Principles of Probabilistic Programming
Lectures at EWSCS 2020 Winter School

Joost-Pieter Katoen

EWSCS 2020, Palmse, Estonia
Overview

1. Expected runtime analysis

2. Analysing Bayesian networks

3. Epilogue
Nuances of termination

...... certain termination

...... termination with probability one

\[ \implies \text{almost-sure termination} \]

...... in an expected finite number of steps

\[ \implies \text{“positive” almost-sure termination} \]

...... in an expected infinite number of steps

\[ \implies \text{“null” almost-sure termination} \]
This lecture

A weakest-precondition technique for proving “positive” almost-sure termination.

In fact:

A weakest-precondition technique for determining the expected runtime of a probabilistic program.
Expected run-time analysis

- **What?** AST+ termination in finite expected time

- **Generalise. How?**
  - Provide a weakest precondition calculus
  - . . . . . . for expected run-times
  - a compositional calculus to reason at program syntax level

- **Why?**
  - Classical weakest-preconditions cannot be used
  - Proving positive AST is a special instance
  - Reason about the efficiency of randomised algorithms
  - Reason about simulative inference of Bayesian networks
The run time of a probabilistic program is random

```plaintext
int i := 0;
repeat {i++; (c := false [1/2] c := true)}
until (c)
```

The expected runtime is \( 1 + 3\cdot\frac{1}{2} + 5\cdot\frac{1}{4} + \ldots + (2n+1)\cdot\frac{1}{2^n} = \ldots < \infty \).
Hurdles in runtime analysis

1. Programs may diverge while having a finite expected runtime:

   ```
   while (x > 0) { x-- [1/2] skip }
   ```

2. Expected runtimes are extremely sensitive to variations in probabilities

   ```
   while (x > 0) { x-- [1/2+e] x++ } // 0 <= e <= 1/2
   ```

   ▶ For e=0, the expected runtime is infinite.
   ▶ For any e > 0, the expected runtime is finite.

3. Having a finite expected time is not compositional (cf. next slide)
PAST is not compositional

Consider the two probabilistic programs:

```c
int x := 1;
bool c := true;
while (c) {
    c := false [0.5] c := true;
    x := 2*x
}
```

Finite expected termination time
PAST is not compositional

Consider the two probabilistic programs:

\[
\begin{aligned}
\text{PAST} & \quad \text{PAST} \\
\text{while } (c) \{ & \quad \text{while } (x > 0) \{ \\
\quad c := \text{false} \ [0.5] \ c := \text{true}; & \quad \text{x--} \\
\quad x := 2\times x & \}
\end{aligned}
\]

Finite expected termination time

\[
\begin{aligned}
\text{E}(x) = \sum_{k=0}^{\infty} \frac{1}{2^k} \\
\Pr \{ k \text{ iterations} \} = \frac{1}{2^k}
\end{aligned}
\]
PAST is not compositional

Consider the two probabilistic programs:

```c
int x := 1;
bool c := true;
while (c) {
    c := false [0.5] c := true;
    x := 2*x
}
```

Finite expected termination time

```c
while (x > 0) {
    x--
}
```

Finite termination time

Running the right after the left program
yields an infinite expected termination time
Using wp for expected runtimes?

\[
\text{while(true) \{ x++ \}}
\]

- Consider the post-expectation \( x \)
- Characteristic function \( \Phi_x(X) = X(x \mapsto x + 1) \)
- Candidate upper bound is \( I = 0 \)
- Induction: \( \Phi_x(I) = 0(x := x + 1) = 0 = I \leq I \)

We — wrongly — conclude that \( 0 \) is the runtime.

Using weakest pre-expectations is unsound for expected run-time analysis.
Expected run-times

- Expected run-time of program $P$ on input $s$:

$$\sum_{k=1}^{\infty} k \cdot Pr(\text{"P terminates after } k \text{ steps on input } s\text{"})$$

- In general, $ert()$ is a function $t : \Sigma \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$

- Let’s call this a run-time. Let $\mathbb{T}$ denote the set of run-times.

- Complete partial order on $\mathbb{T}$:

$$t_1 \leq t_2 \text{ iff } \forall s \in \Sigma. \ t_1(s) \leq t_2(s)$$
Continuation passing

Same principle as for weakest pre-conditions: reason backwards
### Expected runtime transformer

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics $ert(P, t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>$1 + t$</td>
</tr>
<tr>
<td>diverge</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$x := E$</td>
<td>$1 + t[x := E]$</td>
</tr>
<tr>
<td>$P_1 ; P_2$</td>
<td>$ert(P_1, ert(P_2, t))$</td>
</tr>
<tr>
<td>if (G)$P_1$ else $P_2$</td>
<td>$1 + [G] \cdot ert(P_1, t) + [\neg G] \cdot ert(P_2, t)$</td>
</tr>
<tr>
<td>$P_1 [p] P_2$</td>
<td>$1 + p \cdot ert(P_1, t) + (1-p) \cdot ert(P_2, t)$</td>
</tr>
<tr>
<td>while(G)$P$</td>
<td>lfp $X. 1 + ([G] \cdot ert(P, X) + [\neg G] \cdot t)$</td>
</tr>
</tbody>
</table>

---

Joost-Pieter Katoen

Principles of Probabilistic Programming
Expected runtime transformer

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics $ert(P, t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="skip" alt="skip" /></td>
<td>$1 + t$</td>
</tr>
<tr>
<td><img src="diverge" alt="diverge" /></td>
<td>$\infty$</td>
</tr>
<tr>
<td>$x := E$</td>
<td>$1 + t[x := E]$</td>
</tr>
<tr>
<td>$P_1 ; P_2$</td>
<td>$1 + ert(P_1, ert(P_2, t))$</td>
</tr>
<tr>
<td>$\text{if} \ (G) P_1 \ \text{else} \ P_2$</td>
<td>$1 + [G] \cdot ert(P_1, t) + [\neg G] \cdot ert(P_2, t)$</td>
</tr>
<tr>
<td>$P_1 \ [p] \ P_2$</td>
<td>$1 + p \cdot ert(P_1, t) + (1 - p) \cdot ert(P_2, t)$</td>
</tr>
<tr>
<td>$\text{while}(G) P$</td>
<td>$\text{lfp} \ X \ 1 + ([G] \cdot ert(P, X) + [\neg G] \cdot t)$</td>
</tr>
</tbody>
</table>

$Lfp$ is the least fixed point operator wrt. the ordering $\leq$ on run-times

This simple mild twist of weakest pre-expectations is sound.
Example: straight-line code
if \((y \neq 0)\) {
\[
\begin{align*}
&\{ x := x \cdot y \} \ [1/2] \ \{ y := 3 \} \\
\text{else } &\begin{align*}
&\{ x := 1 \} \ [1/3] \ \{ x := 3 \} \\
&\frac{2}{3}(1 + g(x \rightarrow \gamma)) \\
&\begin{align*}
&\left\lceil x \geq \gamma \right\rceil \cdot 4 + \left\lfloor x < \gamma \right\rfloor \cdot 8 = g
\end{align*}
\end{align*}
\}
\]
} else {
\[
\begin{align*}
\text{skip; } &\begin{align*}
&\{ y := 2 \} \ [1/2] \ \{ x := 3 \}
\end{align*}
\]
} else {
\[
\begin{align*}
&\text{skip; } \\
&x := 3; \\
&\text{skip; } \\
&\{ y := x + 1 \} \ [1/4] \ \{ y := 2 \cdot y \}; \\
&\text{skip } \end{align*}
\]
\}
\]
\[
\begin{align*}
&\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1 = 1 + 1 = 2 \\
&\{ x := x^2 \} \ [1/2] \ \{ y := y^3 \}
\end{align*}
\]
A simple rule for upper bounds

We have \( \text{ert}(\text{while}(G) \, P, t) = \text{lfp } X. \Phi_t(X) \)

with

\[
\Phi_t(X) = 1 + ([G] \cdot \text{ert}(P, X) + [\neg G] \cdot t)
\]

By Park’s lemma:

if \( \Phi_t(l) \leq l \) then \( \text{ert}(\text{while}(G) \, P, t) \leq l \).
Expected runtime analysis

Induction on loops

```java
while (c) {
    (x++ [1/2] c := false)
}
```

- Post runtime equals 0

- Characteristic function:
  \[
  \Phi_0(X) = 1 + [c=1] \cdot \left(2 + \frac{1}{2} \cdot (X(x \mapsto x+1) + X(c \mapsto 0))\right)
  \]

- Candidate for upper bound: \( I = 1 + [c=1] \cdot 6 \)

- Induction: \( \Phi_0(I) = 1 + [c=1] \cdot \left(2 + \frac{1}{2} \cdot (1 + [c=1] \cdot 6 + 1 + [0=1] \cdot 6)\right) \)
  
  \[
  = 1 + [c=1] \cdot 6 \leq I
  \]

By Park’s lemma: \( e \text{rt}(\text{while} \ldots) \leq 1 + [c=1] \cdot 6 \)
Coupon collector’s problem

ON A CLASSICAL PROBLEM OF PROBABILITY THEORY

by

P. ERDŐS and A. RÉNYI

From Wikipedia, the free encyclopedia

In probability theory, the coupon collector’s problem describes the "collect all coupons and win" contests. It asks the following question: Suppose that there is an urn of $n$ different coupons, from which coupons are being collected, equally likely, with replacement. What is the probability that more than $i$ sample trials are needed to collect all $n$ coupons? An alternative statement is: Given $n$ coupons, how many coupons do you expect you need to draw with replacement before having drawn each coupon at least once? The mathematical analysis of the problem reveals that the expected number of trials needed grows as $\Theta(n \log(n))$. For example, when $n = 100$, the expected number of trials to collect all 100 coupons is about 2252.
Coupon collector’s problem

\[
\text{cp} := [0, \ldots, 0]; \quad \text{// no coupons yet}
\]

\[
i := 1; \quad \text{// coupon to be collected next}
\]

\[
x := 0; \quad \text{// number of coupons collected}
\]

\[
\text{while } (x < N) \{
\quad \text{while } (\text{cp}[i] \neq 0) \{
\quad \quad i := \text{uniform}(1..N) \quad \text{// next coupon}
\quad \}
\quad \text{cp}[i] := 1; \quad \text{// coupon i obtained}
\quad x++; \quad \text{// one coupon less to go}
\}
\]

Using our ert-calculus one can prove that expected runtime is \( \Theta(N \cdot \log N) \).
By systematic formal verification à la Floyd-Hoare. Machine checkable.
Elementary properties of the ert-calculus

- **Continuity:** $ert(P, t)$ is continuous, that is
  
  for every chain $T = t_0 \leq t_1 \leq t_2 \leq \ldots$:
  
  $ert(P, \sup T) = \sup ert(P, T)$

- **Monotonicity:** $t \leq t'$ implies $ert(P, t) \leq ert(P, t')$

- **Constant propagation:**
  
  $ert(P, k + t) = k + ert(P, t)$

- **Preservation of $\infty$:**
  
  $ert(P, \infty) = \infty$

- **Connection to wp:**
  
  $ert(P, t) = ert(P, 0) + wp(P, t)$

- **Affinity:**
  
  $ert(P, r \cdot t + u) = ert(P, 0) + r \cdot wp(P, t) + wp(P, u)$
(Positive) almost-sure termination

For every pGCL program $P$ and input state $s$:

\[ \text{ert}(P, 0)(s) < \infty \]

implies

\[ \text{wp}(P, 1)(s) = 1 \]

positive a.s-termination on $s$

almost-sure termination on $s$

Moreover:

\[ \text{ert}(P, 0) < \infty \]

implies

\[ \text{wp}(P, 1) = 1 \]

universal positive a.s-termination

universal almost-sure termination

These (well-known) facts can be proven using a short proof using the elementary properties.
\[
\text{proof: assume } \text{ext}(p, 0)(s) \geq 8 \Rightarrow \text{ext}(p, 1)(s) = \text{ext}(p, 0)(s) + 1
\]

\[
\text{const-propagation}
\]

\[
\text{relationship}
\]

\[
\text{ext and \wp}
\]

\[
\text{calculate}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]

\[
\text{iff}
\]
Obtaining lower bounds inductively

Let $n$ be a natural and let $\text{while}(G) \ P$ be our loop.

Runtime transformer $I_n$ is a lower $\omega$-invariant w.r.t. $t$ iff

$$I_0 \leq \Phi_t(0) \quad \text{and} \quad I_{n+1} \leq \Phi_t(I_n) \quad \text{for all } n$$

Recall: $\Phi_t(X) = 1 + ([G] \cdot \text{ert}(P, X) + [\neg G] \cdot t)$. 
Lower bounds

If $l_n$ is a lower $\omega$-invariant w.r.t. $t$ and $\lim_{n \to \infty} l_n$ exists, then:

$$\lim_{n \to \infty} l_n \leq ert(\text{while}(G) P, t)$$

Completeness: such lower $\omega$-invariants always exist.
PAST is not compositional

Consider the two probabilistic programs:

```
int x := 1;
bool c := true;
while (c) {
    c := false [0.5] c := true;
    x := 2*x
}
```

```
while (x > 0) {
    x--; 
}
```

Finite expected termination time

Finite termination time

Running the right after the left program yields an infinite expected termination time
Proving that PAST is not compositional (1)

```
while (x > 0) { x := x-1 }
```

It is easy to check that a lower $\omega$-invariant is:

\[
J_n = 1 + [0 < x < n] \cdot 2x + [x \geq n] \cdot (2n-1)
\]

on iteration

\[
1 + [x > 0] \cdot 2x
\]

on termination

Thus we obtain that:

\[
\lim_{n \to \infty} (1 + [0 < x < n] \cdot 2x + [x \geq n] \cdot (2n-1)) = 1 + [x > 0] \cdot 2x
\]

is a lower bound on the runtime of the above program.
Proving that PAST is not compositional (2)

\[
\text{while } (c) \{ \{ c := \text{false} \ [0.5] \ c := \text{true} \}; \ x := 2^*x \}; \\
\text{while } (x > 0) \{ \ x := x-1 \ }
\]

Template for a lower $\omega$-invariant of composed program:

\[
I_n = 1 + \left[ c \neq 1 \right] \cdot (1 + [x > 0] \cdot 2x) + \left[ c = 1 \right] \cdot (a_n + b_n \cdot [x > 0] \cdot 2x)
\]

on termination

\[
I_0 \leq \Phi^\omega (I_0) \quad I_n \leq \Phi^\omega (I_n)
\]
Proving that PAST is not compositional (2)

\[
\begin{align*}
\text{while } (c) \{ & \{c := \text{false} \ [0.5] c := \text{true}\}; \ x := 2x\}; \\
\text{while } (x > 0) \{ & x := x-1 \}
\end{align*}
\]

Template for a lower $\omega$-invariant of composed program:

\[
I_n = 1 + [c \neq 1] \cdot (1 + [x > 0] \cdot 2x) + [c = 1] \cdot (a_n + b_n \cdot [x > 0] \cdot 2x)
\]

The constraints on being a lower $\omega$-invariant yield:

\[
\begin{align*}
a_0 & \leq 2 \quad \text{and} \quad a_{n+1} \leq 7/2 + 1/2 \cdot a_n \quad \text{and} \quad b_0 \leq 0 \quad \text{and} \quad b_{n+1} \leq 1 + b_n
\end{align*}
\]

This admits the solution $a_n = 7 - 5/2^n$ and $b_n = n$. Then: $\lim_{n \to \infty} I_n = \infty.$
Some works using the ert-calculus

Certification of ert-calculus in Isabelle/HOL theorem prover.
[Hölzl, ITP 2016]

Automated resource analysis for probabilistic programs.
[Hoffmann et al., PLDI 2018]

Type-based complexity analysis of probabilistic functional programs
[Avanzini, Dal Lago et al., 2019]

Expected run-time analysis of quantum programs
[Liu, Zhou and Ying 2019]
Overview

1. Expected runtime analysis

2. Analysing Bayesian networks

3. Epilogue
The importance of Bayesian networks

“Bayesian networks are as important to AI and machine learning as Boolean circuits are to computer science.”

[Stuart Russell (Univ. of California, Berkeley), 2009]

Key problem: probabilistic inference. This is PP-complete.
How likely is it that your print is garbled given that the ps-file is not and the page orientation is portrait?
How likely does a student end up with a bad mood after getting a bad grade for an easy exam, given that she is well prepared?
Bayesian inference

\[
\Pr(D = 0, G = 0, M = 0 \mid P = 1) = \frac{\Pr(D = 0, G = 0, M = 0, P = 1)}{\Pr(P = 1)}
\]

\[
= \frac{0.6 \cdot 0.5 \cdot 0.9 \cdot 0.3}{0.3} = 0.27
\]
Bayesian inference by program verification

- Exact inference of Bayesian networks is **PP-complete**

- Approximate inference of BNs is **NP-hard**

- Typically *simulative* analyses are employed
  - Rejection Sampling
  - Markov Chain Monte Carlo (MCMC)
  - Metropolis-Hastings
  - Gibbs Sampling
  - Importance Sampling
  - .......

- Here: *weakest precondition-reasoning*
Reasoning about loops

Reasoning about loops is hard. Typically, loop invariants are used to capture the effect of loops. Finding such loop invariants in general is undecidable.

Bayesian networks correspond to “simple” probabilistic programs. Loops in such programs are “data-flow” free. Their effect can be given as closed-form solution.
I.i.d-loops

Loop $\text{while}(G)P$ is iid wrt. expectation $f$ whenever:

both $wp(P,[G])$ and $wp(P,[-G] \cdot f)$ are unaffected by $P$.

$f$ is unaffected by $P$ if none of $f$’s variables are modified by $P$:

$x$ is a variable of $f$ iff $\exists s. \forall v, u : f(s[x = v]) \neq f(s[x = u])$

If $g$ is unaffected by program $P$, then: $wp(P,g \cdot f) = g \cdot wp(P,f)$
Example: sampling within a circle

```plaintext
while ((x-5)**2 + (y-5)**2 >= 25){
    x := uniform(0..10);
    y := uniform(0..10)
}
```

This program is iid for every $f$, as both are unaffected by $P$'s body:

\[
wp(P, [G]) = \frac{48}{121} \quad \text{and} \quad wp(P, [\neg G] \cdot f) = \frac{1}{121} \sum_{i=0}^{10p} \sum_{j=0}^{10p} [(i/p-5)^2 + (j/p-5)^2 < 25] \cdot f(x/(i/p), y/(j/p))
\]
Weakest precondition of iid-loops

If $\text{while}(G)P$ is iid for expectation $f$, it holds for every state $s$:

$$wp(\text{while}(G)P, f)(s) = [G](s) \cdot \frac{wp(P, [\neg G] \cdot f)(s)}{1 - wp(P, [G])(s)} + [\neg G](s) \cdot f(s)$$

where we let $\frac{0}{0} = 0$.

Proof: use $wp(\text{while}_n(G)P, f) = [G] \cdot wp(P, [\neg G] \cdot f) \cdot \sum_{i=0}^{n-2} (wp(P, [G])^i) + [\neg G] \cdot f$

No loop invariant or martingale needed. Fully automatable.
How likely does a student end up with a bad mood after getting a bad grade for an easy exam, given that she is well prepared?
Bayesian networks as programs

- Take a topological sort of the BN’s vertices, e.g., $D; P; G; M$

- Map each conditional probability table (aka: node) to a program, e.g.:

```c
if (xD = 0 && xP = 0) {
    xG := 0 [0.95] xG := 1
} else if (xD = 1 && xP = 1) {
    xG := 0 [0.05] xG := 1
} else if (xD = 0 && xP = 1) {
    xG := 0 [0.5] xG := 1
} else if (xD = 1 && xP = 0) {
    xG := 0 [0.6] xG := 1
}
```

<table>
<thead>
<tr>
<th></th>
<th>$G = 0$</th>
<th>$G = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D = 0, P = 0$</td>
<td>0.95</td>
<td>0.05</td>
</tr>
<tr>
<td>$D = 1, P = 1$</td>
<td>0.05</td>
<td>0.95</td>
</tr>
<tr>
<td>$D = 0, P = 1$</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>$D = 1, P = 0$</td>
<td>0.6</td>
<td>0.4</td>
</tr>
</tbody>
</table>
Bayesian networks as programs

- Take a topological sort of the BN’s vertices, e.g., $D; P; G; M$

- Map each conditional probability table (aka: node) to a program, e.g.:

```java
if (xD = 0 && xP = 0) {
    xG := 0 [0.95] xG := 1
} else if (xD = 1 && xP = 1) {
    xG := 0 [0.05] xG := 1
} else if (xD = 0 && xP = 1) {
    xG := 0 [0.5] xG := 1
} else if (xD = 1 && xP = 0) {
    xG := 0 [0.6] xG := 1
}
```

- Condition on the evidence, e.g., for $P = 1$ we get:

```java
repeat { progD ; progP; progG ; progM } until (xP=1)
```
Soundness

For BN $B$ over $V$ with evidence $obs$ for $O \subseteq V$ and value $\_v$ for node $v$:

$$wp\left(\text{prog}(B, obs), \bigwedge_{v \in V \setminus O} x_v = \_v\right) = Pr\left(\bigwedge_{v \in V \setminus O} v = \_v \bigg| \bigwedge_{o \in O} o = obs(o)\right)$$

where $\text{prog}(B, obs)$ equals $\text{repeat } \text{prog}B \text{ until } (\bigwedge_{o \in O} x_o = obs(o))$.

Thus: $wp$-reasoning of BN-programs equals exact Bayes’ inference

As BN-programs are iid for every $f$, this is fully automatable
Exact inference by wp-reasoning

Ergo: exact Bayesian inference by wp-reasoning:

\[ wp(P_{\text{mood}}, [x_D = 0 \land x_G = 0 \land x_M = 0]) = \frac{Pr(D = 0, G = 0, M = 0, P = 1)}{Pr(P = 1)} = 0.27 \]
How long to sample a BN?

“the main challenge in this setting [sampling-based approaches] is that many samples that are generated during execution are ultimately rejected for not satisfying the observations.”

[Gordon, Nori, Henzinger, Rajamani, 2014]
Rejection sampling

For a given Bayesian network and some evidence:

1. Sample from the joint distribution described by the BN
2. If the sample complies with the evidence, accept the sample and halt
3. If not, repeat sampling (that is: go back to step 1.)

If this procedure is applied $N$ times, $N$ iid-samples result.

Q: How many samples do we need on average for a single iid-sample?
A toy Bayesian network

This BN is **parametric** (in $a$)

How many samples are needed on average for a **single** iid-sample for evidence $G = 0$?
Sampling time for example BN

Rejection sampling for $G = 0$ requires \( \frac{200a^2 - 40a - 460}{89a^2 - 69a - 21} \) samples:

For $a \in [0.1, 0.78]$, EST is below 18; for $a \geq 0.98$, 100 samples are needed.

For real-life BNs, the EST may exceed $10^{15}$. 

\( \infty \)
**Expected runtime of iid-loops**

For a.s.-terminating iid-loop `while(G)P` for which every iteration runs in the same expected time, we have:

\[
ert(\text{while}(G)P, t) = 1 + [G] \cdot \frac{1 + ert(P, [\neg G] \cdot t)}{1 - wp(P, [G])} + [\neg G](s) \cdot t
\]

where \(0/0 := 0\) and \(a/0 := \infty\) for \(a \neq 0\).

**Proof:** similar as for the inference \((wp)\) using the decomposition lemma:

\[
ert(P, t) = ert(P, 0) + wp(P, t)
\]

No loop invariant needed. Fully automatable.
Sample times of BN programs

Every BN-program is iid for every $f$, is almost surely terminating, and every loop-iteration takes on average equally long.

This enables determining the exact expected sampling times of BNs in a fully automated manner.

But: BN-programs may be not positively a.s.-terminating. This holds for ill-conditioned BNs. The evidence(s) in such BNs occur with probability zero.
The student’s mood example

\[ \text{ert} \left( \text{repeat } D; P; G; M \text{ until } (P=1), 0 \right) = \frac{1 + \text{ert}(D; P; G; M, 0)}{wp(D; P; G; M, [P = 1])} \approx 23.46 \]

Program of student mood’s BN

\[
\begin{array}{|c|c|c|}
\hline
D = 0 & D = 1 \\
0.6 & 0.4 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
D = 0, P = 0 & G = 0 & G = 1 \\
0.95 & 0.05 \\
\hline
D = 1, P = 1 & 0.05 & 0.95 \\
\hline
D = 0, P = 1 & 0.5 & 0.5 \\
\hline
D = 1, P = 0 & 0.6 & 0.4 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
P = 0 & P = 1 \\
0.7 & 0.3 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
M = 0 & M = 1 \\
G = 0 & 0.9 & 0.1 \\
G = 1 & 0.3 & 0.7 \\
\hline
\end{array}
\]
Experimental results

Benchmark BNs from www.bnlearn.com

| BN        | $|V|$ | $|E|$ | aMB | $|O|$ | EST          | time (s) |
|-----------|-----|-----|-----|-----|--------------|----------|
| hailfinder | 56  | 66  | 3.54| 5   | $5 \times 10^5$| 0.63     |
| hepar2    | 70  | 123 | 4.51| 1   | $1.5 \times 10^2$| 1.84     |
| win95pts  | 76  | 112 | 5.92| 3   | $4.3 \times 10^5$| 0.36     |
| pathfinder| 135 | 200 | 3.04| 7   | $\infty$     | 5.44     |
| andes     | 223 | 338 | 5.61| 3   | $5.2 \times 10^3$| 1.66     |
| pigs      | 441 | 592 | 3.92| 1   | $2.9 \times 10^3$| 0.74     |
| munin     | 1041| 1397| 3.54| 5   | $\infty$     | 1.43     |

aMB = average Markov Blanket size, a measure of independence in BNs
Analysing Bayesian networks

Printer troubleshooting in Windows 95

Java implementation executes about $10^7$ steps in a single second

For $|O|=17$, an EST of $10^{15}$ yields **3.6 years simulation for a single iid-sample**
Overview

1. Expected runtime analysis

2. Analysing Bayesian networks

3. Epilogue
Predictive probabilistic programming

Analysing probabilistic programs at source code level, compositionally.
Two take-home messages

Probabilistic programs are a **universal quantitative** modeling formalism:
Bayesian networks, randomised algorithms, infinite-state Markov chains,
pushdown Markov chains, security mechanisms, quantum programs,
robotics, programs for inexact computing . . . . .
Two take-home messages

Probabilistic programs are a **universal quantitative** modeling formalism: Bayesian networks, randomised algorithms, infinite-state Markov chains, pushdown Markov chains, security mechanisms, quantum programs, robotics, programs for inexact computing ......

“The **crux** of probabilistic programming is to consider **normal-looking** programs **as** if they were **probability distributions**”

[Michael Hicks, The Programming Language Enthusiast blog, 2014]
A big thanks to my co-authors!

Kevin Batz, Christian Dehnert, Friedrich Gretz, Nils Jansen, Benjamin Kaminski, Christoph Matheja, Annabelle McIver, Larissa Meinecke, Carroll Morgan, Fedrico Olmedo, Lukas Westhofen
Further reading

  *Probabilistic programming.* FOSE 2014.
- Z. Ghahramani. 
- JPK, A. McIver, L. Meinicke, and C. Morgan. 
- F. Gretz, JPK, and A. McIver. 
  *PRINSYS — on a quest for probabilistic loop invariants.* QEST 2013.
- F. Gretz, JPK, and A. McIver. 
- F. Olmedo, F. Gretz, N. Jansen, B. Kaminski, JPK, A. McIver 
  *Conditioning in probabilistic programming.* ACM TOPLAS 2018.

pGCL model checking: [www.stormchecker.org](http://www.stormchecker.org)
Further reading

- B. Kaminski, JPK, C. Matheja. 

- B. Kaminski, JPK.

- B. Kaminski, JPK, C. Matheja, and F. Olmedo.

- F. Olmedo, B. Kaminski, JPK, C. Matheja.
  *Reasoning about recursive probabilistic programs.* LICS 2016.

- A. McIver, C. Morgan, B. Kaminski, JPK.

- K. Batz, B. Kaminski, JPK, C. Matheja.
  *How long, O Bayesian network, will I sample thee?* ESOP 2018.

- N. Jansen, C. Dehnert, B. Kaminski, JPK, L. Westhofen.
  *Bounded model checking for probabilistic programs.* ATVA 2016.