Bayes’ Network Analysis by Program Verification

Joost-Pieter Katoen

Alan Turing Institute, January 2018
There are several reasons why probabilistic programming could prove to be revolutionary for machine intelligence and scientific modelling.”

Why? Probabilistic programming

1. ... obviates the need to manually provide inference methods
2. ... enables rapid prototyping
3. ... clearly separates the model and the inference procedures

1 Ghahramani leads the Cambridge ML Group, and is with CMU, UCL, and Turing Institute.
**Predictive probabilistic programming**

Verifiable programs are preferable to simulative guarantees.

Our take: **reason on program code, compositionally.**
Probabilistic graphical models
Student’s mood after an exam

How likely does a well-prepared student end up with a bad mood after getting a bad grade for an easy exam?
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?

see also https://www.youtube.com/watch?v=PyBHYPkwB-Y
Probabilistic programs

What?

Programs with random assignments and conditioning

Why?

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

Languages: webPPL, ProbLog, R2, Figaro, ....
Roadmap

1. Probabilistic weakest pre-conditions
2. Bayesian inference by program analysis
3. Termination
4. Runtime analysis
5. How long to sample a Bayes’ network?
6. Epilogue
Overview

1. Probabilistic weakest pre-conditions
2. Bayesian inference by program analysis
3. Termination
4. Runtime analysis
5. How long to sample a Bayes’ network?
6. Epilogue
Probabilistic GCL

- skip
- diverge
- \( x := E \)
- observe \((G)\)
- \( prog1; prog2 \)
- \( if \ (G) \ prog1 \ else \ prog2 \)
- \( prog1 \ [p] \ prog2 \)
- \( while \ (G) \ prog \)
Let’s start simple

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

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

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

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

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

```c
bool c := true;
int i := 0;
while (c) {
    i := i + 1;
    (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:

\[
\begin{align*}
Pr[i = 2N+1] &= (1-p)^{2N} \cdot p \cdot (2-p) \quad \text{for } N \geq 0 \\
Pr[i = 2N] &= 0
\end{align*}
\]
Or, equivalently

```c
int i := 0;
repeat {
    c := true;
    i := 0;
    while (c) {
        i := i+1;
        (c := false [p] c := true)
    }
} until (odd(i))
```
Weakest pre-expectations

An expectation\(^2\) maps states onto \(\mathbb{R}_{\geq 0} \cup \{\infty\}\). It is the quantitative analogue of a predicate. Let \(f \leq g\) iff \(f(s) \leq g(s)\), for every state \(s\).

An expectation transformer is a total function between two expectations.

The transformer \(wp(P, f)\) yields the least expectation \(e\) on \(P\)'s initial state ensuring that \(P\) terminates with expectation \(f\).

Annotation \(\{e\} P \{f\}\) holds for total correctness iff \(e \leq wp(P, f)\).

Weakest liberal pre-expectation \(wlp(P, f) = "wp(P, f) + Pr[P \text{ diverges}]"\).

\(^2\) ≠ expectations in probability theory.
## Expectation transformer semantics of pGCL

### Syntax

- `skip`
- `diverge`
- `x := E`
- `observe (G)`
- `P1 ; P2`
- `if (G)P1 else P2`
- `P1 [p] P2`
- `while (G)P`

### Semantics \( wp(P, f) \)

\[
\begin{align*}
  f & \quad \text{if } (G)P1 \text{ else } P2 \\
  0 & \quad \text{while } (G)P \\
  [G] \cdot f & \quad \text{observe } (G) \\
  \text{wp}(P_1, \text{wp}(P_2, f)) & \quad \text{if } (G)P1 \text{ else } P2 \\
  [G] \cdot \text{wp}(P_1, f) + [\neg G] \cdot \text{wp}(P_2, f) & \quad \text{while } (G)P \\
  p \cdot \text{wp}(P_1, f) + (1-p) \cdot \text{wp}(P_2, f) & \quad \text{if } (G)P1 \text{ else } P2 \\
  \mu X. ([G] \cdot \text{wp}(P, X) + [\neg G] \cdot f) & \quad \text{while } (G)P
\end{align*}
\]

\( \mu \) is the least fixed point operator wrt. the ordering \( \leq \).

wlp-semantics differs from wp-semantics only for `while` and `diverge`. 
Examples

1. Let program $P$ be:
   
   $x := 5 \ [4/5] x := 10$

   For $f = x$, we have
   
   $$wp(P, x) = \frac{4}{5} \cdot wp(x := 5, x) + \frac{1}{5} \cdot wp(x := 10, x) = \frac{4}{5} \cdot 5 + \frac{1}{5} \cdot 10 = 6$$

2. Let program $P'$ be:
   
   $x := x+5 \ [4/5] x := 10$

   For $f = x$, we have:
   
   $$wp(P', x) = \frac{4}{5} \cdot wp(x + := 5, x) + \frac{1}{5} \cdot wp(x := 10, x) = \frac{4}{5} \cdot (x+5) + \frac{1}{5} \cdot 10 = \frac{4x}{5} + 6$$

3. For program $P'$ (again) and $f = [x = 10]$, we have:
   
   $$wp(P', [x=10]) = \frac{4}{5} \cdot wp(x := x+5, [x=10]) + \frac{1}{5} \cdot wp(x := 10, [x=10])$$
   
   $$= \frac{4}{5} \cdot [x+5 = 10] + \frac{1}{5} \cdot [10 = 10]$$
   
   $$= \frac{4[x=5]+1}{5}$$
An operational perspective

For program $P$, input $s$ and expectation $f$:

$$\frac{wp(P, f)(s)}{wlp(P, 1)(s)} = \mathbb{E}\{ \text{Rew}[P](s, ◇sink \cap \neg◇⊥) \}$$

The ratio $wp(P, f) / wlp(P, 1)$ for input $s$ equals the conditional expected reward to reach a successful terminal state $sink$ while satisfying all observes in MC $[P]$.

For finite-state programs, wp-reasoning can be done with model checkers such as PRISM and Storm (www.stormchecker.org).

---

$^3$ Either both sides are equal or both sides are undefined.
Overview

1. Probabilistic weakest pre-conditions
2. Bayesian inference by program analysis
3. Termination
4. Runtime analysis
5. How long to sample a Bayes’ network?
6. Epilogue
How likely does a well-prepared student end up with a bad mood after getting a bad grade for an easy exam?
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 **NP-hard**

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

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

- Here: weakest precondition-reasoning
**i.i.d-loops**

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

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

If $g$ is unaffected by program $P$, then: $wp(P, g \cdot f) = g \cdot wp(P, f)$

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

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

---

Joost-Pieter Katoen
Bayes’ Network Analysis by Program Verification
**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:

$$wp(P, [G]) = \frac{48}{121} \quad \text{and}$$

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

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

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

where we let \(\frac{0}{0} = 0\).

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

No loop invariant, martingale, or ranking function needed. Fully automatable.
Bayesian inference

How likely does a well-prepared student end up with a bad mood after getting a bad grade for an easy exam?
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.:

  ```
  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:

  ```
  repeat { progD ; progP; progG ; progM } until (P=1)
  ```
Properties of BN programs

\[
\text{repeat } \{ \text{progD ; progP; progG ; progM } \} \text{ until (P=1)}
\]

1. Every BN-program naturally represents rejection sampling

2. Every BN-program is iid for every expectation \( f \)

3. Every BN-program almost surely terminates

4. A BN-program’s size is linear in the BN’s size
Soundness

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

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

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

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 can be done by wp-reasoning, e.g.,

\[ wp(P_{mood}, [x_D = 0 \land x_G = 0 \land x_M = 0]) = \frac{Pr(D = 0, G = 0, M = 0, P = 1)}{Pr(P = 1)} = 0.27 \]
Overview

1. Probabilistic weakest pre-conditions
2. Bayesian inference by program analysis
3. Termination
4. Runtime analysis
5. How long to sample a Bayes’ network?
6. Epilogue
Termination proofs: the classical case

$V(s^i)$

$V(s^4)$

$V(s^5)$

$V(s^5) < V(s^4)$

arrival at 0 guaranteed by well-foundedness of $>$

→ loop iterations
Termination

"[Ordinary] termination is a purely topological property [...], but almost-sure termination is not. [...] Proving almost–sure termination requires arithmetic reasoning not offered by termination provers."

Proving a.s.-termination for a single input is $\Pi_2$-complete
(the same holds for approximate a.s.-termination)
Almost-sure 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.
Proving almost-sure termination

The symmetric random walk:

\[
\text{while } (x > 0) \{ \ x := x-1 \ [0.5] \ x := x+1 \ \}
\]

Is out-of-reach for many proof rules.

A loop iteration decreases \(x\) by one with probability \(1/2\)

This observation is enough to witness almost-sure termination!
Proving almost-sure termination

Goal: prove a.s.–termination of \( \text{while}(G) \ P \)

Ingredients:

- A supermartingale \( V \) mapping states onto non-negative reals
  - \( V(s_n) \geq \mathbb{E}\{V(s_{n+1}) \mid V(s_0), \ldots, V(s_n)\} \)
  - Running body \( P \) on state \( s \models G \) does not increase \( \mathbb{E}(V(s)) \)
  - Loop iteration ceases if \( V(s) = 0 \)

- ....... and a progress condition: on each loop iteration in \( s^i \)
  - \( V(s^i) = v \) decreases by \( \geq d(v) \) with probability \( \geq p(v) \)
  - with antitone \( p \) ("probability") and \( d \) ("decrease") on \( V \)'s values

Then: \( \text{while}(G) \ P \) a.s.-terminates on every input
Proving almost-sure termination

The closer to termination, the more $V$ decreases and this becomes more likely.

Given:
- $\forall i, V(s^i)$
- $V(s^i) \rightarrow V(s^0)$
- With prob. $\geq p(V(s^1))$
- $d(V(s^1))$ with prob. $\geq p(V(s^4))$
- $d(V(s^4))$ with prob. $\geq p(V(s^5))$
- $d(V(s^5))$ with prob. $\geq p(V(s^8))$
- $d(V(s^8))$ with prob. $\geq p(V(s^9))$
- $d(V(s^9))$ with prob. $\geq p(V(s^0))$

Conclusion:
- A.s. arrival at 0 guaranteed by a sup rule.

Probability Relations:
- $p(V1) \leq p(V4)$ by antitone $p$
- $d(V1) \leq d(V4)$ by antitone $d$
The symmetric random walk

- Recall:

  \[
  \text{while } (x > 0) \{ x := x-1 \ [0.5] x := x+1 \}
  \]

- Witnesses of almost-sure termination:
  - \( V = x \)
  - \( p(v) = \frac{1}{2} \) and \( d(v) = 1 \)

  That’s all you need to prove almost-sure termination!
A symmetric-in-the-limit random walk

Consider the program:

```plaintext
while (x > 0) { p := x/(2*x+1) ; x := x-1 [p] x := x+1 }
```

Witnesses of almost-sure termination:

- \( V = H_x \), where \( H_x \) is \( x \)-th Harmonic number \( 1 + \frac{1}{2} + \ldots + \frac{1}{x} \)

- \( p(v) = \frac{1}{3} \) and \( d(v) = \begin{cases} \frac{1}{x} & \text{if } v > 0 \text{ and } H_{x-1} < v \leq H_x \\ 1 & \text{if } v = 0 \end{cases} \)
Expressiveness

This proof rule covers many a.s.-terminating programs that are out-of-reach for almost all existing proof rules.
Overview

1. Probabilistic weakest pre-conditions
2. Bayesian inference by program analysis
3. Termination
4. Runtime analysis
5. How long to sample a Bayes’ network?
6. Epilogue
Null a.s.-termination

\[ x := 10; \textbf{while} (x > 0) \{ x := x-1 \ [0.5] x := x+1 \} \]

This program \textit{almost surely} terminates but requires an \textit{infinite} expected time to do so.
Positive almost-sure termination

Deciding whether a program a.s. terminates in finitely many steps on every input, is $\Pi^0_3$-complete.

Being positively a.s.-terminating is not preserved by sequential composition.

Nonetheless:

Expected run-times can be determined compositionally.

$ert(P, t)$ bounds $P$'s expected run-time if $P$'s continuation takes $t$ time.
Expected runtime transformer

Syntax

- skip
- diverge
- x := μ
- observe (G)
- P1 ; P2
- if (G)P1 else P2
- while(G)P

Semantics ert(P, t)

- 1 + t
- ∞
- 1 + λs.[μ](s)(λv.t[x := v](s))
- [G]⋅(1+t)
- ert(P1, ert(P2, t))
- 1 + [G]⋅ert(P1, t) + [¬G]⋅ert(P2, t)
- μX.1 + ([G]⋅ert(P, X) + [¬G]⋅t)

μ is the least fixed point operator wrt. the ordering ≤ on run-times

and a set of proof rules \(^4\) to get two-sided bounds on run-times of loops

\(^4\) Certified using the Isabelle/HOL theorem prover; see [Hölzl, ITP 2016].
Run-time invariant synthesis

\[
\text{while } (x > 0) \{ \ x := x-1 \ \}
\]

A lower $\omega$-invariant is:

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

on iteration

on termination

We obtain:

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

is a lower bound on the program’s runtime.
Run-time invariant synthesis

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

Template for a lower $\omega$-invariant:

$$I_n = 1 + \underbrace{[c \neq 1] \cdot (1 + [x > 0] \cdot 2x)}_{\text{on termination}} + \underbrace{[c = 1] \cdot (a_n + b_n \cdot [x > 0] \cdot 2x)}_{\text{on iteration}}$$

The derived constraints are:

$$a_0 \leq 2 \quad \text{and} \quad a_{n+1} \leq \frac{7}{2} + \frac{1}{2} \cdot a_n \quad \text{and} \quad b_0 \leq 0 \quad \text{and} \quad b_{n+1} \leq 1 + b_n$$

This admits the solution $a_n = 7 - \frac{5}{2^n}$ and $b_n = n$. Then: $\lim_{n \to \infty} I_n = \infty$
Coupon collector’s problem

On a Classical Problem of Probability Theory

by

P. Erdős and A. Rényi

Coupon collector's problem

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 questions: 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 \( t \) 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 about 225 trials to collect all 50 coupons.
Coupon collector’s problem

```plaintext
cp := [0,...,0]; // no coupons yet
i := 1; // coupon to be collected next
x := 0; // number of coupons collected
while (x < N) {
    while (cp[i] != 0) {
        i := uniform(1..N) // next coupon
    }
    cp[i] := 1; // coupon i obtained
    x++; // one coupon less to go
}
```

Using our ert-calculus one can prove that expected run-time is \( \Theta(N \cdot \log N) \).

By systematic code verification à la Floyd-Hoare. Machine checkable.
Overview

1. Probabilistic weakest pre-conditions
2. Bayesian inference by program analysis
3. Termination
4. Runtime analysis
5. How long to sample a Bayes’ network?
6. Epilogue
How long to sample a BN?

[Gordon, Nori, Henzinger, Rajamani, 2014]

“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.”
A toy Bayesian network

This BN is parametric (in $a$)

How often to sample this BN given the evidence $G = 0$?
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?
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 $10^{15}$
Expected runtime of iid-loops

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

\[
er(t) = 1 + [G] \cdot \frac{1 + e(r)(P, \neg G) \cdot t}{1 - w(p)(P, [G])} + [\neg G](s) \cdot t
\]

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

\textbf{Proof:} similar as for the inference (w(p)) using the decomposition result:

\[
er(P, t) = er(t)(P, 0) + w(p)(P, t)
\]

No loop invariant, martingale, or metering function needed. Fully automatable.
Example: sampling within a circle

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

This iid-loop is a.s.-terminating, and every iteration has same expected time.

Then: \( \text{ert}(P_{\text{circle}}, 0) = 1 + [(x-5)^2 + (y-5)^2 \geq 25] \cdot \frac{363}{73} \)

So: \( 1 + \frac{363}{73} \approx 5.97 \) operations are required on average using rejection sampling
How long to sample a Bayes’ network?

**Expected runtime of BN programs**

For every runtime \( t \) we have:

\[
\text{ert}
\left( \begin{array}{c}
\text{repeat } \text{Seq } \text{until } (G), t \\
\text{program of the BN}
\end{array} \right)
\]

\[
= \frac{1 + \text{ert}(\text{Seq}, [G] \cdot t)}{\text{wp}(\text{Seq}, [G])}
\]

Seq is a sequence of blocks, where a block corresponds to a single BN node.

A closed-form for a BN’s expected runtime can be obtained compositionally.

Fully automated way to obtain a BN’s expected sampling time
The student’s mood example

\[ \text{ert} \left( \underbrace{\text{repeat } D; P; G; M \text{ until } (P=1)}_{\text{program of student mood’s BN}}, 0 \right) = \frac{1 + \text{ert}(D; P; G; M, 0)}{\text{wp}(D; P; G; M, [P = 1])} \approx 23.46 \]
### Experimental results

Benchmark BNs from [www.bnlearn.com](http://www.bnlearn.com)

<table>
<thead>
<tr>
<th>BN</th>
<th></th>
<th>V</th>
<th></th>
<th></th>
<th>E</th>
<th></th>
<th>aMB</th>
<th></th>
<th>O</th>
<th></th>
<th>EST</th>
<th>time (s)</th>
<th></th>
<th>O</th>
<th></th>
<th>EST</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>hailfinder</td>
<td>56</td>
<td>66</td>
<td>3.54</td>
<td>5</td>
<td>$5 \times 10^5$</td>
<td>0.63</td>
<td>9</td>
<td>$9 \times 10^6$</td>
<td>0.46</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hepar2</td>
<td>70</td>
<td>123</td>
<td>4.51</td>
<td>1</td>
<td>$1.5 \times 10^2$</td>
<td>1.84</td>
<td>2</td>
<td>—</td>
<td>MO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>win95pts</td>
<td>76</td>
<td>112</td>
<td>5.92</td>
<td>3</td>
<td>$4.3 \times 10^5$</td>
<td>0.36</td>
<td>12</td>
<td>$4 \times 10^7$</td>
<td>0.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pathfinder</td>
<td>135</td>
<td>200</td>
<td>3.04</td>
<td>3</td>
<td>$2.9 \times 10^4$</td>
<td>31</td>
<td>7</td>
<td>$\infty$</td>
<td>5.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>andes</td>
<td>223</td>
<td>338</td>
<td>5.61</td>
<td>3</td>
<td>$5.2 \times 10^3$</td>
<td>1.66</td>
<td>7</td>
<td>$9 \times 10^4$</td>
<td>0.99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pigs</td>
<td>441</td>
<td>592</td>
<td>3.92</td>
<td>1</td>
<td>$2.9 \times 10^3$</td>
<td>0.74</td>
<td>7</td>
<td>$1.5 \times 10^6$</td>
<td>1.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>munin</td>
<td>1041</td>
<td>1397</td>
<td>3.54</td>
<td>5</td>
<td>$\infty$</td>
<td>1.43</td>
<td>10</td>
<td>$1.2 \times 10^{18}$</td>
<td>65</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

aMB = *average Markov Blanket size*, a measure of independence in BNs
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. Probabilistic weakest pre-conditions

2. Bayesian inference by program analysis

3. Termination

4. Runtime analysis

5. How long to sample a Bayes’ network?

6. Epilogue
Predictive probabilistic programming

Analysing probabilistic programs at source code level, compositionally.

Some open problems:
- Completeness
- Sensitivity analysis
- Nondeterminism
- Query processing
- Invariant synthesis
- .......
Thanks to my co-authors!

- **F. Olmedo, F. Gretz, N. Jansen, B. Kaminski, JPK, A. McIver**
- **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.

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

---

5 EATCS best paper award of ETAPS 2016.