Principles of Probabilistic Programming

Lectures at EWSCS 2020 Winter School

Joost-Pieter Katoen

EWSCS 2020, Palmse, Estonia
Probabilistic programs

What?

Programs with random assignments, conditioning and usual control-flow constructs

Why?

- Random assignments: to describe randomised algorithms
- Conditioning: to describe stochastic decision making
Applications

Quantum Computing

Security

Machine Learning

Bayesian Networks

Robotics

Approximate Computing

Randomised Algorithms
Planning in AI: robot navigation

```javascript
var updateBelief = function(belief, observation, action){
    return Infer({ model: { 
        var state = sample(belief); 
        var predictedNextState = transition(state, action); 
        var predictedObservation = observe(predictedNextState); 
        condition(_.isEqual(predictedObservation, observation)); 
        return predictedNextState; 
    } });
}

var act = function(belief) { 
    return Infer({ model: { 
        var action = uniformDraw(actions); 
        var eu = expectedUtility(belief, action); 
        factor(alpha * eu); 
        return action; 
    } });
}

var expectedUtility = function(belief, action) { 
    return expectation({ 
        Infer({ model: { 
            var state = sample(belief); 
            var u = utility(state, action); 
            if (state.terminateAfterAction) {
                return u;
            } else {
                var nextState = transition(state, action); 
                var nextObservation = observe(nextState); 
                var nextBelief = updateBelief(belief, nextObservation, action); 
                var nextAction = sample(act(nextBelief)); 
                return u + expectedUtility(nextBelief, nextAction); 
            }
        } });
    });
}
```

Uncertainty: noisy sensors and actuators, unknown environment

---

5 Evans et al., Modeling Agents with Probabilistic Programs, 2019
Security: The RSA-OAEP protocol

Correctness proof took more than 20 years
What is probabilistic programming?

Printer troubleshooting in Windows 95

How likely is it that your print is garbled given that the ps-file is not and the page orientation is portrait?

[Ramanna et al., Emerging Paradigms in Machine Learning, 2013]
What is probabilistic programming?

Languages:

- Probabilistic C
- ProbLog
- Church
- webPPL
- Figaro
- PyMC
- Tabular
- R2

probabilistic-programming.org

A. Pfeffer

N. Goodman
Issue 1: Program correctness

- Classical programs:
  - A program is correct with respect to a (formal) specification
    “for input array A, the output array B is sorted and contains all elements contained in A”
  - Defines a deterministic input-output relation
  - Partial correctness: if an output is produced, it is correct
  - Total correctness: in addition, the program terminates

- Probabilistic programs:
  - They do not always generate the same output
  - They generate a probability distribution over possible outputs
### Issue 2: Termination

#### Classical programs:
- They terminate (on a given/all inputs), or they do not.
- If they terminate, they take **finitely many steps** to do so.
- Showing program termination is **undecidable** (halting problem).

#### Probabilistic programs:
- They terminate (or not) with a certain likelihood.
- They may have diverging runs whose likelihood is zero.
- They may take infinitely many steps (on average) to terminate, even if they terminate with probability one!
- Showing “probability-one” termination is **“more” undecidable** and showing they do in finite time on average, even more!
Issue 3: The program’s runtime

- Classical programs:
  - They have a deterministic, fixed run-time for a given input
  - Runtimes of terminating programs in sequence are compositional:
    
    \[ \text{if } P \text{ and } Q \text{ terminate in } n \text{ and } k \text{ steps, then } P;Q \text{ halts in } n+k \text{ steps} \]
  - Analysis techniques: recurrence equations, tree analysis, etc.

- Probabilistic programs:
  - Every runtime has a probability; their runtime is a distribution
  - Runtimes of “probability-one” terminating programs may not sum up
    
    \[ \text{if } P \text{ and } Q \text{ terminate in } n \text{ and } k \text{ steps on average, then } P;Q \text{ may need infinitely many steps on average} \]
  - Analysis techniques: involve reasoning about expected values etc.
This EWSCS 2020 tutorial

- The probabilistic guarded command language pGCL
  - examples, syntax, semantics (Markov chains), conditioning, recursion

- Proving correctness of probabilistic programs
  - weakest pre-conditions, loop invariants, post-conditions, conditioning

- Almost-sure termination
  - positive a.s.-termination, (a bit of) hardness, stochastic ranking functions

- Analysing runtimes of probabilistic programs
  - examples, finite versus infinite expected runtime, wp-reasoning

- Verifying and runtime analysis of Bayesian networks
Overview

1. What is probabilistic programming?
2. Probabilistic Guarded Command Language
3. Weakest preconditions
4. Conditioning
5. Expected runtime analysis
6. Analysing Bayesian networks
Overview

1. What is probabilistic programming?
2. Probabilistic Guarded Command Language
3. Weakest preconditions
4. Conditioning
5. Expected runtime analysis
6. Analysing Bayesian networks
Dijkstra’s guarded command language

- skip
- diverge
- $x := E$
- $\text{prog1} ; \text{prog2}$
- $\text{if } (G) \text{ prog1 else prog2}$
- $\text{prog1 [] prog2}$
- $\text{while } (G) \text{ prog}$

empty statement
divergence
assignment
sequential composition
choice
non-deterministic choice
iteration
Probabilistic GCL

- skip
- diverge
- \( x := E \)
- observe \((G)\)
- \( prog_1 ; prog_2 \)
- if \((G)\) prog1 else prog2
- \( prog_1 [p] prog_2 \)
- while \((G)\) prog

\( p = \frac{1}{2} \)

\( P = \frac{1}{x+y} \)

- empty statement
- divergence
- assignment
- conditioning
- sequential composition
- choice
- probabilistic choice
- iteration
Let’s start simple

\[
\begin{align*}
x & := 0 \quad [0.5] \quad x := 1; \\
y & := -1 \quad [0.5] \quad y := 0
\end{align*}
\]

This program admits four runs and yields the outcome:

\[
\begin{align*}
Pr[x=0, y=0] & = Pr[x=0, y=-1] = Pr[x=1, y=0] = Pr[x=1, y=-1] = \frac{1}{4}
\end{align*}
\]
A loopy program

For $0 < p < 1$ an arbitrary probability:

```
bool c := true;
int i := 0;
while (c) {
    i++;
    (c := false [p] c := true)
}
```

The loopy program models a geometric distribution with parameter $p$.

$$Pr[i = N] = (1-p)^{N-1} \cdot p$$ for $N > 0$
On termination

```c
bool c := true;
int i := 0;
while (c) {
    i++;
    (c := false [p] c := true)
}
```

This program does **not always** terminate. It **almost surely** terminates.
Conditioning

\[ P(A|B) = \frac{P(B|A)P(A)}{P(B)} \]
Let’s start simple

\[
x := 0 \ [0.5] \ x := 1;
y := -1 \ [0.5] \ y := 0;
\text{observe} \ (x+y = 0)
\]

This program blocks two runs as they violate \( x+y = 0 \). Outcome:

\[
Pr[x=0, y=0] = Pr[x=1, y=-1] = \frac{1}{2}
\]
Let’s start simple

\[
\begin{align*}
\text{x := 0 [0.5]} & \quad \text{x := 1}; \\
\text{y := -1 [0.5]} & \quad \text{y := 0}; \\
\text{observe (x+y = 0)} & \\
\end{align*}
\]

This program blocks two runs as they violate \(x+y = 0\). Outcome:

\[
Pr[x=0, y=0] = Pr[x=1, y=-1] = \frac{1}{2}
\]

Observations thus normalize the probability of the “feasible” program runs
A loopy program

For $0 < p < 1$ an arbitrary probability:

```plaintext
bool c := true;
int i := 0;
while (c) {
    i++;
    (c := false [p] c := true)
}
observe (odd(i))
```

The feasible program runs have a probability \(8N_0(1-p)^2Np^2=1/2p\) for \(N_0 \neq 0\).
A loopy program

For $0 < p < 1$ an arbitrary probability:


```plaintext
bool c := true;
int i := 0;
while (c) {
    i++;
    (c := false [p] c := true)
}
observe (odd(i))
```

The feasible program runs have a probability $\sum_{N \geq 0} (1-p)^{2N} \cdot p = \frac{1}{2-p}$

This program models the distribution:

$Pr[i = 2N+1] = (1-p)^{2N} \cdot p \cdot (2-p)$ for $N \geq 0$

$Pr[i = 2N] = 0$
Why formal semantics matters

- Unambiguous meaning to (almost) all probabilistic programs
- Operational interpretation to weakest pre-expectations
- Basis for proving correctness
  - of programs
  - of program transformations
  - of program equivalence
  - of static analysis
  - of compilers
  - .........
Andrei Andrejewitsch Markow
Markov chains

A Markov chain (MC) is a triple \((\Sigma, \sigma_I, P)\) with:

- \(\Sigma\) being a countable set of states
- \(\sigma_I \in \Sigma\) the initial state, and
- \(P : \Sigma \to \text{Dist}(\Sigma)\) the transition probability function

where \(\text{Dist}(\Sigma)\) is a discrete probability measure on \(\Sigma\).

\[
P(2, \cdot) = \mu \quad \text{with} \quad \mu(2) = \frac{1}{3} \quad \mu(1) = \frac{1}{3} \quad \mu(3) = \frac{1}{3} \quad \mu(s) = 0 \quad s \neq 1, 2, 3
\]
Operational semantics

**Aim:** Model the behaviour of a program \( P \) by the MC \( \llbracket P \rrbracket \).
**Operational semantics**

**Aim:** Model the behaviour of a program $P$ by the MC $\llbracket P \rrbracket$.

This can be defined using Plotkin’s SOS-style semantics.
Operational semantics

Aim: Model the behaviour of a program $P$ by the MC $\llbracket P \rrbracket$.

Approach:
- Take states of the form $Z \triangleleft Q$, $s \triangleright$ with program $Q$ or $\downarrow$, and variable valuation $s : Var \rightarrow Q$
- $\langle \mathcal{E} \rangle$ models the violation of an observation, and
- $\langle sink \rangle$ models program termination (successful or violated observation)
- Take initial state $\langle P, s \rangle$ where $s$ fulfils the initial conditions
- Take transition relation $\rightarrow$ as smallest relation satisfying the SOS rules
Some SOS rules

\[
\langle \text{skip}, s \rangle \rightarrow \langle \downarrow, s \rangle \\
\langle \text{diverge}, s \rangle \rightarrow \langle \text{diverge}, s \rangle
\]
Some SOS rules

\[
\langle \text{skip}, s \rangle \rightarrow \langle \downarrow, s \rangle \quad \langle \text{diverge}, s \rangle \rightarrow \langle \text{diverge}, s \rangle
\]

\[
\frac{s \models G}{\langle \text{observe}(G), s \rangle \rightarrow \langle \downarrow, s \rangle}
\]

\[
\frac{s \not\models G}{\langle \text{observe}(G), s \rangle \rightarrow \langle \uparrow \rangle}
\]
Some SOS rules

\[
\langle \text{skip}, s \rangle \rightarrow \langle \downarrow, s \rangle \quad \langle \text{diverge}, s \rangle \rightarrow \langle \text{diverge}, s \rangle
\]

\[
\begin{align*}
\frac{s \models G}{\langle \text{observe}(G), s \rangle \rightarrow \langle \downarrow, s \rangle} & \quad \frac{s \not\models G}{\langle \text{observe}(G), s \rangle \rightarrow \langle \uparrow \rangle} \\
\langle \downarrow, s \rangle \rightarrow \langle \text{sink} \rangle & \quad \langle \uparrow \rangle \rightarrow \langle \text{sink} \rangle & \quad \langle \text{sink} \rangle \rightarrow \langle \text{sink} \rangle
\end{align*}
\]
Some SOS rules

\[
\begin{align*}
\langle \text{skip}, s \rangle & \rightarrow \langle \downarrow, s \rangle & \langle \text{diverge}, s \rangle & \rightarrow \langle \text{diverge}, s \rangle \\
\frac{s \models G}{\langle \text{observe}(G), s \rangle \rightarrow \langle \downarrow, s \rangle} & & \frac{s \not\models G}{\langle \text{observe}(G), s \rangle \rightarrow \langle \nabla \rangle} \\
\langle \downarrow, s \rangle & \rightarrow \langle \text{sink} \rangle & \langle \nabla \rangle & \rightarrow \langle \text{sink} \rangle & \langle \text{sink} \rangle & \rightarrow \langle \text{sink} \rangle \\
\langle x := E, s \rangle & \rightarrow \langle \downarrow, s[x := s(\lfloor E \rfloor)] \rangle
\end{align*}
\]
Some SOS rules

\[
\begin{align*}
\langle \text{skip}, s \rangle &\rightarrow \langle \downarrow, s \rangle \\
\langle \text{diverge}, s \rangle &\rightarrow \langle \text{diverge}, s \rangle
\end{align*}
\]

\[
\begin{align*}
\text{If } s &\models G \\
\langle \text{observe}(G), s \rangle &\rightarrow \langle \downarrow, s \rangle
\end{align*}
\]

\[
\begin{align*}
\text{If } s &\not\models G \\
\langle \text{observe}(G), s \rangle &\rightarrow \langle \uparrow \rangle
\end{align*}
\]

\[
\begin{align*}
\langle \downarrow, s \rangle &\rightarrow \langle \text{sink} \rangle \\
\langle \uparrow \rangle &\rightarrow \langle \text{sink} \rangle \\
\langle \text{sink} \rangle &\rightarrow \langle \text{sink} \rangle
\end{align*}
\]

\[
\langle x := E, s \rangle \rightarrow \langle \downarrow, s[x := s(\llbracket E \rrbracket)] \rangle
\]

\[
\langle P[\ p\ ] Q, s \rangle \rightarrow \mu \text{ with } \mu(\langle P, s \rangle) = p \text{ and } \mu(\langle Q, s \rangle) = 1 - p
\]

\[
\begin{tikzpicture}
  \node (o) at (0,0) [circle,draw] {};
  \draw[->,red] (o) -- (0,-1);
  \draw[->,red] (o) -- (0,1);
  \node at (-0.5,0) {$p$};
  \node at (0.5,0) {$1-p$};
\end{tikzpicture}
\]
Some SOS rules

\[
\langle \text{skip}, s \rangle \rightarrow \langle \downarrow, s \rangle \quad \langle \text{diverge}, s \rangle \rightarrow \langle \text{diverge}, s \rangle
\]

\[
\frac{s \vdash G}{\langle \text{observe}(G), s \rangle \rightarrow \langle \downarrow, s \rangle} \quad \frac{s \not\vdash G}{\langle \text{observe}(G), s \rangle \rightarrow \langle \uparrow \rangle}
\]

\[
\langle \downarrow, s \rangle \rightarrow \langle \text{sink} \rangle \quad \langle \uparrow \rangle \rightarrow \langle \text{sink} \rangle \quad \langle \text{sink} \rangle \rightarrow \langle \text{sink} \rangle
\]

\[
\langle x := E, s \rangle \rightarrow \langle \downarrow, s[x := s(\llbracket E \rrbracket)] \rangle
\]

\[
\langle P[ p ] Q, s \rangle \rightarrow \mu \text{ with } \mu(\langle P, s \rangle) = p \text{ and } \mu(\langle Q, s \rangle) = 1 - p
\]

\[
\frac{\langle P, s \rangle \rightarrow \langle \uparrow \rangle}{\langle P; Q, s \rangle \rightarrow \langle \uparrow \rangle}
\]

\[
\frac{\langle P, s \rangle \rightarrow \mu}{\langle P; Q, s \rangle \rightarrow \nu} \quad \text{with } \nu(\langle P'; Q', s' \rangle) = \mu(\langle P', s' \rangle) \text{ where } \downarrow; Q = Q
\]
Some SOS rules

\[ \langle \text{skip}, s \rangle \rightarrow \langle \downarrow, s \rangle \quad \langle \text{diverge}, s \rangle \rightarrow \langle \text{diverge}, s \rangle \]

\[
\begin{align*}
\frac{s \not\models G}{\langle \text{observe}(G), s \rangle \rightarrow \langle \downarrow, s \rangle} & \quad \frac{s \not\models G}{\langle \text{observe}(G), s \rangle \rightarrow \langle \downarrow, s \rangle} \\
\langle \downarrow, s \rangle \rightarrow \langle \text{sink} \rangle & \quad \langle \downarrow, s \rangle \rightarrow \langle \downarrow, s \rangle \\
\langle \uparrow, s \rangle \rightarrow \langle \text{sink} \rangle & \quad \langle \uparrow, s \rangle \rightarrow \langle \text{sink} \rangle \\
\langle \text{sink} \rangle \rightarrow \langle \text{sink} \rangle
\end{align*}
\]

\[ \langle x := E, s \rangle \rightarrow \langle \downarrow, s[x := s(\llbracket E \rrbracket)] \rangle \]

\[ \langle P[ p] Q, s \rangle \rightarrow \mu \text{ with } \mu(\langle P, s \rangle) = p \text{ and } \mu(\langle Q, s \rangle) = 1 - p \]

\[
\begin{align*}
\frac{\langle P, s \rangle \rightarrow \langle \uparrow \rangle}{\langle P; Q, s \rangle \rightarrow \langle \uparrow \rangle} & \quad \frac{\langle P, s \rangle \rightarrow \mu}{\langle P; Q, s \rangle \rightarrow \nu} \\
\text{with } \nu(\langle P'; Q, s' \rangle) = \mu(\langle P', s' \rangle) \text{ where } \downarrow; Q = Q \\
s \models G & \quad s \not\models G
\end{align*}
\]

\[ \langle \text{while}(G)\{P\}, s \rangle \rightarrow \langle P; \text{while}(G)\{P\}, s \rangle \]

\[ \langle \text{while}(G)\{P\}, s \rangle \rightarrow \langle \downarrow, s \rangle \]
The piranha problem

[ Tijms, 2004 ]

One fish is contained within the confines of an opaque fishbowl. The fish is equally likely to be a piranha or a goldfish. A sushi lover throws a piranha into the fish bowl alongside the other fish. Then, immediately, before either fish can devour the other, one of the fish is blindly removed from the fishbowl. The fish that has been removed from the bowl turns out to be a piranha. What is the probability that the fish that was originally in the bowl by itself was a piranha?
The operational semantics

\begin{verbatim}
f1 := gf [0.5] f1 := pir;
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
\end{verbatim}
The operational semantics

\[
f_1 := \text{gf } [0.5] \ f1 := \text{pir}
\]
\[
f_2 := \text{pir};
\]
\[
s := f1 [0.5] \ s := f2;
\]
\[
\text{observe } (s = \text{pir})
\]
The good, the bad, and the ugly
Example operational semantics

```c
int cowboyDuel(float a, b) {
    int t := A [0.5] t := B;
    bool c := true;
    while (c) {
        if (t = A) {
            (c := false [a] t := B);
        } else {
            (c := false [b] t := A);
        }
    }
    return t;
}
```
Example operational semantics

```cpp
int cowboyDuel(float a, b) {
    int t := A [0.5] t := B;
    bool c := true;
    while (c) {
        if (t = A) {
            (c := false [a] t := B);
        } else {
            (c := false [b] t := A);
        }
    }
    return t;
}
```
Example operational semantics

```c
int cowboyDuel(float a, b) {
    int t := A [0.5] t := B;
    bool c := true;
    while (c) {
        if (t = A) {
            (c := false [a] t := B);
        } else {
            (c := false [b] t := A);
        }
    }
    return t;
}
```

This (parametric) MC is finite. Once we count the number of shots before one of the cowboys dies, the MC becomes countably infinite.
Duelling cowboys

\[ \omega_A = a + (1-a)(1-b) \cdot \omega_A \]

```
int cowboyDuel(float a, b) {
    // 0 < a < 1, 0 < b < 1
    int t := A \[1\] t := B;  // decide who shoots first
    bool c := true;
    while (c) {
        if (t = A) {
            (c := false \[a\] t := B);  // A shoots B with prob. a
        } else {
            (c := false \[b\] t := A);  // B shoots A with prob. b
        }
    }
    return t;  // the survivor
}
```

Claim:

Cowboy A wins the duel with probability \( \frac{a}{a+b-a\cdot b} \).
The piranha puzzle

\[
f_1 := \mathrm{gf} [0.5] f_1 := p \\
f_2 := \text{pir}; \\
s := f_1 [0.5] s := f_2; \\
\text{observe} (s = \text{pir})
\]

What is the probability that the original fish in the bowl was a piranha?
The piranha puzzle

```plaintext
f1 := gf [0.5] f1 := p
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
```

Equip the Markov chain with rewards. Consider expected rewards.
Rewards

Reasoning about expectations using the operational semantics: use rewards.

**MC with rewards**

An MC with rewards is a pair \((M, r)\) with \(M\) an MC with state space \(\Sigma\) and \(r : \Sigma \rightarrow \mathbb{R}\) a function assigning a real reward to each state. The reward \(r(\sigma)\) stands for the reward earned on leaving state \(\sigma\).
Rewards

Reasoning about expectations using the operational semantics: use rewards.

**MC with rewards**

An MC with rewards is a pair \((M, r)\) with \(M\) an MC with state space \(\Sigma\) and \(r: \Sigma \rightarrow \mathbb{R}\) a function assigning a real reward to each state.

The reward \(r(\sigma)\) stands for the reward earned on leaving state \(\sigma\).

**Cumulative reward for reachability**

Let \(\pi = \sigma_0 \ldots \sigma_n\) be a finite path in \((M, r)\) and \(T \subseteq \Sigma\) a set of target states. The cumulative reward along \(\pi\) until reaching \(T\) is for \(\pi \models \Diamond T\):

\[
    r_T(\pi) = r(\sigma_0) + \ldots + r(\sigma_{k-1}) \quad \text{where} \quad \sigma_i \notin T \quad \text{for all} \quad i < k \quad \text{and} \quad \sigma_k \in T.
\]

If \(\pi \not\models \Diamond T\), then \(r_T(\pi) = \infty\).
Expected reward reachability

Expected reward for reachability

The expected reward until reaching $T \subseteq \Sigma$ from $\sigma \in \Sigma$ is:

$$ER^M(\sigma, \Diamond T) = \sum_{\pi \models \Diamond T} Pr^M(\hat{\pi}) \cdot r_T(\hat{\pi})$$

where $\hat{\pi} = \sigma_0 \ldots \sigma_k$ is the shortest prefix of $\pi$ such that $\sigma_k \in T$ and $\sigma_0 = \sigma$. 
Expected reward reachability

**Expected reward for reachability**

The expected reward until reaching $T \subseteq \Sigma$ from $\sigma \in \Sigma$ is:

$$ER^M(\sigma, \Diamond T) = \sum_{\hat{\pi} \in \Diamond T} Pr^M(\hat{\pi}) \cdot r_T(\hat{\pi})$$

where $\hat{\pi} = \sigma_0 \ldots \sigma_k$ is the shortest prefix of $\pi$ such that $\sigma_k \in T$ and $\sigma_0 = \sigma$.

**Conditional expected reward**

Let

$$ER^M(\sigma, \Diamond T \mid \neg \Diamond F) = \frac{ER^M(\sigma, \Diamond T \land \neg \Diamond F)}{1 - Pr(\Diamond F)}$$

be the conditional expected reward until reaching $T$ while avoiding $F$. 
The piranha puzzle

\[
f_1 := \text{gf} \ [0.5] \ f_1 := \text{pir}; \\
\]

\[
f_2 := \text{pir}; \\
\]

\[
s := f_1 \ [0.5] \ s := f_2; \\
\]

\[
\text{observe} \ (s = \text{pir})
\]

What is the probability that the original fish in the bowl was a piranha?
The piranha puzzle

```
f1 := gf [0.5] f1 := pir;
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
```

What is the probability that the original fish in the bowl was a piranha?

Consider the expected reward of successful termination without violating any observation.
The piranha puzzle

\[
f_1 := gf [0.5] f_1 := pir; \\
f_2 := pir; \\
s := f_1 [0.5] s := f_2; \\
\text{observe} \ (s = pir)
\]

What is the probability that the original fish in the bowl was a piranha?

Consider the expected reward of successful termination without violating any observation

\[
\mathbb{E}R^P_\mathcal{I} (\sigma_I, \Diamond (sink) \mid \neg \Diamond (\mathcal{I})) = \frac{1 \cdot 1/2 + 0 \cdot 1/4}{1 - 1/4} = \frac{1/2}{3/4} = \frac{2}{3}.
\]
On computing expected rewards

Expected rewards in finite Markov chains can be computed in polynomial time by solving a system of linear equations.

The same holds for conditional expected rewards.
Recursion: pushdown Markov chains

\[ P ::= \{ \text{skip} \}^{1/2} \{ \text{call } P \}^3 \{ \text{call } P \}^4 \{ \text{call } P \}^5 \]

\[ t_P = \frac{1}{2} t_P + \frac{1}{2} t_{P^3} \quad \text{// termination probability} \]
Take-home messages

Probabilistic programs:

- extend the expressive power of probabilistic graphical models
- are claimed to have many applications and great potential
- are classical programs with random assignment and conditioning
- have an operational semantics as (countably infinite) Markov chains
- recursive programs give rise to push-down Markov chains

Next lecture: how to prove properties of probabilistic programs?