On Termination of Probabilistic Programs

Joost-Pieter Katoen

Online Worldwide Seminar Logic and Semantics, April 15, 2020
What we all know about termination

The halting problem
— does a program $P$ terminate on a given input state $s$? —
is semi-decidable.

The universal halting problem
— does a program $P$ terminate on all input states? —
is undecidable.

Alan Mathison Turing
On computable numbers,
with an application to the Entscheidungsproblem
1937
What if programs roll dice?
A radical change

- A program either terminates or not (on a given input)
- Terminating programs have a finite run-time
- Having a finite run-time is compositional

All these facts do not hold for probabilistic programs!
Certain termination

\[
\text{while } (x > 0) \{
\quad x := x - 1 [1/2] x := x - 2
\}
\]

This program never diverges.
For all integer inputs x.
Almost-sure termination

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

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

This program does not always terminate.
It diverges with probability zero.
It almost surely terminates.
Non almost-sure termination

\[ P :: \text{skip } [1/2] \{ \text{call } P; \text{call } P; \text{call } P \} \]

\[ X_P = \frac{1}{2} \cdot 1 + \frac{1}{2} X_P^3 \]
Non almost-sure termination

\[ P :: \text{skip} \ [1/2] \ \{ \text{call P; call P; call P} \} \]

This program terminates with probability \( \frac{\sqrt{5}-1}{2} < 1 \).
Positive almost-sure termination

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

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

This program almost surely terminates. In finite expected time. Despite its possible divergence.
Null almost-sure termination

Consider the symmetric one-dimensional random walk:

```c
int x := 10; while (x > 0) { x-- [1/2] x++ }
```

This program almost surely terminates.
But:
It requires an infinite expected time to do so.
Nuances of termination

...... certain termination

...... termination with probability one
\[\implies \text{almost-sure termination}\]

...... in an expected finite number of steps
\[\implies \text{“positive” almost-sure termination}\]

...... a.s.-termination in an expected infinite number of steps
\[\implies \text{“null” almost-sure termination}\]
Three contributions

The hardness of the various notions of termination.

[MFCS 2015, Acta Informatica 2019]

A powerful proof rule for almost-sure termination.

[POPL 2018]

Proving positive almost-sure termination using weakest pre-conditions.

[ESOP 2016, J. ACM 2018]
Part 1: Hardness of termination

It is a known fact that deciding termination of ordinary programs is undecidable.

Our aim is to classify “how undecidable” (positive) almost-sure termination is.
Kleene and Mostovski

Stephen Kleene (1909–1994)

 Andrzej Mostovski (1913–1975)
Hardness of almost-sure termination

Hardness landscape
Hardness of almost-sure termination

- Hardness landscape
- AST for **one** input is as hard as ordinary termination for **all** inputs
Hardness of almost-sure termination

- Hardness landscape
- AST for one input is as hard as ordinary termination for all inputs
- Finite termination is even “more undecidable”
Proof idea: hardness of positive as-termination

Reduction from the complement of the universal halting problem

For an ordinary program $Q$, provide a probabilistic program $P$ (depending on $Q$) and an input $s$, such that

$P$ terminates in a finite expected number of steps on $s$
if and only if
$Q$ does not terminate on some input
Let’s start simple

```c
bool c := true;
int nrflips := 0;
while (c) {
    nrflips++;
    (c := false [1/2] c := true);
}
```

Expected runtime (integral over the bars):

The `nrflips`-th iteration takes place with probability $\frac{1}{2^{nrflips}}$. 
Reducing an ordinary program to a probabilistic one

Assume an enumeration of all inputs for $Q$ is given.

bool c := true;
int nrflips := 0;
int i := 0;
while (c) {
    // simulate $Q$ for one (further) step on its $i$-th input
    if (Q terminates) {
        cheer; // take $2^{nrflips}$ effectless steps
        i++;
        // reset simulation of program $Q$
    }
    nrflips++;
    (c := false [1/2] c := true);
}
Reducing an ordinary program to a probabilistic one

Assume an enumeration of all inputs for $Q$ is given

```c
bool c := true;
int nrflips := 0;
int i := 0;
while (c) {
    // simulate $Q$ for one (further) step on its $i$-th input
    if (Q terminates) {
        cheer; // take $2^{nrflips}$ effectless steps
        i++;
        // reset simulation of program $Q$
    }
    nrflips++;
    (c := false [1/2] c := true);
}
```

$P$ looses interest in further simulating $Q$ by a coin flip to decide for termination.
**Q does not always halt**

Let $i$ be the first input for which $Q$ does not terminate.

Expected runtime of $P$ (integral over the bars):

**cheering** on termination on input $i-1$
**Q does not always halt**

Let $i$ be the first input for which $Q$ does not terminate.

Expected runtime of $P$ (integral over the bars):

Finite **cheering** — finite expected runtime
Q terminates on all inputs

Expected runtime of $P$ (integral over the bars):

Infinite *cheering* — infinite expected runtime
Hardness of almost sure termination

No change for **non-deterministic** probabilistic programs.
No change when **approximating** termination probabilities.
Part 2: Proving almost-sure termination

- **What?** Termination with probability one. For all inputs.

- **Why?**
  - Reachability can be encoded as termination
  - Often a prerequisite for proving correctness
  - Often implicitly assumed

- **Why is it hard in practice?**
  - Requires a lower bound 1 for termination probability
Almost-sure 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."

Javier Esparza
CAV 2012
How to prove termination?

Use a variant function on the program’s state space whose value — on each loop iteration — is monotonically decreasing with respect to a (strict) well-founded relation.

Alan Mathison Turing
Checking a large routine
1949
Variant (aka: ranking) functions

\( V : \Sigma \rightarrow \mathbb{R}_{\geq 0} \) is \textit{variant function} for loop \texttt{while}(\( G \)) \texttt{P} if for every state \( s \):

1. If \( s \models G \), then \( P \)'s execution on \( s \) terminates in a state \( t \) with:

\[
V(t) \leq V(s) - \varepsilon
\]

for some fixed \( \varepsilon > 0 \), and

2. If \( V(s) \leq 0 \), then \( s \not\models G \).
Termination proofs

\[ V(s^i) \]

\[ \rightarrow \text{loop iterations} \]
Termination proofs

\[ V(s^i) \]

arrival at 0 guaranteed by well-foundedness of

\[ V(s^4) \]

\[ V(s^5) < V(s^4) \]

\[ \rightarrow \text{loop iterations} \]
Termination proofs

$V(s^i)$

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

while \( (x > 0) \) \{ x-- \}

Ranking function \( V = x \).

\[
x := \ldots ; y := \ldots \quad // x \text{ and } y \text{ are positive}
while \ (x \neq y) \{
\quad \text{if } (x > y) \{ x := x-y \} \quad \text{else} \quad \{ y := y-x \}
\}
\]

Ranking function \( V = x + y \).
A large body of existing works

Hart/Sharir/Pnueli: Termination of Probabilistic Concurrent Programs. POPL 1982
Bournez/Garnier: Proving Positive Almost-Sure Termination. RTA 2005
McIver/Morgan: Abstraction, Refinement and Proof for Probabilistic Systems. 2005
Esparza et al.: Proving Termination of Probabilistic Programs Using Patterns. CAV 2012
Chakarov/Sankaranarayanan: Probabilistic Program Analysis w. Martingales. CAV 2013
Fioriti/Hermanns: Probabilistic Termination: Soundness, Completeness, and Compositionality. POPL 2015
Chatterjee et al.: Algorithmic Termination of Affine Probabilistic Programs. POPL 2016
Agrawal/Chatterjee/Novotný: Lexicographic Ranking Supermartingales. POPL 2018

Key ingredient: super- (or some form of) martingales
On super-martingales

A stochastic process $X_1, X_2, \ldots$ is a **martingale** whenever:

$$\mathbb{E}(X_{n+1} \mid X_1, \ldots, X_n) = X_n$$

It is a **super-martingale** whenever:

$$\mathbb{E}(X_{n+1} \mid X_1, \ldots, X_n) \leq X_n$$
A historical perspective

A countable Markov process is “non-dissipative” if almost every infinite path eventually enters — and remains in — positive recurrent states.

\[ \text{expected return} \]
\[ \text{time} < \infty \]
A historical perspective

A countable Markov process is “non-dissipative” if almost every infinite path eventually enters — and remains in — positive recurrent states.

A sufficient condition for being non-dissipative is:

$$\sum_{j \geq 0} j \cdot p_{ij} \leq i$$

for all states $i$

Frederic Gordon Foster
Markoff chains with an enumerable number of states and a class of cascade processes
1951
Kendall’s variation

A Markov process is non-dissipative if for some function $V : \Sigma \to \mathbb{R}$:

$$\sum_{j \geq 0} V(j) \cdot p_{ij} \leq V(i) \quad \text{for all states } i$$

and for each $r \geq 0$ there are finitely many states $i$ with $V(i) \leq r$

David George Kendall
On non-dissipative Markov chains with an enumerable infinity of states
1951
On positive recurrence

Every irreducible positive recurrent Markov chain is non-dissipative.

A Markov process is positive recurrent iff there is a Lyapunov function $V: \Sigma \to \mathbb{R}_{\geq 0}$ with for finite $F \subseteq \Sigma$ and $\varepsilon > 0$:

\[
\sum_j V(j) \cdot p_{ij} < \infty \quad \text{for } i \in F, \text{ and}
\]

\[
\sum_j V(j) \cdot p_{ij} < V(i) - \varepsilon \quad \text{for } i \notin F.
\]
Our aim

A powerful, simple proof rule for almost-sure termination.

At the source code level.

No “descend” into the underlying probabilistic model.
Proving almost-sure termination

The symmetric random walk:

$$\text{while } (x > 0) \{ x := x-1 \ [1/2] \ x := x+1 \}$$

$$V = x$$

$$\mathbb{E}(X_{k+1}) = X_k$$

$$< X_k - \varepsilon$$

does not work

V = x

\[\mathbb{E}(X_{k+1}) = X_k\]

< \[X_k - \varepsilon\]

does not work

does not work
Proving almost-sure termination

The symmetric random walk:

\[
\text{while } (x > 0) \{ x := x-1 \ [1/2] \ x := x+1 \}
\]

Is out-of-reach for many proof rules.

A loop iteration decreases \( x \) by one with probability \( \frac{1}{2} \)

\[ d = 1 \]
\[ p = \frac{1}{2} \]
Proving almost-sure termination

The symmetric random walk:

\[
\textbf{while } (x > 0) \{ x := x-1 \ [1/2] \ x := x+1 \}
\]

Is \textcolor{red}{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!
Are these programs almost surely terminating?

**Escaping spline:**

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

![Image of a diagram showing a random walk process with probabilities and steps labeled.]
Are these programs almost surely terminating?

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

- **A slightly unbiased random walk:**
  
  ```
  1/2-eps ; while (x > 0) { x-- [p] x++ }
  ```

[Diagram showing a random walk with steps labeled with probabilities and positions marked with integers.]
Are these programs almost surely terminating?

- Escaping spline:
  \[
  \text{while } (x > 0) \{ \ p := 1/(x+1); \ (x := 0 \ [p] \ x++) \}
  \]
  ✓

- A slightly unbiased random walk:
  \[
  \frac{1}{2} - \epsilon; \ \text{while } (x > 0) \{ \ x-- \ [p] \ x++ \}
  \]
  ✗

- A symmetric-in-the-limit random walk:
  \[
  \text{while } (x > 0) \{ \ p := x/(2*x+1); \ (x-- \ [p] \ x++) \}
  \]
  ✓

\[0 \ 1 \ 2 \ 3 \ 4 \ 5 \ \ldots\]

Joost-Pieter Katoen
Proving almost-sure termination

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

Ingredients:

- A supermartingale \( V : \Sigma \rightarrow \mathbb{R}_{\geq 0} \) with
  
  \[
  \mathbb{E} \{ V(s_{n+1}) \mid V(s_0), \ldots, V(s_n) \} \leq V(s_n)
  \]
  
  - 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 \( \geq d(v) > 0 \) with probability \( \geq p(v) > 0 \)
  
  - with antitone \( p \) ("probability") and \( d \) ("decrease")

\[
\begin{align*}
\text{monotone} & : x \leq y \quad \Rightarrow \quad f(x) \leq f(y) \\
\text{antitone} & : x \leq y \quad \Rightarrow \quad f(y) \leq f(x)
\end{align*}
\]
Proving almost-sure termination

Goal: prove a.s.–termination of while(G) P, for all inputs

Ingredients:

- A supermartingale \( V : \Sigma \rightarrow \mathbb{R}_{\geq 0} \) with
  - \( \mathbb{E}\{V(s_{n+1}) \mid V(s_0), \ldots, V(s_n)\} \leq V(s_n) \)
  - 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 \( s^i \)
  - \( V(s^i) = \nu \) decreases by \( \geq d(\nu) > 0 \) with probability \( \geq p(\nu) > 0 \)
  - with antitone \( p \) ("probability") and \( d \) ("decrease")

Then: while(G) P is universally almost-surely terminating
Proving almost-sure termination

With prob. $\geq p(V(1))$

$V(i)$

$V(1)$

$V(2)$

$V(3)$

$V(4)$

$s^0$ $s^1$ $s^2$ $s^3$ $s^4$ $s^5$ $s^6$ $s^7$ $s^8$ $s^9$

$\rightarrow$ loop iterations

$a.s.$ arrival at 0 guaranteed by our proof rule

The closer to termination, the more $V$ decreases and this becomes more likely
Proving almost-sure termination

with prob. \( \geq p(V(1)) \)

with prob. \( \geq p(V(4)) \)

\( V(i) \)

\( V(1) \)

\( V(2) \)

\( V(4) \)

\( V(5) \)

\( s^0 \) \( s^1 \) \( s^2 \) \( s^3 \) \( s^4 \) \( s^5 \) \( s^6 \) \( s^7 \) \( s^8 \) \( s^9 \)

→ loop iterations
Proving almost-sure termination

\[ V(i) \]

\[ V(1) \rightarrow V(2) \rightarrow V(4) \rightarrow V(5) \]

\[ s^0 \rightarrow s^1 \rightarrow s^2 \rightarrow s^3 \rightarrow s^4 \rightarrow s^5 \rightarrow s^6 \rightarrow s^7 \rightarrow s^8 \rightarrow s^9 \]

\[ \text{with prob. } \geq p(V(1)) \]

\[ d(V(1)) \]

\[ \text{with prob. } \geq p(V(4)) \]

\[ d(V(1)) \leq d(V(4)) \]

by antitone \( d \)

\[ V(2) \]

\[ V(4) \]

\[ \text{a.s. arrival at 0 guaranteed by our proof rule} \]

The closer to termination, the more \( V \) decreases and this becomes more likely.
Proving almost-sure termination

with prob. \( \geq p(V(1)) \)

\( V(i) \)

with prob. \( \geq p(V(4)) \)

\( d(V(4)) \)

\( d(V(1)) \leq d(V(4)) \)

by antitone \( p \)

by antitone \( d \)

\( V(1) \)

\( V(2) \)

\( V(4) \)

\( V(5) \)

\( s^0 \) \( s^1 \) \( s^2 \) \( s^3 \) \( s^4 \) \( s^5 \) \( s^6 \) \( s^7 \) \( s^8 \) \( s^9 \)

\( \rightarrow \) loop iterations

\( \text{a.s. arrival at 0 guaranteed} \)

by our proof rule
Proving almost-sure termination

\[ p(V(1)) \leq p(V(4)) \]

by antitone \( p \)

\[ d(V(1)) \leq d(V(4)) \]

by antitone \( d \)

The closer to termination, the more \( V \) decreases and this becomes more likely.
The symmetric random walk

Recall:

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

 Witnesses of almost-sure termination:

\[ V = x \]

\[ p(v) = \frac{1}{2} \]

\[ d(v) = 1 \]

That's all you need to prove almost-sure termination!
The symmetric random walk

Recall:

while \( (x > 0) \) { \( x := x - 1 \) \( \frac{1}{2} \) \( x := x + 1 \) }

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!
The escaping spline

Consider the program:

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

Witnesses of almost-sure termination:
- \( V = x \)
- \( p(v) = \frac{1}{v+1} \) and \( d(v) = 1 \)
A symmetric-in-the-limit random walk

Consider the program:

```plaintext
while (x > 0) { p := x/(2*x+1) ; x-- [p] x++ }
```
A symmetric-in-the-limit random walk

Consider the program:

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

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 \( d(v) = \begin{cases} 
\frac{1}{x} & \text{if } v > 0 \text{ and } H_{x-1} < v \leq H_x \\
1 & \text{if } v = 0
\end{cases} \)
Part 3: Proving positive almost-sure termination

- **What?** Termination in finite expected time

- **How?**
  - Weakest-precondition calculus for expected run-times

- **Why?**
  - Reason about the efficiency of randomised algorithms
  - Reason about simulation (in)efficiency of Bayesian networks
  - Is compositional and reasons at the program’s code
AST by weakest preconditions

Determine $wp(P, 1)$ for program $P$ and postcondition $1$.

Dexter Kozen
A probabilistic PDL
1983
The run time of a probabilistic program is random

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

The **expected runtime** is

\[
1 + 3 \cdot \frac{1}{2} + 5 \cdot \frac{1}{4} + \ldots + (2n+1) \cdot \frac{1}{2^n} = \ldots
\]
Expected run-times

- Expected run-time of program $P$ on input $s$:
  \[ \sum_{k=1}^{\infty} k \cdot Pr\left( \text{"} P \text{ terminates after } k \text{ steps on input } s \text{"} \right) \]

- Let $ert$ be a function $t : \Sigma \to \mathbb{R}_{\geq 0} \cup \{\infty\}$

- This is called a run-time. Complete partial order:
  \[ t_1 \leq t_2 \iff \forall s \in \Sigma. \ t_1(s) \leq t_2(s) \]
PAST is **not** compositional

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

**Finite** expected termination time

= **PAST**
PAST is not compositional

Consider the two probabilistic programs:

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

Finite expected termination time

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

Finite termination time
PAST is not compositional

Consider the two probabilistic programs:

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

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

\[ \sum \frac{1}{2^x} \cdot 2^x = \infty \]
Run-times by program verification

\( ert(P, t)(s) \) is the expected run-time of \( P \) on input state \( s \) if \( t \) captures the run-time of the computation following \( P \).
## Expected run-time transformer

### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Run-time $ert(P, t)$</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)P1 else P2</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>▶️ $\text{lfp } X. 1 + ([G] \cdot ert(P, X) + [\neg G] \cdot t)$</td>
</tr>
</tbody>
</table>

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

Plus a set of proof rules to get bounds on run-times of loops
Elementary properties

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

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

- **Affinity**: $ert(P, r \cdot t + t') = ert(P, 0) + r \cdot wp(P, t) + wp(P, t')$
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 \( \infty \): \( ert(P, \infty) = \infty \)

Relation to \( \text{wp} \): \( ert(P, t) = ert(P, 0) + \text{wp}(P, t) \)

Affinity: \( ert(P, r \cdot t + t') = ert(P, 0) + r \cdot \text{wp}(P, t) + \text{wp}(P, t') \)
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 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 $\Theta(n \log(n))$.\(^{[1]}\) For example, when

Joost-Pieter Katoen

On Termination of Probabilistic Programs
Coupon collector’s problem

```plaintext
cp := [0,...,0]; i := 1; x := 0; // no coupons yet
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 the ert-calculus one can prove that:

\[
ert(cpcl, 0) = 4 + [N > 0] \cdot 2N \cdot (2 + H_{N-1}) \in \Theta(N \cdot \log N)
\]

By systematic program verification à la Floyd-Hoare. Machine checkable.
How long to sample a Bayes’ network?

“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." [FOSE 2014]

Andy Gordon
Tom Henzinger
Aditya Nori
Sriram Rajamani
How long to simulate a Bayes network?

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, a measure of independence in BNs

# evidences

ert
Epilogue

1. Hardness of probabilistic termination.
   - AST for one input $\equiv_{\text{hard}}$ universal halting problem.
   - Positive almost-sure termination is $\Pi_3$-complete.

2. Proof rule for almost-sure termination.
   - Widely applicable.

3. Weakest pre-conditions for expected run-time analysis.
   - To (dis)prove positive almost-sure termination. And more.
A big thanks to my co-authors!

Kevin Batz
Benjamin Kaminski
Christoph Matheja
Annabelle McIver
Carroll Morgan
Federico Olmedo
Further reading

- B. Kaminski, JPK, C. Matheja.

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

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

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

- K. Chatterjee, H. Fu and P. Novotny.
  *Termination analysis of probabilistic programs with martingales.*
Using \textit{wp} for expected run-times?

\begin{center}
\texttt{while(true) \{ x++ \}}
\end{center}

\begin{itemize}
\item Consider the post-expectation $x$
\item Characteristic function $\Phi_x(X) = X(x \mapsto x + 1)$
\item Candidate upper bound is $I = 0$
\item Induction: $\Phi_x(I) = 0(x := x + 1) = 0 = I \leq I$
\end{itemize}

We \textit{wrongly} conclude that $0$ is the runtime.

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