Proving Termination of Probabilistic Programs using Patterns

Lukas Westhofen

Chair of Computer Science 2
RWTH Aachen

4.2.2015

Outline

1 Motivation

2 Introduction to probabilistic programs

3 Almost-sure termination

4 Patterns

5 The algorithm

6 Conclusion

Based on *Proving termination of probabilistic programs using patterns* by Javier Esparza, Andreas Gaiser and Stefan Kiefer.

Motivation

Why do want to show termination of probabilistic programs?

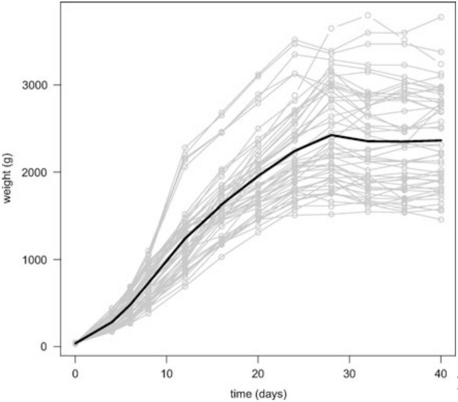
Motivation

Why do want to show termination of probabilistic programs?

$\rightarrow$ **First:** Understand the applications of probabilistic programs!

Applications

Biology Modeling populations, e.g. in agronomics



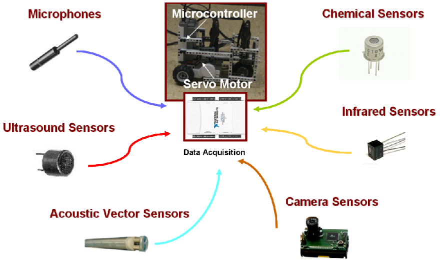
[1]

[1] *Introduction to Stochastic Models in Biology*, S. Ditlevsen and A. Samson

Applications

Biology Modeling populations, e.g. in agronomics

Robotics Collecting sensor data in a probabilistic way

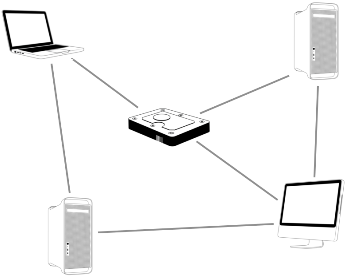


[2]

[2] Source: Washington University St. Louis

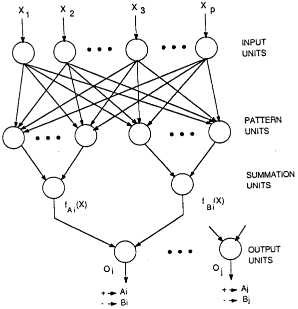
Applications

Biology	Modeling populations, e.g. in agronomics
Robotics	Collecting sensor data in a probabilistic way
Networks	Solve concurrency related problems



Applications

Biology Modeling populations, e.g. in agronomics
Robotics Collecting sensor data in a probabilistic way
Networks Solve concurrency related problems
Machine Learning Neural networks



[3]

[3] *Probabilistic Neural Networks for Classification, Mapping, or Associative Memory*, Donald F. Specht.

Proving Termination

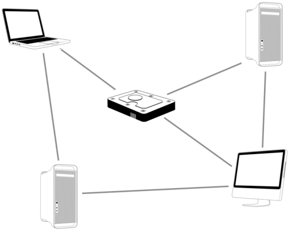
Why do want to show termination of probabilistic programs?

Proving Termination

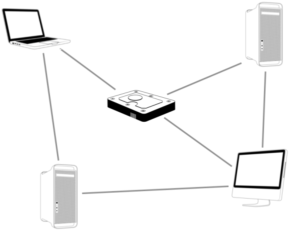
Why do want to show termination of probabilistic programs?

$\rightarrow$ Termination of algorithms used *in the field* is of utter importance!

The FireWire IEEE 1394 root contention protocol



The FireWire IEEE 1394 root contention protocol



Idea

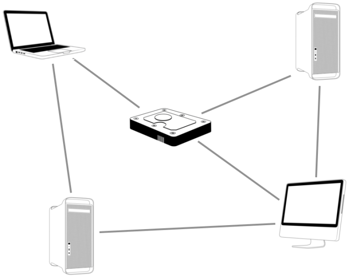
- FireWire builds a hierarchical tree-like network structure
- But no device is set as the root or parent node as default!

→ FireWire devices will have to build up the tree dynamically

The FireWire IEEE 1394 root contention protocol

Solution – The FireWire parent election algorithm (simplified)

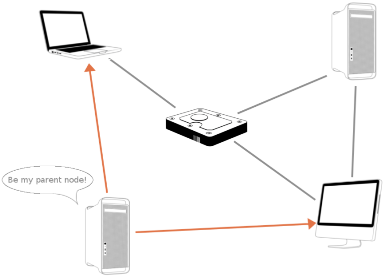
- Every node says *"Be my parent!"* to each neighboured node
- Every node that receives such a message saves the sending node as its child



The FireWire IEEE 1394 root contention protocol

Solution – The FireWire parent election algorithm (simplified)

- Every node says *"Be my parent!"* to each neighboured node
- Every node that receives such a message saves the sending node as its child



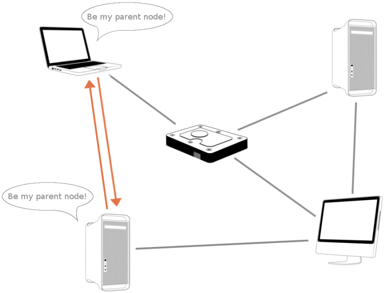
The FireWire IEEE 1394 root contention protocol

Is there any problem with this algorithm?

The FireWire IEEE 1394 root contention protocol

Is there any problem with this algorithm?

What happens if two nodes receiving *"Be my parent?"* at the same time from each other?



Probabilistic parent node election

To solve this problem, we introduce randomness!

Root contention solver

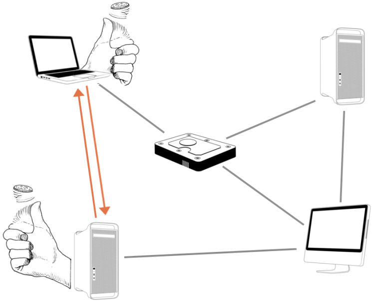
- Each node tosses a coin
- If both results differ, the node with the *head* coin will win and be the parent node

Probabilistic parent node election

To solve this problem, we introduce randomness!

Root contention solver

- Each node tosses a coin
- If both results differ, the node with the *head* coin will win and be the parent node



Probabilistic parent node election

$\rightarrow$ **Problem solved?**

Probabilistic parent node election

$\rightarrow$ **Problem solved?**

What happens if both coins always show the same outcome?

We need to show termination of the election algorithm, otherwise we can't be sure that the FireWire-communication would not come to a halt!

In this presentation, a termination prover for probabilistic programs will be presented.

1 Motivation

2 Introduction to probabilistic programs

3 Almost-sure termination

4 Patterns

5 The algorithm

6 Conclusion

Informal definition of probabilistic programs

- Similar to normal programs, but allow for randomization

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Informal definition of probabilistic programs

- Similar to normal programs, but allow for randomization

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

- `coin(p)` tosses a (possibly unfair) coin

Informal definition of probabilistic programs

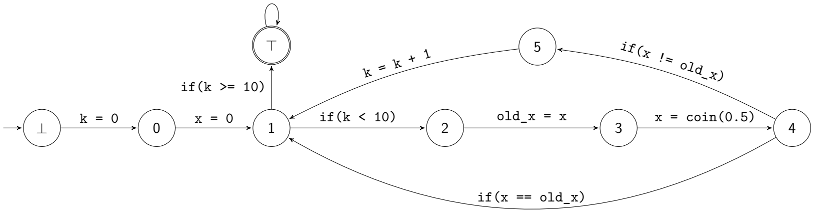
- Similar to normal programs, but allow for randomization

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

- coin(p) tosses a (possibly unfair) coin
- Additionally, we allow for nondeterminism via nondet()

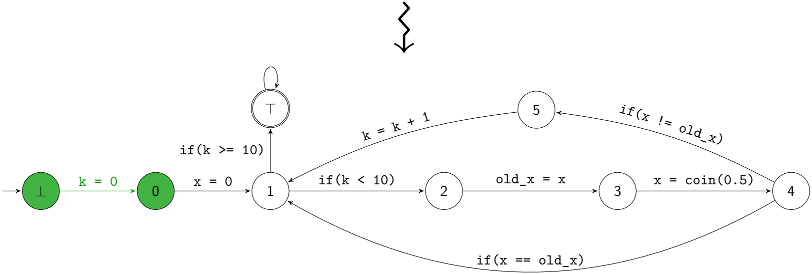
Formal definition via flow graphs

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```



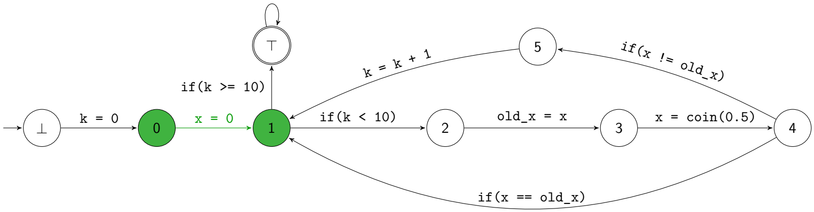
Formal definition via flow graphs

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```



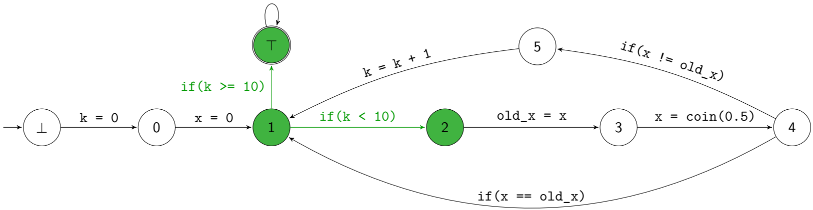
Formal definition via flow graphs

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```



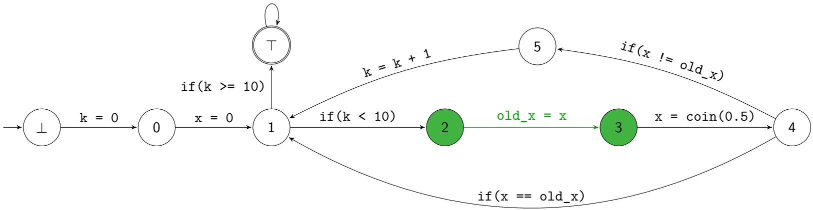
Formal definition via flow graphs

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```



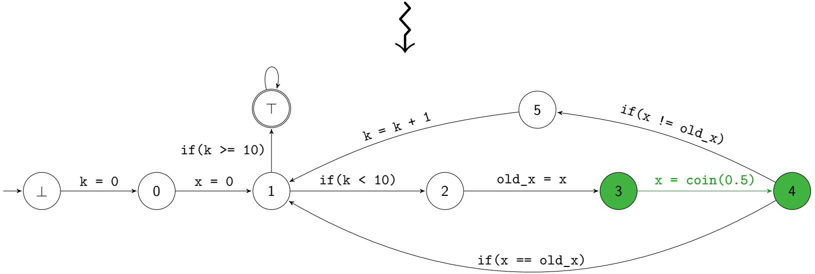
Formal definition via flow graphs

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```



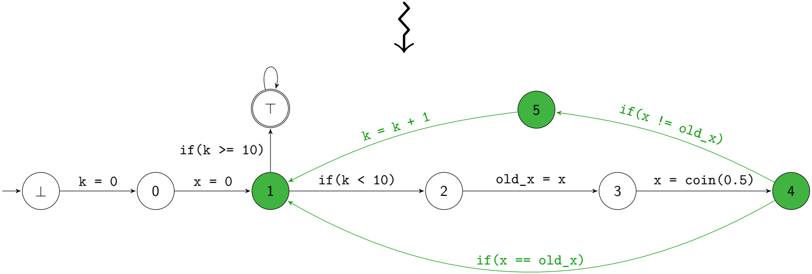
Formal definition via flow graphs

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```



Formal definition via flow graphs

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```



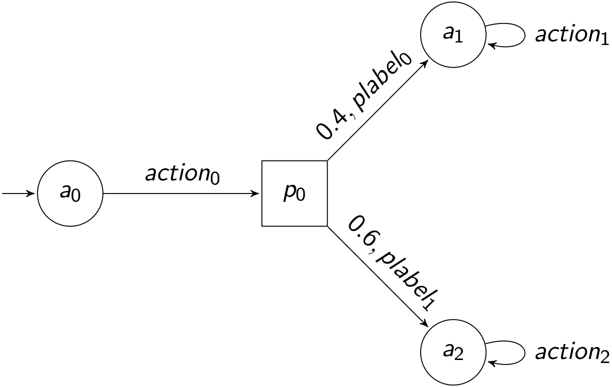
Markov Decision Processes (MDPs)

- Consist of **action nodes** and **probabilistic nodes**
- Probabilistic nodes allow for probabilistic successor choosing
- Action nodes allow to choose between several **system actions**

Markov Decision Processes (MDPs)

- Consist of **action nodes** and **probabilistic nodes**
- Probabilistic nodes allow for probabilistic successor choosing
- Action nodes allow to choose between several **system actions**

Example MDP:



Operational semantics of probabilistic programs

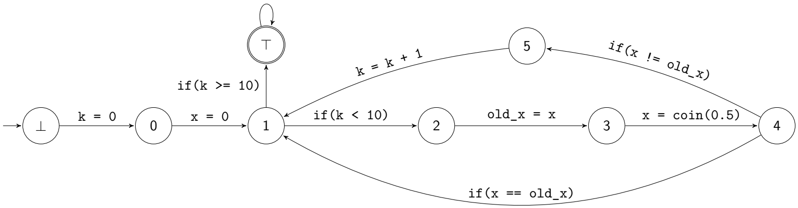
We will use an MDP to define the operational semantics of a probabilistic program P.

Its nodes consist of two tupel elements:

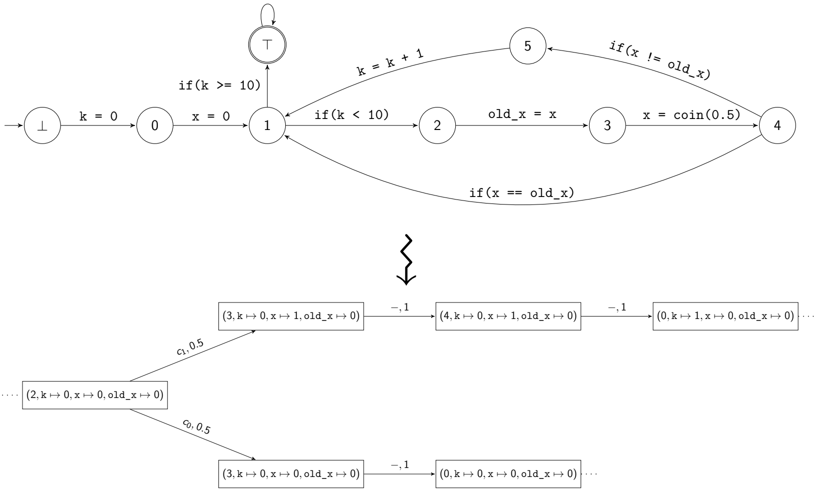
1. A node of the flow-graph representation of P
2. A program configuration σ

More details: *Friedrich Gretz, Joost-Pieter Katoen and Annabelle McIver*, **Operational versus weakest pre-expectation semantics for the probabilistic guarded command language**, 2014.

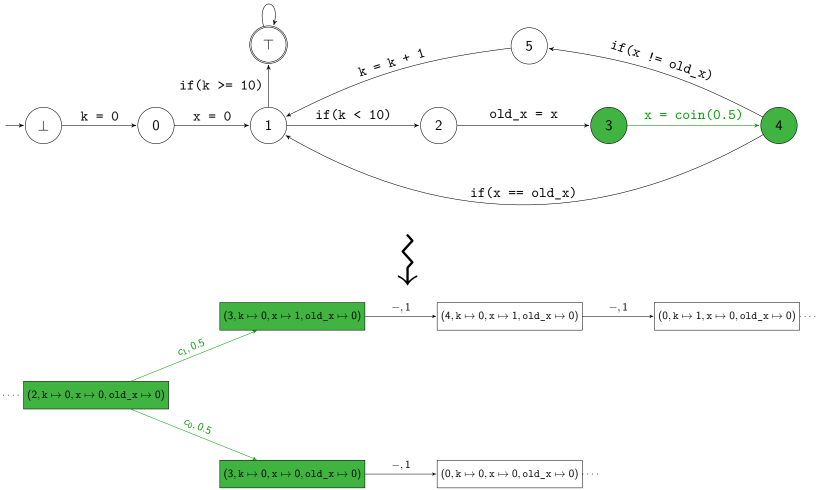
Example program MDP



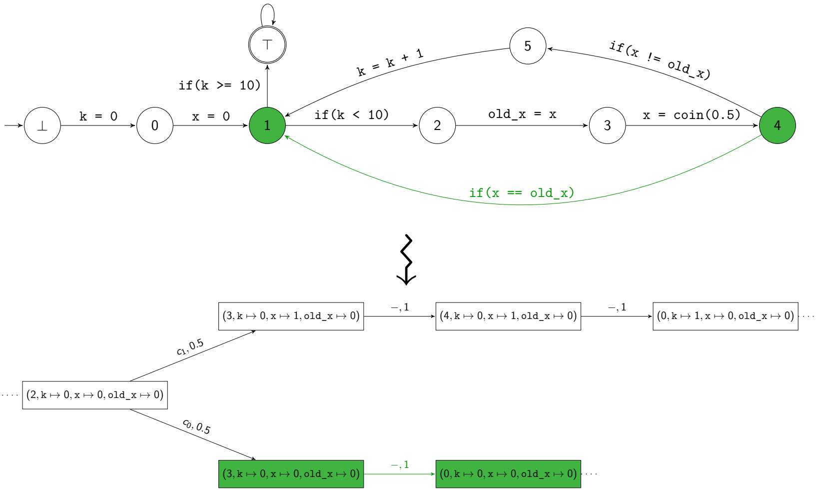
Example program MDP



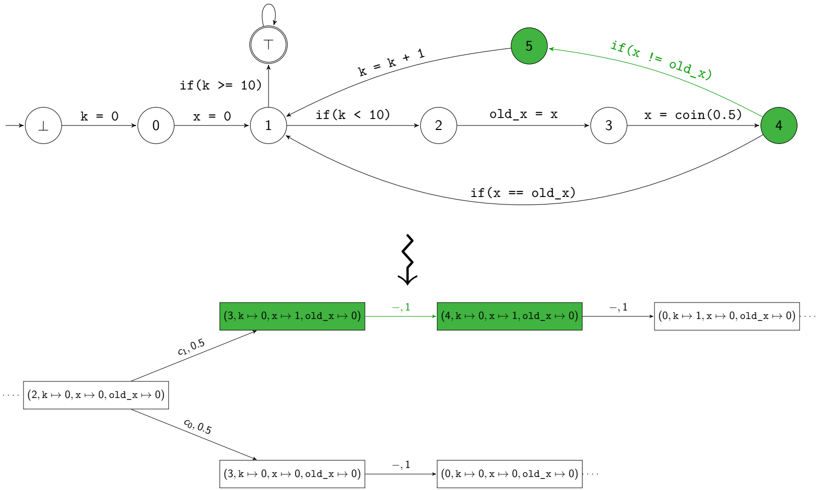
Example program MDP



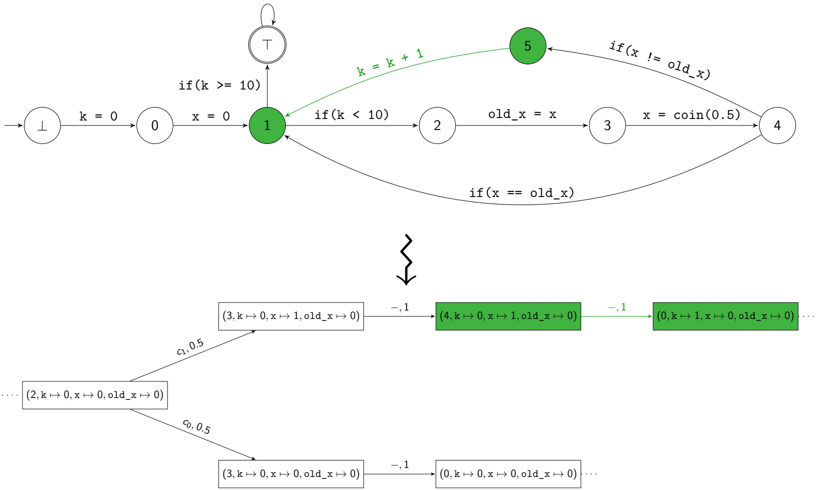
Example program MDP



Example program MDP



Example program MDP



Classes of probabilistic programs

Definition (Finite and weakly finite programs)

- For a **finite program** P holds that its associated MDP has only a finite number of nodes reachable from every initial node.
- A **weakly finite program** is additionally allowed to have an infinite number of initial nodes.

Classes of probabilistic programs

Definition (Finite and weakly finite programs)

- For a **finite program** P holds that its associated MDP has only a finite number of nodes reachable from every initial node.
- A **weakly finite program** is additionally allowed to have an infinite number of initial nodes.

Remark: The class of weakly finite programs contains parameterized programs!

1 Motivation

2 Introduction to probabilistic programs

3 Almost-sure termination

4 Patterns

5 The algorithm

6 Conclusion

Almost-sure termination

Termination of probabilistic programs is more involved!

Intuitively: Does this program “terminate”?

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Almost-sure termination

Termination of probabilistic programs is more involved!

Intuitively: Does this program "terminate"?

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Yes, it does, although there exist non-terminating runs!

Almost-sure termination

Termination of probabilistic programs is more involved!

Intuitively: Does this program “terminate”?

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Yes, it does, although there exist non-terminating runs!

- A non-terminating run: `coin(0.5)` yields always 0.
- A terminating run: `coin(0.5)` yields 01 ten times in a row.

Almost-sure termination

→ Ordinary termination notion does not work for probabilistic programs.

Goal

Devise a termination notion that excludes non-terminating runs which have probability 0. It should be sufficient to find a set of terminating runs that have probability 1.

Almost-sure termination

Definition (Almost-sure termination)

A probabilistic program P is *terminating almost surely* if the probability of all terminating runs of the associated MDP $\mathfrak{M}_P$, starting in an initial node, is 1.

Almost-sure termination

Definition (Almost-sure termination)

A probabilistic program P is *terminating almost surely* if the probability of all terminating runs of the associated MDP $\mathfrak{M}_P$, starting in an initial node, is 1.

Consequence: The algorithm needs to find a set of program runs which has the probability 1!

Example on almust-sure termination

The example program

```
k = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

terminates almost surely, because the set of terminating runs which have probability 1 is. . .

Example on almust-sure termination

The example program

```
k = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

terminates almost surely, because the set of terminating runs which have probability 1 is...

$$T = \{r \in Runs(\mathfrak{M}_P) | r \text{ alternates at least 10 times between } c_1 \text{ and } c_0\}$$

Example on almust-sure termination

The example program

```
k = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

terminates almost surely, because the set of terminating runs which have probability 1 is...

$$T = \{r \in Runs(\mathfrak{M}_P) | r \text{ alternates at least 10 times between } c_1 \text{ and } c_0\}$$

Generalize this concept?

1 Motivation

2 Introduction to probabilistic programs

3 Almost-sure termination

4 Patterns

5 The algorithm

6 Conclusion

Patterns

It's hard to show that a run-set has probability 1 – The pattern approach simplifies that!

Patterns

It's hard to show that a run-set has probability 1 – The pattern approach simplifies that!

Definition (Pattern)

A pattern Φ is a subset of C^ω, where $C = \{0, 1\}$. We denote Φ as the following expression to indicate its structure:
$\Phi = C^* w_1 C^* w_2 C^* w_3 C^* \ldots$ with $w_i \in C^*$.

Patterns

It's hard to show that a run-set has probability 1 – The pattern approach simplifies that!

Definition (Pattern)

A pattern Φ is a subset of C^ω, where $C = \{0, 1\}$. We denote Φ as the following expression to indicate its structure:
$\Phi = C^* w_1 C^* w_2 C^* w_3 C^* \dots$ with $w_i \in C^*$.

- w_i parts denote the important coin-toss outcomes
- C^* stands for irrelevant and finite parts of the run

Patterns

An example pattern. . .

$$\Phi = C^*010C^*11C^*(01C^*)^\omega$$

Patterns

An example pattern...

$$\Phi = C^* 010 C^* 11 C^* (01 C^*)^\omega$$

Remark 1: It is of importance that after getting w_i as a coin toss outcome sequence, we will eventually see w_{i+1} later.

Patterns

An example pattern...

$$\Phi = C^* 010 C^* 11 C^* (01 C^*)^\omega$$

Remark 1: It is of importance that after getting w_i as a coin toss outcome sequence, we will eventually see w_{i+1} later.

Remark 2: Patterns aren't just regular expressions over omega-languages!

Terminating patterns

Combining probabilistic programs and patterns:

Definition (Pattern-conforming runs)

A run r of a probabilistic program P *conforms* a pattern Φ if the coin toss outcomes of r match the structure defined by the pattern.

Terminating patterns

Combining probabilistic programs and patterns:

Definition (Pattern-conforming runs)

A run r of a probabilistic program P *conforms* a pattern Φ if the coin toss outcomes of r match the structure defined by the pattern.

Combining almost-sure termination and patterns:

Definition (Terminating patterns)

A pattern Φ is *terminating* if **every** Φ-conforming run eventually reaches the final state $\top$, i.e. the run terminates.

Example on terminating patterns

Our example has various patterns, but only some are terminating.

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Which pattern is terminating?

Example on terminating patterns

Our example has various patterns, but only some are terminating.

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Which pattern is terminating?

- $\Phi_1 = C^* 10 C^* 00 C^\omega$?

Example on terminating patterns

Our example has various patterns, but only some are terminating.

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Which pattern is terminating?

- $\Phi_1 = C^*10C^*00C^\omega$? **X**

Example on terminating patterns

Our example has various patterns, but only some are terminating.

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Which pattern is terminating?

- $\Phi_1 = C^*10C^*00C^\omega$? X
- $\Phi_2 = C^*(10)^{10}C^\omega$?

Example on terminating patterns

Our example has various patterns, but only some are terminating.

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Which pattern is terminating?

- $\Phi_1 = C^*10C^*00C^\omega$? X
- $\Phi_2 = C^*(10)^{10}C^\omega$? ✓

Example on terminating patterns

Our example has various patterns, but only some are terminating.

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Which pattern is terminating?

- $\Phi_1 = C^*10C^*00C^\omega$? X
- $\Phi_2 = C^*(10)^{10}C^\omega$? ✓
- $\Phi_3 = (C^*10)^\omega$?

Example on terminating patterns

Our example has various patterns, but only some are terminating.

```
k = 0;
x = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Which pattern is terminating?

- $\Phi_1 = C^*10C^*00C^\omega$? X
- $\Phi_2 = C^*(10)^{10}C^\omega$? ✓
- $\Phi_3 = (C^*10)^\omega$? ✓

Correctness of the pattern approach – Theorems

1. The probability of every pattern is 1.

Correctness of the pattern approach – Theorems

1. The probability of every pattern is 1.

Intuition on why this holds:

- Stochastic theory guarantees that when infinitely often tossing coins, we will eventually see every possible finite coin-toss outcome sequence!
- Additionally, the C^*-parts of the pattern allow for arbitrary, but finite coin-toss outcomes between those finite, fixed parts.

Correctness of the pattern approach – Theorems

2. If P has a terminating pattern, then P terminates almost-surely.

Correctness of the pattern approach – Theorems

3. Every almost-surely terminating finite-program has a simple terminating pattern of the form $\Phi = (C^* w)^\omega$ for some $w \in C$.

```
k = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Correctness of the pattern approach – Theorems

3. Every almost-surely terminating finite-program has a simple terminating pattern of the form $\Phi = (C^* w)^\omega$ for some $w \in C$.

```
k = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Simple terminating pattern for the example is $\Phi = (C^* 01)^\omega$.

Correctness of the pattern approach – Theorems

3. Every almost-surely terminating finite-program has a simple terminating pattern of the form $\Phi = (C^* w)^\omega$ for some $w \in C$.

```
k = 0;
while(k < 10) {
    old_x = x;
    x = coin(0.5)
    if(x != old_x)
        k = k + 1;
}
```

Simple terminating pattern for the example is $\Phi = (C^* 01)^\omega$.

Conclusion: The algorithm needs to construct a terminating pattern! (And for finite programs, only a simple terminating pattern)

1 Motivation

2 Introduction to probabilistic programs

3 Almost-sure termination

4 Patterns

5 The algorithm

6 Conclusion

Overview

Goal: Find a terminating pattern!

Overview

Goal: Find a terminating pattern!

Two subparts:

1. Iteratively construct a pattern (*Pattern constructor*)
2. Check if this pattern is terminating (*Pattern checker*)

Pattern checker

Pattern checker

- **Input:** A pattern Φ and a probabilistic program P.
- **Output:** True if Φ is a terminating pattern of P, and a counterexample (lasso) otherwise.

In practice, this pattern checker is realized via replacing the probabilistic programs parts with nondeterministic ones and passing it to *SPIN* and *ARMC*.

Using SPIN and ARMC

Model-Checkers

- Verify properties of systems
- Return counterexamples if the property does not hold

Using SPIN and ARMC

Model-Checkers

- Verify properties of systems
- Return counterexamples if the property does not hold

How to express those properties? Logics such as LTL!

Using SPIN and ARMC

Model-Checkers

- Verify properties of systems
- Return counterexamples if the property does not hold

How to express those properties? Logics such as LTL!

SPIN

- One of the most popular model-checkers
- Takes a *Finite* State Machine and an LTL formula

ARMC

- Is able to verify properties over infinite systems (Longer computation time)
- Important for weakly finite programs!

The pattern checker – Intuition

A basic overview on how to implement such a pattern checker:

Pattern checker

1. Transform a given probabilistic program P into a non-probabilistic program P' employing nondeterminism
2. Execute a model checker on P' to check whether P' has a run induced by Φ which is non-terminating

The pattern construtor – For finite programs

Overview

1. Iterate until a terminating pattern was constructed
2. For every iteration, construct a new pattern which excludes loops induced by previously constructed patterns

The pattern construtor – For finite programs

Constructs a *simple* terminating pattern $\Phi = (C^* w)^\omega$.

Data: A probabilistic program P and a baseword $s_0 \in C^*$.
Result: *True* if P terminates almost surely, *false* otherwise.

return *true*

The pattern construtor – For finite programs

Constructs a *simple* terminating pattern $\Phi = (C^* w)^\omega$.

Data: A probabilistic program P and a baseword $s_0 \in C^*$.
Result: *True* if P terminates almost surely, *false* otherwise.
$i := 0$
while $(C^* s_i)^\omega$ *is not a terminating pattern* **do**

end
return *true*

The pattern construtor – For finite programs

Constructs a *simple* terminating pattern $\Phi = (C^* w)^\omega$.

Data: A probabilistic program P and a baseword $s_0 \in C^*$.
Result: *True* if P terminates almost surely, *false* otherwise.

$i := 0$
while $(C^* s_i)^\omega$ *is not a terminating pattern* **do**
  $l_i :=$ lasso, taken from the termination checker.
  $u_i :=$ loop of l_i.
  if $u_i = \epsilon$ **then**
    return *false*
  else
    $s_{i+1} :=$ shortest word that has s_0 as prefix and is not an infix of any u_k^ω for $k \in \{1, \ldots, i\}$.
  end
  $i := i + 1$.
end
return *true*

The pattern construtor – For finite programs

Constructs a *simple* terminating pattern $\Phi = (C^* w)^\omega$.

Data: A probabilistic program P and a baseword $s_0 \in C^*$.
Result: *True* if P terminates almost surely, *false* otherwise.

$i := 0$
while $(C^* s_i)^\omega$ *is not a terminating pattern* **do**
  $l_i :=$ lasso, taken from the termination checker.
  $u_i :=$ loop of l_i.
  if $u_i = \epsilon$ **then**
    return *false*
  else
    $s_{i+1} :=$ shortest word that has s_0 as prefix and is not an infix of any u_k^ω for $k \in \{1, \dots, i\}$.
  end
  $i := i + 1$.
end
return *true*

Example on board.

The pattern construtor – For weakly finite programs

Overview

1. Iterate over every initial node i_k
2. In every iteration, check whether the finite program P_{i_k}, which is P but fixes i_k as its only initial node, terminates
3. Append the word of the simple terminating pattern to the new pattern
4. **Ask a human to extrapolate a general pattern**

The pattern constructor – For weakly finite programs

Data: A weakly finite probabilistic program P.
Result: *True* and the terminating pattern if P terminates almost surely, *false* otherwise.

The pattern constructor – For weakly finite programs

Data: A weakly finite probabilistic program P.
Result: *True* and the terminating pattern if P terminates almost surely, *false* otherwise.

Fix an enumeration $i_1, i_2, \ldots$ of $Init^P$.
$k := 0$
while *true* **do**

end
return *true*

The pattern constructor – For weakly finite programs

Data: A weakly finite probabilistic program P.
Result: *True* and the terminating pattern if P terminates almost surely, *false* otherwise.

Fix an enumeration $i_1, i_2, \ldots$ of $Init^P$.
$k := 0$
while *true* **do**
    Construct P_{i_k}.
    if P_{i_k} *is almost-sure terminating* **then**
    else
    end
end
return *true*

The pattern constructor – For weakly finite programs

Data: A weakly finite probabilistic program P.
Result: *True* and the terminating pattern if P terminates almost surely, *false* otherwise.

Fix an enumeration $i_1, i_2, \ldots$ of $Init^P$.
$k := 0$
while *true* **do**
    Construct P_{i_k}.
    if P_{i_k} *is almost-sure terminating* **then**
        Φ_{i_k} := simple terminating pattern $C^* w_{i_k} C^\omega$ of P_{i_k} using $w_{i_{k-1}}$ as a baseword.
        $\Phi_k := C^* w_{i_1} C^* w_{i_2} \ldots C^* w_{i_k} C^\omega$.
        if *human is able to extrapolate a sequence* $(w_{i_n})_{n \in \mathbb{N}}$ *when given* Φ_k **then**
            if $\Phi = C^* w_{i_1} C^* w_{i_2} \ldots$ *is a terminating pattern* **then**
                return *true and* Φ.
            end
        end
        $k := k + 1$
    else
        return *false*
    end
end
return *true*

Example on board.

Results and applications

- The algorithm is able to prove termination **fully automated** for finite programs and **semi-automated** for weakly-finite programs.

Results and applications

- The algorithm is able to prove termination **fully automated** for finite programs and **semi-automated** for weakly-finite programs.
- Is correct for both program classes, but only complete for finite programs.

Results and applications

- The algorithm is able to prove termination **fully automated** for finite programs and **semi-automated** for weakly-finite programs.
- Is correct for both program classes, but only complete for finite programs.
- Applied on various exemplary programs, such as
  - *FireWire* (weakly-finite, 1m53s): $w_i = 010$
  - *Randomwalk* (weakly-finite, 1m45s): $w_i = 0^i$
  - *BRP* (weakly-finite, 45m33s): $w_i = 00$

Results and applications

- The algorithm is able to prove termination **fully automated** for finite programs and **semi-automated** for weakly-finite programs.
- Is correct for both program classes, but only complete for finite programs.
- Applied on various exemplary programs, such as
  - *FireWire* (weakly-finite, 1m53s): $w_i = 010$
  - *Randomwalk* (weakly-finite, 1m45s): $w_i = 0^i$
  - *BRP* (weakly-finite, 45m33s): $w_i = 00$
- Pattern checker was not presented, but it heavily relies on model checkers such as ARMC and SPIN. Most of the time the algorithm waits for those tools to return an output
  - $\rightarrow$ Best way to enhance the algorithm is to enhance the model checkers!

Conclusion

We are now easily able to prove termination for probabilistic programs using the pattern approach!