Probabilistic Programming

Lecture #16+#17: Expected Runtime Analysis

Joost-Pieter Katoen

RWTH Lecture Series on Probabilistic Programming 2018

Overview

Overview

1. Motivation
2. An unsound approach
3. The expected runtime transformer
4. Properties
5. Proof rules for runtimes of loops
6. Proving positive almost-sure termination
7. Case studies

The runtime of a probabilistic program

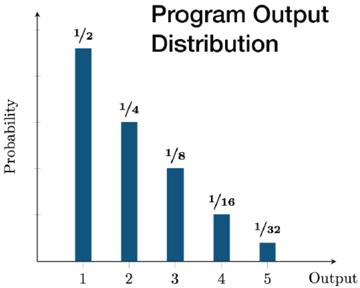
The runtime of a probabilistic program depends

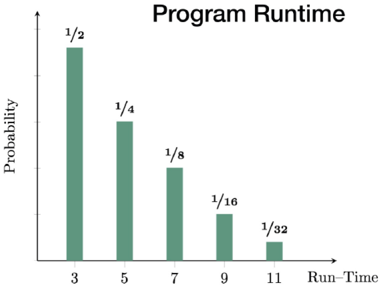
on the input and

on the internal randomness of the program.

The runtime of a probabilistic program is random

```
int i := 0;
repeat {i++; (c := false [0.5] c := true)}
until (c)
```


Program Output Distribution


Program Runtime

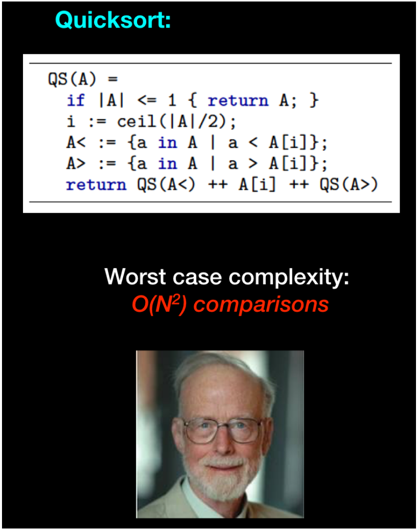
The expected runtime is $1 + 3\cdot{}^{1}/_{2} + 6\cdot{}^{1}/_{4} + \ldots (3n+1)\cdot{}^{1}/_{2^n} \;=\; 5.$

Expected runtimes

Expected run-time of program P on input s:

$$\sum_{i=1}^{\infty} i \cdot Pr\left(\begin{array}{c} \text{“}P\text{ terminates after} \\ i\text{ steps on input }s\text{”} \end{array} \right)$$

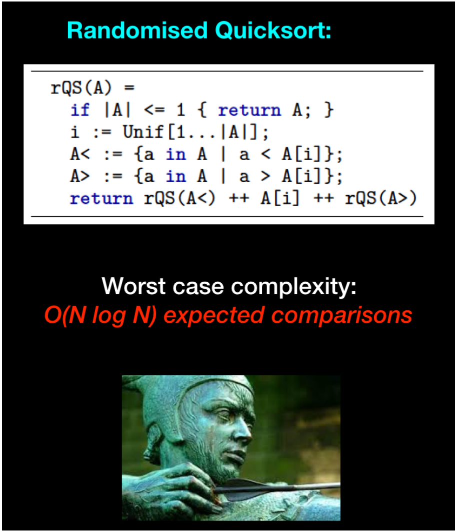
Efficiency of randomised algorithms



Quicksort:

```
QS(A) =
  if |A| <= 1 { return A; }
  i := ceil(|A|/2);
  A< := {a in A | a < A[i]};
  A> := {a in A | a > A[i]};
  return QS(A<) ++ A[i] ++ QS(A>)
```

Worst case complexity:
$O(N^2)$ comparisons

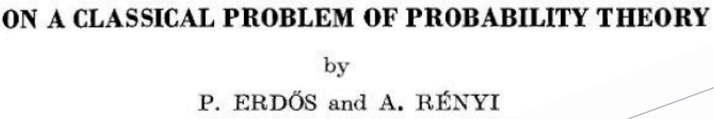


Randomised Quicksort:

```
rQS(A) =
  if |A| <= 1 { return A; }
  i := Unif[1...|A|];
  A< := {a in A | a < A[i]};
  A> := {a in A | a > A[i]};
  return rQS(A<) ++ A[i] ++ rQS(A>)
```

Worst case complexity:
O(N log N) expected comparisons

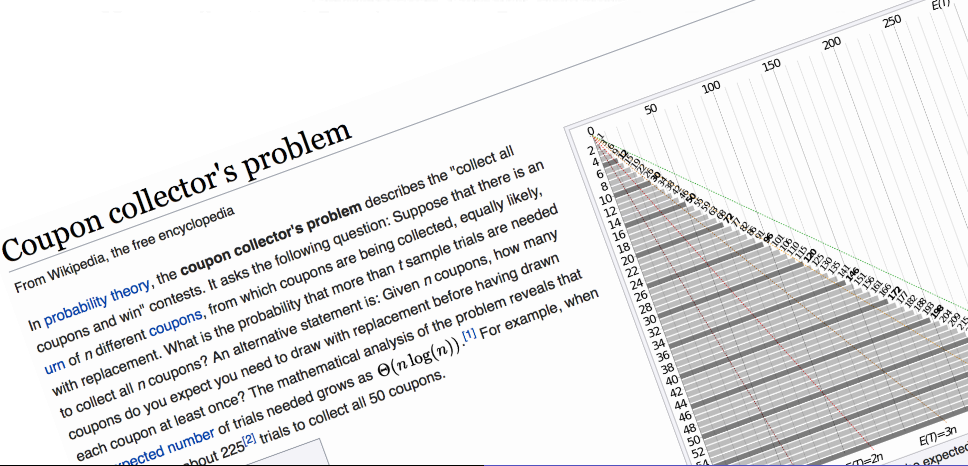
Coupon collector's problem



ON A CLASSICAL PROBLEM OF PROBABILITY THEORY

by

P. ERDŐS and A. RÉNYI



Coupon collector's problem

From Wikipedia, the free encyclopedia

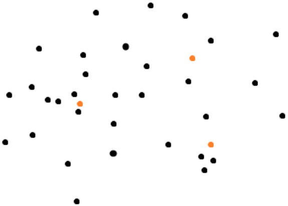
In probability theory, the **coupon collector's problem** describes the "collect all coupons and win" contests. It asks the following question: Suppose that there is an urn of n different coupons, from which coupons are being collected, equally likely, with replacement. What is the probability that more than 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 pected number of trials needed grows as $\Theta(n \log(n))$.[1] For example, when bout 225[2] trials to collect all 50 coupons.

Coupon collector's problem

```
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
}
```

The expected runtime of this program is in $\Theta(N \cdot \log N)$.

Closest-pair problem



Closest-pair problem: find two distinct points $u, v \in \mathbb{R}^2$ among N points in the plane that minimise the Euclidean distance among all pairs of these points.

A naive deterministic approach takes $O(N^2)$. More efficient version in $O(N \cdot \log N)$.

Rabin's randomised algorithm has an expected runtime in $O(N)$.

Randomised primality test

Problem: is N prime or not?

Basic structure of a randomised primality test:

1. Randomly pick a number a, say
2. Do the primality test: Check some equality involving a and N
3. If equality fails, N is composite (with witness a)
4. Otherwise repeat the process.

If after $K > 0$ iterations, N is not found to be composite, then N is probably prime.

Some primality tests

- Fermat primality test:
  Select $a \in \mathbb{Z}$ relative prime to N. If $a^{N-1} \bmod N \neq 1$, then N is composite.
- Rabin-Miller test:
  Select $0 < a < N$. Let $2^s \cdot d = N{-}1$ where d is odd. If $a^d \neq 1 \pmod{N}$ and $a^{2^r \cdot d} \neq -1 \pmod{N}$ for all $0 \leq r \leq s{-}1$, then N is composite.
- Solovay and Strassen test:
  For N odd, pick $a < N$. If $a^{N-1/2} \neq \ldots\ldots$, then N is composite.

Adleman and Huang (1992) provided a randomised primality test that terminates with expected polynomial runtime and certainly provides the correct answer.[1]

---

[1] Decision problems with this characteristic constitute the complexity class ZPP (zero-error probabilistic polynomial time).

The aim of this lecture

A wp-calculus to reason about runtimes at the source code level.

No "descend" into the underlying probabilistic model.

The calculus should be compositional.

$$\mathrm{ert}\,(P\,;\,Q) = \mathrm{ert}\,(P) \overset{?}{\cancel{+}} \mathrm{ert}(Q)$$

Proving positive almost-sure termination

- What? AST+termination in finite expected time

Proving positive almost-sure termination

- What? AST+termination in finite expected time
- Generalise. How?
  - Provide an weakest-precondition calculus
  - . . . . . . for expected runtimes
- Why?
  - Reason about the efficiency of randomised algorithms
  - Reason about simulation efficiency of Bayesian networks
  - Is compositional and reasons at the program's code

Hurdles in runtime analysis

1. Programs may admit diverging runs while still having a finite expected runtime

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

   admits a diverging run but has expected runtime $O(x)$.

2. Having a finite expected time is not compositional w.r.t. sequencing
3. Expected runtimes are extremely sensitive to variations in probabilities

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

   - For p=0, the expected runtime is infinite.
   - For arbitrary small $p > 0$, the expected runtime is $^1/_2 \cdot p \cdot x$, linear in x.

P ::

```
x := 1
c := 1;
while (c) {
    c := 0 [1/2] c := 1;
    x := 2 * x
}
```

;

$$\sum_{i=1}^{\infty} \underbrace{\frac{1}{2^i} \cdot 2^i}_{\text{exp. value of } x \text{ after } i \text{ iteration}} = \infty$$

PAST(P)

Q ::

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

PAST(Q)

$\neg$ PAST(P;Q)

Overview

1. Motivation
2. An unsound approach
3. The expected runtime transformer
4. Properties
5. Proof rules for runtimes of loops
6. Proving positive almost-sure termination
7. Case studies

Re-use weakest preconditions?

Idea: equip the program with a counter `rc`
and use standard wp-reasoning to determine its expected value.

$= P_{rc}$

Determine $wp(\underset{rc}{P}, \texttt{rc})$ for program P.

Dexter Kozen
A probabilistic PDL
1983

An example

Consider the program P:

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

An example

Consider the program *P*:

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

Equipping *P* with a runtime counter yields P_{rc}:

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

An example

Consider the program P:

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

Equipping P with a runtime counter yields P_{rc}:

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

now next replace this fragment

It follows $\Phi(I) \;\leq\; I$ for $I = \texttt{rc} + [x > 0] \cdot 2$.

In total, we thus obtain $wp(P_{rc}, \texttt{rc}) = 2$.

An example

Consider the program *Q*:

```
x := 1;
while (x > 0) { x := 0 [1/2] while(true) { skip } }
```

diverge

An example

Consider the program Q:

```
x := 1;
while (x > 0) { x := 0 [1/2] while(true) { skip } }
```

Equipping Q with a runtime counter yields Q_{rc}:

```
x := 1; rc := 0;
while (x > 0) {
    rc++;
    (x := 0 [1/2] while(true) { rc++ ; skip})
}
```

An example

Consider the program Q:

```
x := 1;
while (x > 0) { x := 0 [1/2] while(true) { skip } }
```

Equipping Q with a runtime counter yields Q_{rc}:

```
x := 1; rc := 0;
while (x > 0) {
    rc++;
    (x := 0 [1/2] while(true) { rc++ ; skip})
}
```

As wp(inner loop, f) = 0 for every f, it follows $\Phi_{Q_{rc}} \leq \Phi_{P_{rc}}$.

↓ "skip" variant of Q_{rc} (prev. slide)

An example

Consider the program Q:

```
x := 1;
while (x > 0) { x := 0 [1/2] while(true) { skip } }
```

Equipping Q with a runtime counter yields Q_{rc}:

```
x := 1; rc := 0;
while (x > 0) {
    rc++;
    (x := 0 [1/2] while(true) { rc++ ; skip})
}
```

As $wp(\text{inner loop}, f) = 0$ for every f, it follows $\Phi_{Q_{rc}} \leq \Phi_{P_{rc}}$.

Thus, $\Phi_{Q_{rc}}(I) \leq \Phi_{P_{rc}}(I) \leq I$ for $I = \texttt{rc} + [x > 0] \cdot 2$.

An example

Consider the program Q:

```
x := 1;
while (x > 0) { x := 0 [1/2] while(true) { skip } }
```

Equipping Q with a runtime counter yields Q_{rc}:

```
x := 1; rc := 0;
while (x > 0) {
    rc++;
    (x := 0 [1/2] while(true) { rc++ ; skip})
}
```

As $wp(\text{inner loop}, f) = 0$ for every f, it follows $\Phi_{Q_{rc}} \leq \Phi_{P_{rc}}$.

Thus, $\Phi_{Q_{rc}}(I) \leq \Phi_{P_{rc}}(I) \leq I$ for $I = \mathtt{rc} + [x > 0] \cdot 2$.

This contradicts the fact that the true expected runtime of Q is ∞.

Overview

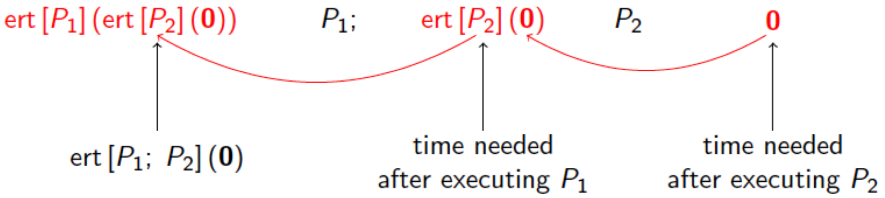
1. Motivation
2. An unsound approach
3. The expected runtime transformer
4. Properties
5. Proof rules for runtimes of loops
6. Proving positive almost-sure termination
7. Case studies

The basic idea

function: $S \longrightarrow \mathbb{R}_{\geq 0} + \infty$

Let $ert() : \text{pGCL} \to (\mathbb{T} \to \mathbb{T})$ where:

- $ert(P, t)(s)$ is the expected runtime of P on input state s if t captures the runtime of the computation following P.
- $ert(P, \mathbf{0})(s)$ is the expected runtime of P on input state s.



Runtimes

Expectations

A expectation $f : \mathbb{S} \rightarrow \mathbb{R}_{\geq 0} \cup \{ \infty \}$.

Let $\mathbb{E}$ be the set of all expectations and let $\sqsubseteq$ be defined for $f, g \in \mathbb{E}$ by:

$$f \sqsubseteq g \quad \text{if and only if} \quad f(s) \leq g(s) \quad \text{for all } s \in \mathbb{S}.$$

Runtimes

A runtime $t : \mathbb{S} \rightarrow \mathbb{R}_{\geq 0} \cup \{ \infty \}$.

Let $\mathbb{T}$ denote the set of all runtimes and let $\leq$ be defined for $t, u \in \mathbb{T}$ by:

$$t \leq u \quad \text{if and only if} \quad t(s) \leq u(s) \quad \text{for all } s \in \mathbb{S}.$$

A runtime transformer is defined in a similar way as an expectation transformer

The runtime model

We assume the following runtimes:

- Executing a `skip`-statement takes a single time unit
- Executing an (ordinary or random) assignment takes a single time unit
- Evaluating a guard takes a single time unit
- Flipping a coin in a probabilistic choice takes a single time unit
- Sequential composition does not take time

The ert-calculus can be easily adapted to other runtime models.

Expected runtime transformer for `pGCL`

ert $(P_1, ●)$

P

Syntax

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

Expected runtime $ert(P, t)$

P_1 ; P_2 t

ert (P_2, t)

- $\mathbf{1}+t$
- ∞
- $\mathbf{1} + t[x := E]$
- $\mathbf{1} + \lambda s. \int_{\mathbb{Q}} (\lambda v. t(s[x := v]))\, d\mu_s$
- $ert(P_1, ert(P_2, t))$
- $\mathbf{1} + [G] \cdot ert(P_1, t) + [\neg G] \cdot ert(P_2, t)$
- $\mathbf{1} + p \cdot ert(P_1, t) + (1-p) \cdot ert(P_2, t)$
- $\mathrm{lfp}\, X.\ (\mathbf{1} + [G] \cdot ert(P, X) + [\neg G] \cdot t)$

Expected runtime transformer for `pGCL`

Syntax

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

Expected runtime $ert(P, t)$

- $\mathbf{1}+t$
- ∞
- $\mathbf{1} + t[x := E]$
- $\mathbf{1} + \lambda s. \int_{\mathbb{Q}} (\lambda v. t(s[x := v]))\, d\mu_s$
- $ert(P_1, ert(P_2, t))$
- $\mathbf{1} + [G] \cdot ert(P_1, t) + [\neg G] \cdot ert(P_2, t)$
- $\mathbf{1} + p \cdot ert(P_1, t) + (1-p) \cdot ert(P_2, t)$
- $\mathrm{lfp}\, X. \, (\mathbf{1} + [G] \cdot ert(P, X) + [\neg G] \cdot t)$

lfp is the least fixed point operator wrt. the ordering $\preceq$ on runtimes

Examples

$P :: \; succ := 1 \; [\tfrac{1}{2}] \; (succ := 1 \; [\tfrac{1}{2}] \; succ := 0)$

$$\mathrm{ert}(P, \underline{0}) = 1 + \tfrac{1}{2}\, \mathrm{ert}(succ := 1, \underline{0}) + \tfrac{1}{2}\, \mathrm{ert}(succ := 1 \; [\tfrac{1}{2}] \; 0, \underline{0})$$

$$= \underbrace{1 + \tfrac{1}{2}\big(1 + \underbrace{\underline{0}\,[succ := 1]}_{= \underline{0}}\big)}_{= 1/2}$$

$$+ \tfrac{1}{2}\Big(1 + \underbrace{\tfrac{1}{2}\, \mathrm{ert}(succ := 1, \underline{0})}_{\frac{1}{2}(1 + \underline{0})} + \underbrace{\tfrac{1}{2}\, \mathrm{ert}(succ := 0, \underline{0})}_{\frac{1}{2}(1 + \underline{0})}\Big)$$

$$= 1 + \tfrac{1}{2} + \tfrac{1}{2}\big(1 + \tfrac{1}{2} + \tfrac{1}{2}\big)$$

$$= \tfrac{5}{2}$$

Overview

1. Motivation
2. An unsound approach
3. The expected runtime transformer
4. Properties
5. Proof rules for runtimes of loops
6. Proving positive almost-sure termination
7. Case studies

Elementary properties

- Continuity: $ert(P, t)$ is continuous on $(\mathbb{T}, \leq)$
- Monotonicity: $t \leq t'$ implies $ert(P, t) \leq ert(P, t')$
- Constant propagation: $ert(P, \mathbf{k} + t) = \mathbf{k} + ert(P, t)$
- Preservation of ∞: $ert(P, \infty) = \infty$
- Connection to wp: $\boxed{ert(P, t) = ert(P, \mathbf{0}) + wp(P, t)}$
- Affinity: $ert(P, a \cdot t + t') \; = \; ert(P, \mathbf{0}) + a \cdot ert(P, t) + ert(P, t')$

$$\text{ert}(P, t+t') = \text{ert}(P,t) + \text{wp}(P,t')$$

Proof (sketch) by induction on the structure of P.

$$\text{ert}(P;Q, t+t') = (*\ \text{def. of ert}\ *)$$

$$\text{ert}(P, \text{ert}(Q, t+t'))$$

$$= (*\ \text{I.H on } Q\ *)$$

$$\text{ert}(P, \underbrace{\text{ert}(Q,t) + \text{wp}(Q,t')})$$

$$= (*\ \text{I.H. on } P\ *)$$

$$\underbrace{\text{ert}(P, \text{ert}(Q,t))}_{\text{ert}(P;Q,t)} + \underbrace{\text{wp}(P, \text{wp}(Q,t'))}_{\text{wp}(P;Q,t')}$$

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

while (G) {P}

$$\Phi'_{t'} = [\neg G]\cdot t' + [G]\,\text{wp}(P,X)$$

To show

$$\underbrace{\text{lfp}\, X.\ \Phi_{t+t'}(X)}_{\text{ert}(\text{loop}, t+t')} = \overbrace{\text{lfp}\, X.\ \Phi_t(X)}^{\text{ert}(\text{loop}, t)} + \underbrace{\text{lfp}\, X.\ \Phi'_{t'}(X)}_{\text{wp}(\text{loop}, t')}$$

This is equivalet to prove:

$$\underbrace{\lim_{n\to\infty} \Phi^{n}_{t+t'}(\underline{0})}_{ert(loop,\, t+t')} = \underbrace{\lim_{n\to\infty} \Phi^{n}_{t}(\underline{0}) + \Phi'^{n}_{t'}(0)}_{ert(loop,\, t) \;+\; wp(loop,\, t')}$$

(*)

Proof: $\forall n: \Phi^{n}_{t+t'}(\underline{0}) = \Phi^{n}_{t}(\underline{0}) + \Phi'^{n}_{t'}(\underline{0})$

by induction on n. Base case $n=0$: $\underline{0} = \underline{0} + \underline{0}$

Ind. step.

$$\Phi^{n+1}_{t+t'}(\underline{0}) = 1 + [\neg G]\cdot(t+t') + [G]\, ert(P, \Phi^{n}_{t+t'}(\underline{0}))$$

$$= \text{(* I.H. on n *)}$$

$$1 + [\neg G](t+t') + [G]\cdot ert(P, \underbrace{\Phi^{n}_{t}(\underline{0}) + \Phi'^{n}_{t'}(\underline{0})}_{})$$

$$= \text{(* I.H. on loop body P *)}$$

$$1 + [\neg G](t+t') + [G]\Big(ert(P, \Phi^{n}_{t}(\underline{0})) + wp(P, \Phi'^{n}_{t'}(\underline{0})) \Big)$$

$$= 1 + [\neg G](t) + [G]\cdot ert(P, \Phi^{n}_{t}(\underline{0}))$$

$$+ [\neg G](t') + [G]\cdot wp(P, \Phi'^{n}_{t'}(\underline{0}))$$

☒

(Positive) almost-sure termination

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

$$\underbrace{ert(P, \mathbf{0})(s) \ < \ \infty}_{\text{positive a.s-termination on } s} \qquad \text{implies} \qquad \underbrace{wp(P, \mathbf{1})(s) \ = \ \mathbf{1}}_{\text{almost-sure termination on } s}$$

Moreover:

$$\underbrace{ert(P, \mathbf{0}) \ \cancel{\leq} < \ \infty}_{\text{universal positive a.s-termination}} \qquad \text{implies} \qquad \underbrace{wp(P, \mathbf{1}) \ = \ \mathbf{1}}_{\text{universal almost-sure termination}}$$

A Markov chain perspective on runtimes

- Consider $ert(P, t)$ for `pCGL` program P
- Consider the Markov chain $[\![P]\!]$ of program P

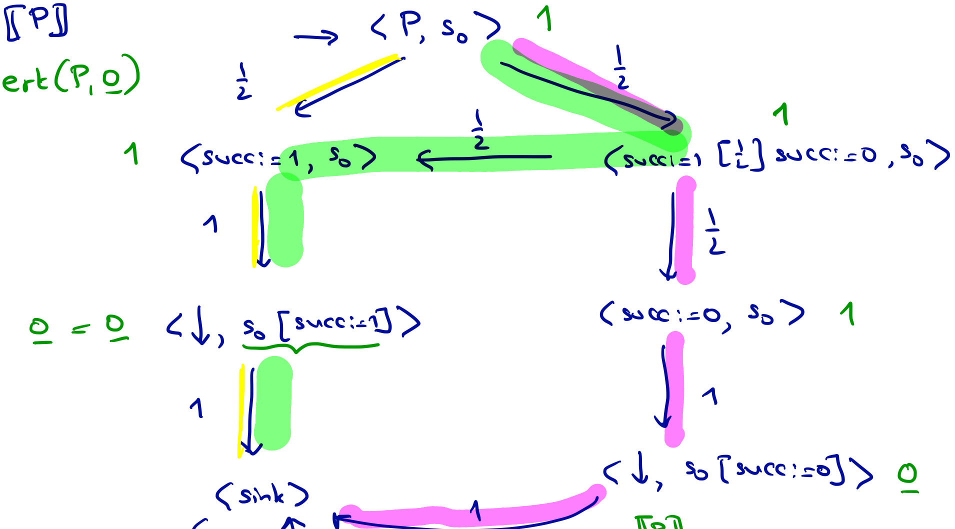
keeps track of the expected runtime of a program statement

- Attach rewards to each Markov chain state in $[\![P]\!]$:
  - State $\langle \downarrow, s \rangle$ gets reward $t(s)$
  - State $\langle \texttt{skip}, s \rangle$ gets reward one
  - State $\langle \texttt{diverge}, s \rangle$ gets reward ∞
  - State $\langle x := E, s \rangle$ gets reward one
  - State $\langle x :\approx \mu, s \rangle$ gets reward one
  - State $\langle \texttt{if}\ G \ldots, s \rangle$ gets reward one
  - State $\langle P[p]Q, s \rangle$ gets reward one
  - State $\langle \texttt{while}(G)P' \ldots, s \rangle$ gets reward one
  - All other states get reward zero

$r : \Sigma \longrightarrow \mathbb{R}_{\geq 0}^{+\infty}$

Example

$succ := 1 \; [\tfrac{1}{2}] \; (succ := 1 \; [\tfrac{1}{2}] \; succ := 0) = P$



$ert(P, 0) = ER^{⟦P⟧}(\Diamond sink)$

Correspondence between *ert*() and Markov chains

Compatibility theorem

For every `pGCL` program P and input s:

$$ert(P, \mathbf{0})(s) \quad = \quad ER^{[\![\, P \,]\!]}\big(\, s, \Diamond sink \,\big)$$

In words: the $ert(P, \mathbf{0})$ for input s equals the expected reward to reach final state *sink* in MC $[\![\, P \,]\!]$ where reward function r in $[\![\, P \,]\!]$ is defined as defined on the previous slide.

Backward compatibility

Deterministic programs

For any `GCL` program P, $\mathit{ert}(P, \mathbf{0})$ equals the number of executed computational steps[2] of P until P terminates.

---

[2] This equals the number of skip statements, guard evaluations and assignments.

Overview

1. Motivation
2. An unsound approach
3. The expected runtime transformer
4. Properties
5. Proof rules for runtimes of loops
6. Proving positive almost-sure termination
7. Case studies

Loops

$$\underset{\underset{\text{subinvariant}}{\uparrow}}{lb} \leq \overset{\text{ert}}{(\text{while}\,(G)\,\{P\})} \leq \underset{\underset{\text{superinvariant}}{\uparrow}}{ub}$$

Reasoning about loops requires — like for wp — invariants.

Runtime invariants

$1 + [\neg G] \cdot t + [G] \cdot \mathrm{ert}(P, X)$

Runtime invariants

Let Φ_{t} be the wp-characteristic function of $P' = \texttt{while}(G)\{P\}$ with respect to post-runtime $t \in \mathbb{T}$ and let $I \in \mathbb{T}$. Then:

Runtime invariants

Runtime invariants

Let Φ_t be the wp-characteristic function of $P' = \texttt{while}(G)\{P\}$ with respect to post-runtime $t \in \mathbb{T}$ and let $I \in \mathbb{T}$. Then:

1. I is a runtime-superinvariant of P' w.r.t. t iff $\underbrace{\Phi_t}_{\text{ert (loop, t)}}(I) \leq I$.

Runtime invariants

Runtime invariants

Let Φ_t be the wp-characteristic function of $P' = \texttt{while}(G)\{P\}$ with respect to post-runtime $t \in \mathbb{T}$ and let $I \in \mathbb{T}$. Then:

1. I is a runtime-superinvariant of P' w.r.t. t iff $\Phi_t(I) \leq I$.
2. I is a runtime-subinvariant of P' w.r.t. t iff $I \leq \Phi_t(I)$.

If I is a runtime-superinvariant of $\texttt{while}(G)\{P\}$ with respect to $t \in \mathbb{T}$, then:

$$ert(\texttt{while}(G)\{P\}, t) \;\leq\; I$$

Example

$$\text{while } (c=1) \ \{ \ c:=0 \ [\tfrac{1}{2}] \ c:=1 \}$$

claim: $\boxed{I = 1 + [c=1]\, 6}$ is a runtime super invariant wrt 0

<u>proof:</u> $\Phi_0(I) \leq I$

$$\Phi_0(I) = 1 + \underbrace{[c \neq 1] \cdot 0}_{=0} + [c=1]\, \mathrm{ert}(\text{body}, I)$$

$$= 1 + [c=1] \left(1 + \tfrac{1}{2}\, \mathrm{ert}(c:=0, I) + \tfrac{1}{2}\, \mathrm{ert}(c:=1, I) \right)$$

$$= 1 + [c=1] \left(1 + \tfrac{1}{2} \left(1 + \underbrace{I(c:=0)}_{1} \right) + \tfrac{1}{2} \left(1 + \underbrace{I(c:=1)}_{7} \right) \right)$$

$$= 1 + [c=1] \underbrace{(1 + 1 + 4)}_{=6}$$

Theorem says that $I = 1 + [c=1] \cdot 6$ is an upperbound of $\mathrm{ert}(\text{program})$

A wrong proof rule for lower bonds

Probabilistic programs do not satisfy:

$$\text{if } \underbrace{I \preceq \Phi_t(I)}_{\text{I is a subinvariant}} \text{ then } \underbrace{I \preceq \mathit{ert}(\texttt{while}(G)\, P, t)}_{\text{I is a lb of ert (loop)}}.$$

These "metering" functions I do work for ordinary programs

[Frohn *et al.*, IJCAR 2016]

why? see lecture on loop invariants
(part on Park's lemma)

A counterexample

```
while (true) { skip [1/2] x++ }
```

- Characteristic functional $F(X) = \mathbf{1} + {}^1\!/_2\,(\mathbf{1} + \mathbf{1} + X[x/x{+}1])$
- Least fixed point is $\mathbf{4}$ as $F(\mathbf{4}) = \mathbf{2} + {}^1\!/_2 \cdot \mathbf{4} = \mathbf{4}$
- $\mathbf{4} + \mathbf{2}^i$ is a fixed point of F too:

$$F\left(\mathbf{4} + \mathbf{2}^i\right) \;=\; \mathbf{2} + \frac{1}{2}\left(\mathbf{4} + \mathbf{2}^{i+1}\right) \;=\; \mathbf{4} + \mathbf{2}^i$$

- Thus: $\mathbf{4} + \mathbf{2}^i \;\preceq\; F\left(\mathbf{4} + \mathbf{2}^i\right)$ but $\mathbf{4} + \mathbf{2}^i \;\not\preceq\; \mathbf{4} \;=\; \mathrm{lfp}\ F$
- In fact, $\mathbf{4} + \mathbf{2}^{i+c}$ is a fixed point of F for any c:

$$F\left(\mathbf{4} + \mathbf{2}^{i+c}\right) \;=\; \mathbf{2} + \frac{1}{2}\left(\mathbf{4} + \mathbf{2}^{i+c+1}\right) \;=\; \mathbf{4} + \mathbf{2}^{i+c}$$

Runtime ω-invariants

$I_0, I_1, I_2, \ldots$

$I_0 \le I_1 \le I_2 \ldots$

Runtime ω-invariants

Let $n \in \mathbb{N}$, $t \in \mathbb{T}$ and Φ_t the ert-characteristic function of `while`$(G)\{P\}$.

The monotonically increasing[3] sequence $(I)_{n\in\mathbb{N}}$ is a runtime-ω-$\underline{\underline{\text{sub}}}$invariant of the loop w.r.t. runtime t iff

lb

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

In a similar way, runtime ω-superinvariants can be defined, but we will not use them here.

---

[3] But not necessarily strictly increasing.

Lower bounds

$I_0 \leq I_1 \leq I_2 \leq I_3$ s.t. $\begin{cases} I_0 \leq \Phi_t(0) \\ I_{n+1} \leq \Phi_t(I_n) \quad \forall n \end{cases}$

Runtime lower bounds

If I_n is a runtime ω-subinvariant of `while`$(G)\{P\}$ with respect to t, then:

$$\sup_n I_n \leq ert(\texttt{while}(G)\, P, t)$$

Example

Consider the same program as for proving an upper bound on the expected runtime.

$P ::$ $\boxed{\text{while } (c=1) \ \{ c:=0 \ [\tfrac{1}{2}] \ c:=1 \}}$ $ert(P, \underline{0})$

guess I_n's structure. How?

$a_0 \le a_1 \le \dots$

$ub = 1 + [c=1] \cdot b$

↑

a_i are monotonically increasing

let $I_n = 1 + [c=1] \cdot a_n$

↑ variable of the invariat

In order for I_n to be a ω-subinvariant we have to show:

① $I_0 \le \Phi_0(\underline{0})$

$$\cancel{1} + \cancel{[c=1]} \cdot a_0 \le \cancel{1} + \underbrace{[c \ne 1]}_{=0} \cdot 0 + \cancel{[c=1]} \cdot (ert(body, \underline{0}))$$

$$a_0 \le 1 + \tfrac{1}{2} \, ert(c:=0, \underline{0}) + \tfrac{1}{2} \, ert(c:=1, \underline{0})$$

$$a_0 \le 1 + \tfrac{1}{2}(1 + \underline{0}) + \tfrac{1}{2}(1 + \underline{0})$$

$\boxed{a_0 \le 2}$ (*)

① $I_{n+1} \le \Phi_0(I_n)$

$$\cancel{1} + \cancel{[c=1]} \cdot a_{n+1} \le \cancel{1} + \underbrace{[c \ne 1]}_{=0} \cdot 0 + \cancel{[c=1]} \, ert(body, I_n)$$

↑

$= \boxed{a_{n+1} \le 3 + \tfrac{1}{2} a_n}$ (**)

A possible solution is

$$\boxed{a_n = 5 - \frac{3}{2^n}}$$

$n=0$ $\quad$ $a_0 = 2$

So it follows that

$$I_n = 1 + [c=1]\left(5 - \frac{3}{2^n}\right)$$

is a ω-subinvariant.

According to the theorem

$$\lim_{n\to\infty} 1 + [c=1]\left(5 - \frac{3}{2^n}\right) =$$

$$1 + [c=1]\cdot 5$$

is a lowerbound to the runtime of our program

Overview

1. Motivation
2. An unsound approach
3. The expected runtime transformer
4. Properties
5. Proof rules for runtimes of loops
6. Proving positive almost-sure termination
7. Case studies

Π_3-complete

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
}
```

Finite expected termination time

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
}
```

Finite expected termination time

PAST

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

Finite termination time

PAST

PAST is not compositional

$\mathrm{ert}(P;Q,\underline{0}) = \infty$

$\Rightarrow \neg \mathrm{PAST}(P;Q)$

Consider the two probabilistic programs:

P

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

Finite expected termination time

;

Q

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

Finite termination time

$\neg \mathrm{PAST}(P;Q)$

Running the right after the left program
yields an infinite expected termination time

$$\mathrm{ert}\,(P; Q, \underline{0})$$

$$= \underbrace{\mathrm{ert}\,(P, \underbrace{\mathrm{ert}\,(Q, \underline{0})}_{\text{①}})}_{\text{②}}$$

function depending on $2 \cdot X$

Q: while $(x>0)$ $\{x--\}$

focus on lower bounds.

Proving that PAST is not compositional (1)

Q :

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

lowerbound on $\mathrm{ert}(Q, \underline{0})$

$\longrightarrow$ ω-subinvariant

J_n

(*) $J_0 \leq \Phi_{Q,\underline{0}}(\underline{0})$

$J_0, J_1, J_2, J_3, \ldots\ldots$

(**) $J_{n+1} \leq \Phi_{Q,\underline{0}}(J_n)$

$J_0 \leq J_1 \leq J_2 \leq \ldots\ldots$

Proving that PAST is not compositional (1)

$n = 10$

Q:

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

(*) and (**)

$x < n$ ~> run Q x times

or

$x \geq n$ ~> run Q n times

It is easy to check that a lower ω-invariant is:

$$J_n \;=\; \mathbf{1} \;+\; \underbrace{\underbrace{[0 < x < n]\cdot \boxed{2x}}_{\text{on iteration}}}_{x<n} \;+\; \underbrace{\underbrace{[x \geq n]\cdot (2n{-}1)}_{\text{on termination}}}_{\text{run Q } n \text{ times}}$$

check whether $x>0$

intuition: J_n is a lower b. on running the loop n times

every iteration of Q takes 2 time units (check $x>0$ and x--)

Proving that PAST is not compositional (1)

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

It is easy to check that a lower ω-invariant is:

$$J_n \;=\; \mathbf{1} \;+\; \underbrace{[0 < x < n]\cdot 2x}_{\text{on iteration}} \;+\; \underbrace{[x \geq n]\cdot(2n-1)}_{\text{on termination}}$$

use this in ert(P, ●)

Thus we obtain that: $0 < x$

$$\lim_{n\to\infty}\left(\mathbf{1} + \underset{\overset{\downarrow}{\infty}}{\widetilde{[0 < x < n]}}\cdot 2x + \cancel{[x \geq n]\cdot(2n-1)}\right) \;=\; \mathbf{1} + [x > 0]\cdot 2x$$

is a lower bound on the runtime of the above program.

Proving that PAST is not compositional (2)

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

aim: a lower bound on $\mathrm{ert}(P; Q, \underline{0})$

$$= \mathrm{ert}(P, \underbrace{\mathrm{ert}(Q, \underline{0})}_{1 + [x>0]\cdot 2x})$$

ω-subinvariant

$I_0, I_1, I_2, I_3, \ldots \qquad I_0 \leq I_1 \leq I_2 \leq \ldots$

$$(*) \quad I_0 \leq \Phi_{P,}(\underline{0}) \;\wedge\; \underbrace{\forall n \in \mathbb{N}.\; I_{n+1} \leq \Phi}_{(**)}{}_{P,}(I_n)$$

How to find I_n ?

Proving that PAST is not compositional (2)

P :

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

Q :

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

Template for a lower ω-invariant of composed program:

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

check c

ert (Q, 0)

$$\underbrace{1 + [x>0]\cdot 2x}$$

lb on

ert (Q, 0)

Proving that PAST is not compositional (2)

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

Template for a lower ω-invariant of composed program:

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

(*) and (**)

The constraints on being a lower ω-invariant yield:

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

Proving that PAST is not compositional (2)

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

Template for a lower ω-invariant of composed program:

$$\lim_{n\to\infty} I_n \;=\; \mathbf{1} \;+\; \underbrace{[c \neq 1]\cdot(\mathbf{1} + [x>0]\cdot 2x)}_{\text{on termination}} \;+\; \underbrace{[c = 1]\cdot(\overset{7-\frac{5}{2^n}}{} + n\cdot[x>0]\cdot 2x)}_{\text{on iteration}}$$

The constraints on being a lower ω-invariant yield:

$a_0 \leq 2$ and $a_{n+1} \leq {}^7\!/_2 + {}^1\!/_2 \cdot a_n$ and $b_0 \leq 0$ and $b_{n+1} \leq 1 + b_n$

This admits the solution $a_n = 7 - {}^5\!/_{2^n}$ and $b_n = n$.

Proving that PAST is not compositional (2)

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

Template for a lower ω-invariant of composed program:

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

The constraints on being a lower ω-invariant yield:

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

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

Proving PAST

The ert-transformer enables to prove

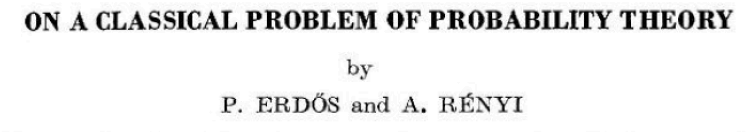
that a program is positively almost-surely terminating

in a compositional manner,

although PAST itself is not a compositional property.

Overview

Coupon collector's problem



ON A CLASSICAL PROBLEM OF PROBABILITY THEORY

by

P. ERDŐS and A. RÉNYI

Coupon collector's problem

```
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
}
```

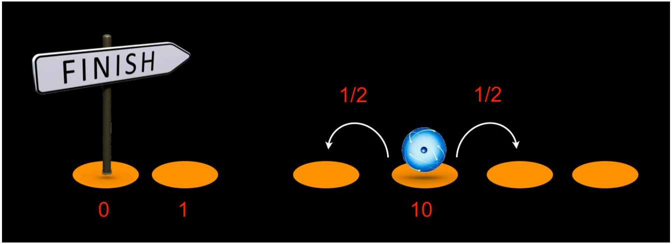
Using the ert-calculus one can prove that:

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

As Harmonic number $H_{N-1} \in \Theta(\log N)$.

By systematic program verification. Machine checkable.

Random walk

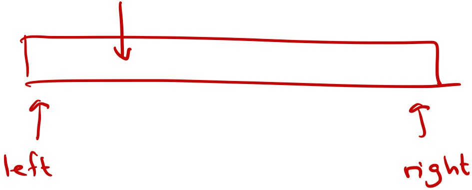


Using the ert-calculus one can prove that its expected runtime is ∞.

By systematic formal verification. Machine checkable.

Randomised binary search

```
proc BinSearch {
 mid := Unif(left, right); // pick mid uniformly
 if (left < right) {
     if (A[mid] < val) {
         left := min(mid+1, right);
         call BinSearch
     } else {
         if (A[mid] > val) {
         right := max(mid-1, left);
         call BinSearch
     } else { skip }
 } else { skip }
}
```



Randomised binary search

```
proc BinSearch {
 mid := Unif(left, right); // pick mid uniformly
 if (left < right) {
     if (A[mid] < val) {
         left := min(mid+1, right);
         call BinSearch
     } else {
         if (A[mid] > val) {
         right := max(mid-1, left);
         call BinSearch
     } else { skip }
 } else { skip }
}
```

Using the ert-calculus one can prove that its expected runtime is $\Theta(\log N)$.

By systematic formal verification. Machine checkable.