean expressions)

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

Predicates: \( i < 10, i \mod 2 = 0 \)

\( 0, 2, \ldots, 8 \)

\( 1, 3, \ldots, 9 \)

Min. and max. \( \text{Prob}(\diamond \text{XX}) = 0 \)
A two-player stochastic game is a tuple $\hat{M} = (\hat{S}, \hat{s}_{init}, \hat{P})$ with:

- A countable, non-empty set of states $\hat{S}$
- Initial state $\hat{s}_{init} \in \hat{S}$
- Transition function $\hat{P} : \hat{S} \rightarrow \mathcal{P}(\mathcal{P}(\text{Dist}\hat{S}))$
Definition

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

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

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

- countable, non-empty set of states $\hat{S}$
- initial state $\hat{s}_{\text{init}} \in \hat{S}$
- transition function $\hat{P} : \hat{S} \rightarrow \mathcal{P}(\mathcal{P}(\text{Dist}\hat{S}))$
A two-player stochastic game is a tuple $\mathcal{M} = (\hat{S}, \hat{s}_{\text{init}}, \hat{P})$ with

- countable, non-empty set of states $\hat{S}$
- initial state $\hat{s}_{\text{init}} \in \hat{S}$
- transition function $\hat{P} : \hat{S} \to \mathcal{P}(\mathcal{P}(\text{Dist}_\hat{S}))$
A two-player stochastic game is a tuple $\hat{M} = (\hat{S}, \hat{s}_{init}, \hat{P})$ with
- countable, non-empty set of states $\hat{S}$
- initial state $\hat{s}_{init} \in \hat{S}$
- transition function $\hat{P} : \hat{S} \rightarrow \mathcal{P}(\mathcal{P}(\text{Dist} \hat{S}))$
A two-player stochastic game is a tuple $\hat{M} = (\hat{S}, \hat{s}_{init}, \hat{P})$ with

- countable, non-empty set of states $\hat{S}$
- initial state $\hat{s}_{init} \in \hat{S}$
- transition function $\hat{P} : \hat{S} \rightarrow \mathcal{P}(\mathcal{P}(\text{Dist}\hat{S}))$
Outline

- ANSI-C program
- Probabilistic program
- Model extraction
- Model construction
- Stochastic game
- Model checking
- Abstraction of Probabilistic Programs
- Results

\(\text{error} \leq \epsilon?\) Yes
Abstraction of Gambler’s ruin

Predicates: $c = 0$, $c = 2$
Abstraction of Probabilistic Programs

Abstraction of Gambler’s ruin

Diagram showing the states and transitions in a probabilistic program, where the states are labeled with conditions on the variables c:
- 1: \( c = 0, c \neq 2 \)
- 2: \( c \neq 0, c \neq 2 \)
- 3: \( c \neq 0, c = 2 \)
- 4: \( c = 0, c \neq 2 \)

The diagram includes arrows indicating the probabilities of transitioning between states.
Abstraction of Gambler’s ruin

Player 2 decision:
Represents non-determinism of original
Abstraction of Gambler’s ruin

Player 1 decision:
Represents non-determinism introduced by abstraction
Abstraction of Gambler’s ruin

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

Probabilities in abstraction give bounds on original probabilities

\[ p^{\text{Min-Min}}(\hat{\mathcal{F}}) \leq p^{\text{Min}}(\mathcal{F}) \leq p^{\text{Max-Min}}(\hat{\mathcal{F}}) \]

\[ p^{\text{Min-Max}}(\hat{\mathcal{F}}) \leq p^{\text{Max}}(\mathcal{F}) \leq p^{\text{Max-Max}}(\hat{\mathcal{F}}) \]

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

Problem

No meaningful information: error \( \epsilon = 1 \) \( \implies \) Imprecise abstraction
Outline

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

Blue is a refinable state: Decision made there leads to the high error.
Abstraction of Gambler’s ruin

Outgoing edges in CFG are conditionals
if (c <= 0 || c >= 3) and if (0 < c && c < 3)
We can add the new predicate $c \leq 0 \lor c \geq 3$.

Abstraction of Gambler’s ruin.
Abstraction refinement

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
Abstraction-refinement loop
Thank you for your attention

Questions?