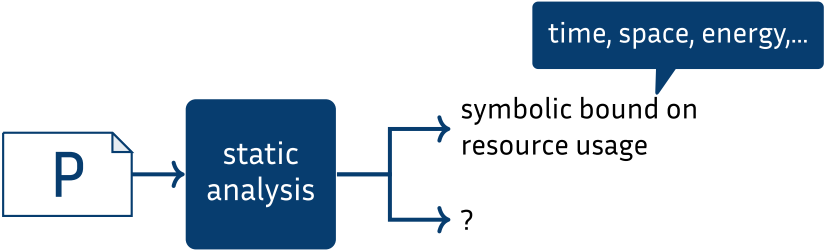
A Modular Cost Analysis for Probabilistic Programs

Martin Avanzini and Georg Moser and Michael Schaper

OOPSLA, SPLASH 2020

Static Resource Analysis



Motivations

- ⋆ integral part of software verification
- ⋆ embedded-systems
- ⋆ detect side-channel attacks
- ⋆ help programmers and compilers ...

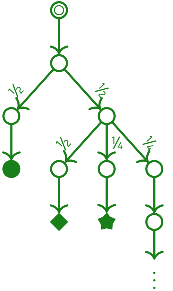
Solutions

- ⋆ recurrence relations
- ⋆ type systems
- ⋆ term rewriting
- ⋆ ...

Conventional vs Probabilistic Programs

Sequential | *Probabilistic*

Dynamics



- ★ randomised algorithms
- ★ cryptography
- ★ robotics
- ★ machine learning
- ★ ...

Semantics

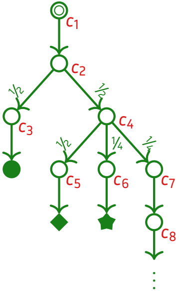
$[\![P]\!](◎) = ●$

$[\![P]\!](◎) = \{●^{1/2}, ◆^{1/4}, ★^{1/8}, \ldots\}$

Conventional vs Probabilistic Programs

Sequential — *Probabilistic*

Dynamics



- ★ randomised algorithms
- ★ cryptography
- ★ robotics
- ★ machine learning
- ★ ...

Semantics $[\![P]\!](◎) = ●$ $\quad [\![P]\!](◎) = \{●^{1/2}, ◆^{1/4}, ⬟^{1/8}, \ldots\}$

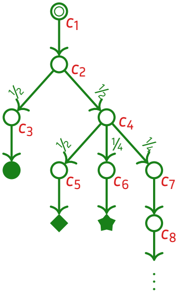
Cost Analysis

- ★ assign cost c_i to each operation
- ★ total cost of computation given by sum of all operation costs

Conventional vs Probabilistic Programs

Sequential — *Probabilistic*

Dynamics



- ★ randomised algorithms
- ★ cryptography
- ★ robotics
- ★ machine learning
- ★ ...

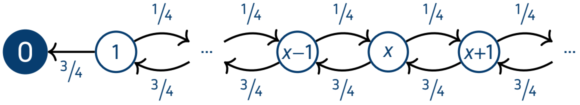
Semantics $[\![P]\!](◎) = ●$ $[\![P]\!](◎) = \{●^{1/2}, ◆^{1/4}, ⬟^{1/8}, \ldots\}$

Cost Analysis

- ★ assign cost c_i to each operation
- ★ total cost of computation given by sum of all operation costs

$$cost(◎) \overset{?}{\leq} b(◎) \qquad \mathbb{E}(cost(◎)) \overset{?}{\leq} b(◎)$$

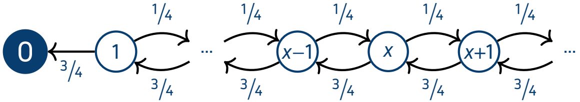
Example: Random walk




```
while (x > 0) {
  b = Coin(3/4);
  if (b = 1) {
    x = x - 1
  } else {
    x = x + 1
  };
  consume(1)
}
```

★ implementation of biased random walk over $\mathbb{N}$, stopping at $x = 0$

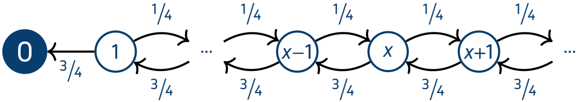
Example: Random walk




```
while (x > 0) {
  b = Coin(3/4);
  if (b = 1) {
    x = x - 1
  } else {
    x = x + 1
  };
  consume(1)
}
```

- ⋆ implementation of biased random walk over $\mathbb{N}$, stopping at $x = 0$
- ⋆ cost given by number of loop iterations

Example: Random walk

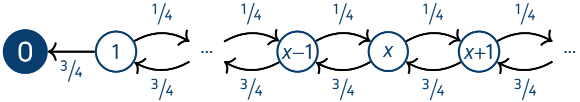



```
while (x > 0) {
  b = Coin(3/4);
  if (b = 1) {
    x = x - 1
  } else {
    x = x + 1
  };
  consume(1)
}
```

- ⋆ implementation of biased random walk over $\mathbb{N}$, stopping at $x = 0$
- ⋆ cost given by number of loop iterations
- ⋆ while potentially non-terminating, expected cost is finite

$$\mathbb{E}(cost(\{x \mapsto n\})) = 2 \cdot n$$

Example: Random walk




```
while (x > 0) {
  b = Coin(3/4);
  if (b = 1) {
    x = x - 1
  } else {
    x = x + 1
  };
  consume(1)
}
```

- ⋆ implementation of biased random walk over $\mathbb{N}$, stopping at $x = 0$
- ⋆ cost given by number of loop iterations
- ⋆ while potentially non-terminating, expected cost is finite

$$\mathbb{E}(cost(\{x \mapsto n\})) = 2 \cdot n$$

- ⋆ manual analysis is difficult, tedious and error prone

State of the Art in Automated Analysis

"Probabilistic Ranking Functions" η

$$s \xrightarrow{c} d \implies \eta(s) \geq c + \mathbb{E}_d(\eta)$$

- ⋆ Lyapunov ranking functions (RF) *[Bournez and Garnier'05]*
- ⋆ super martingale RF *[Chakarov and Sankaranarayanan'13]*
- ⋆ upper invariants *[Kaminski et al.'16]*

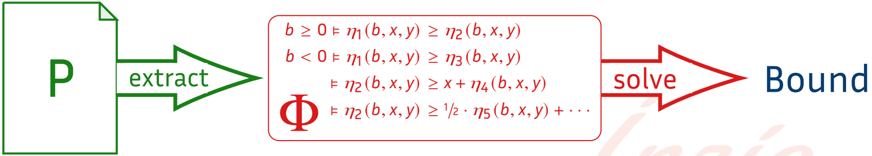
State of the Art in Automated Analysis

"Probabilistic Ranking Functions" η

$$s \xrightarrow{c} d \implies \eta(s) \geq c + \mathbb{E}_d(\eta)$$

- ★ Lyapunov ranking functions (RF) *[Bournez and Garnier'05]*
- ★ super martingale RF *[Chakarov and Sankaranarayanan'13]*
- ★ upper invariants *[Kaminski et al.'16]*

Automation



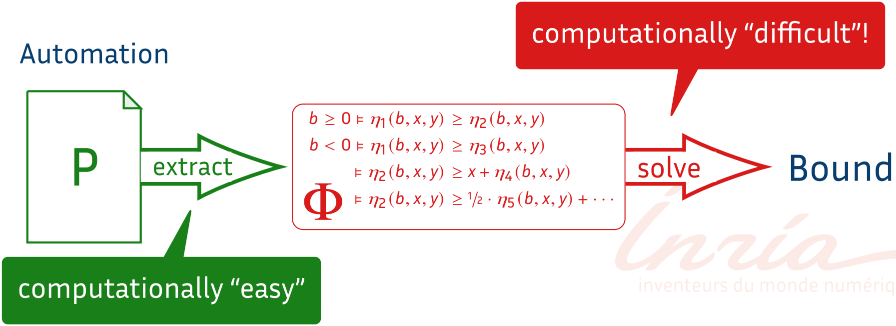
State of the Art in Automated Analysis

"Probabilistic Ranking Functions" η

$$s \xrightarrow{c} d \implies \eta(s) \geq c + \mathbb{E}_d(\eta)$$

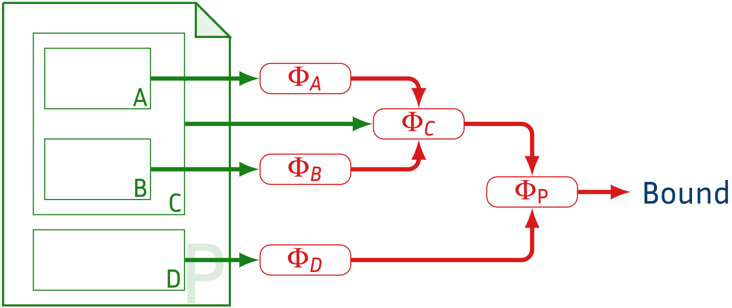
- ★ Lyapunov ranking functions (RF) *[Bournez and Garnier'05]*
- ★ super martingale RF *[Chakarov and Sankaranarayanan'13]*
- ★ upper invariants *[Kaminski et al.'16]*

Automation



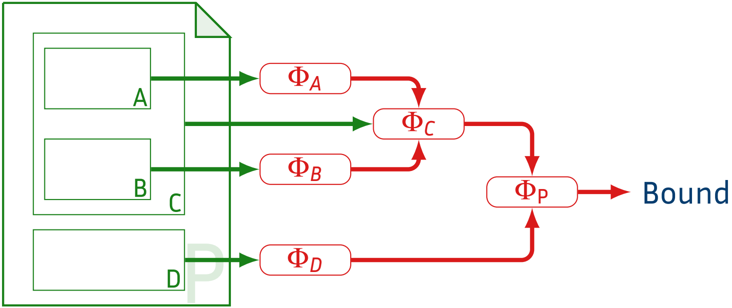
Our Contributions

① a modular cost analysis for probabilistic imperative programs



Our Contributions

① a modular cost analysis for probabilistic imperative programs



- ⋆ based on adaptation of ERT-calculus of Kaminski et al. (2018)
- ⋆ interleaves cost and value analysis

Our Contributions

② a novel operational semantics

- ⋆ in terms of (weighted) probabilistic abstract reduction systems by Bournez & Garnier (2005)
- ⋆ seamless combination of cost, probabilistic and non-deterministic choice

Our Contributions

③ fully automated implementation

- ⋆ competitive in strength
- ⋆ often, orders of magnitude faster
- ⋆ first to support sampling from parametric distributions

http://www-sop.inria.fr/members/Martin.Avanzini/software/eco-imp/

Probabilistic GCL

$$\texttt{C},\texttt{D} ::= \texttt{skip} \mid \texttt{abort} \mid \texttt{C};\texttt{D} \mid \texttt{if}\ (\phi)\ \{\texttt{C}\}\ \{\texttt{D}\} \mid \texttt{while}\ (\phi)\ \{\texttt{C}\}$$

$\mid x := d$ // *probabilistic assignment*

$\mid \texttt{consume}(e)$ // *resource annotation*

- ⋆ probabilistic variation of Dijkstra's *Guarded Command Language*
- ⋆ $x := d$ assigns Integer to x sampled from distribution d
  - – encompasses *usual assignment*: $x := e$
- ⋆ $\texttt{consume}(e)$ incurs cost of $e \geq 0$

Probabilistic GCL

distribution of *final stores*

Semantics

$$[\![\texttt{C}]\!] : \Sigma \to \mathcal{D}(\Sigma)$$

initial store $\sigma \in \Sigma \triangleq \mathsf{Vars} \to \mathbb{Z}$

Probabilistic GCL

Semantics

$$[\![\mathtt{C}]\!] : \Sigma \longrightarrow \mathcal{D}(\Sigma)$$

expected cost $\mathbb{E}(cost_{\mathtt{C}}(\sigma))$
of running $\mathtt{C}$

Expected Cost

$$\mathsf{ecost}[\mathtt{C}] : \Sigma \longrightarrow [0 \ldots \infty]$$

initial store σ

Probabilistic GCL

Semantics

$$[\![\mathtt{C}]\!] : \Sigma \to \mathcal{D}(\Sigma)$$

Expected Cost

$$\mathsf{ecost}[\mathtt{C}] : \Sigma \to [0 \ldots \infty]$$

expected value of f on $[\![\mathtt{C}]\!](\sigma)$, in terms of initial store σ

Expected Value

$$\mathsf{evalue}[\mathtt{C}] : (\Sigma \to [0 \ldots \infty]) \to (\Sigma \to [0 \ldots \infty])$$

non-negative, real-valued function f on stores

The ERT-Calculus

⋆ inspired by Dijkstra's weakest precondition transformer

$$\mathsf{wp}[\mathtt{C}] : (\Sigma \to \mathbb{B}) \to (\Sigma \to \mathbb{B})$$

⋆ within ERT-calculus, generalised to reason about expected costs

$$\mathsf{ert}[\mathtt{C}] : (\Sigma \to [0 \ldots \infty]) \to (\Sigma \to [0 \ldots \infty])$$

B. L. Kaminski et al. *"Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms"*.
JACM, Vol. 65, 30:1–30:68, 2018.

The ERT-Calculus

- ⋆ inspired by Dijkstra's weakest precondition transformer

$$\mathsf{wp}[C] : (\Sigma \to \mathbb{B}) \to (\Sigma \to \mathbb{B})$$

- ⋆ within ERT-calculus, generalised to reason about expected costs

resources required before execution

$$\mathsf{ert}[C] : (\Sigma \to [0 \ldots \infty]) \to (\Sigma \to [0 \ldots \infty])$$

resources (expected time) available after execution

B. L. Kaminski et al. *"Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms"*.
JACM, Vol. 65, 30:1–30:68, 2018.

Expected Cost Transformer

Informal Definition

C	$\mathrm{ect}[\mathtt{C}](f)$
skip	f
abort	$\lambda\sigma.0$
consume(e)	$\lambda\sigma.e(\sigma) + f(\sigma)$
$x := d$	$\lambda\sigma.\sum_{i\in\ldots}$
C; D	ec…
if (ϕ) {C}…	…(σ) if $\sigma \models \phi$ …$(f)(\sigma)$ if $\sigma \models \neg\phi$
while (ϕ) {C}	$f(\sigma)$ if $\sigma \models \neg\phi$ $\mathrm{ect}[\mathtt{C}](\mathrm{ect}[\mathtt{while}\ (\phi)\ \{\mathtt{C}\}](f))(\sigma)$ if $\sigma \models \phi$

natural adaptation of ert[·] to cost analysis

The ECT-Calculus

Theorem (Correctness)

$$\mathrm{ect}[\mathtt{C}](f) = \mathrm{ecost}[\mathtt{C}] + \mathrm{evalue}[\mathtt{C}](f)\ .$$

Thus, in particular,

1. $\mathrm{ecost}[\mathtt{C}] = \mathrm{ect}[\mathtt{C}](\mathbf{0})$*; and*
2. $\mathrm{evalue}[\mathtt{C}](f) = \mathrm{ect}[\mathtt{C}](f)$ *when* $\mathtt{C}$ *is cost-free.*

The ECT-Calculus

Theorem (Correctness)

$$\mathsf{ect}[\mathtt{C}](f) = \mathsf{ecost}[\mathtt{C}] + \mathsf{evalue}[\mathtt{C}](f)\ .$$

Thus, in particular,

1. $\mathsf{ecost}[\mathtt{C}] = \mathsf{ect}[\mathtt{C}](\mathbf{0})$*; and*
2. $\mathsf{evalue}[\mathtt{C}](f) = \mathsf{ect}[\mathtt{C}](f)$ *when* $\mathtt{C}$ *is cost-free.*

Theorem (Upper Invariant)

The following are equivalent:

1. $\mathsf{ecost}[\mathtt{while}\ (\phi)\ \{\mathtt{C}\}] \leq \mathcal{I}$
2. *(i)* $\phi \vDash \mathsf{ecost}[\mathtt{C}] + \mathsf{evalue}[\mathtt{C}](\mathcal{I}) \leq \mathcal{I}$ *and (ii)* $\neg\phi \vDash \mathbf{0} \leq \mathcal{I}$

Automation

```
while (x > 0) {
  b = Coin(3/4);
  if (b = 1) {
    x = x - 1
  } else {
    x = x + 1
  };
  consume(1)
}
```

Constraint Extraction

1. Invariant Template Assignment

$$\mathcal{I}(\sigma) \triangleq \kappa(\langle \sigma \rangle_1, \ldots, \langle \sigma \rangle_n)$$

- $\kappa(r_1, \ldots, r_n) = \sum_{i=1}^{n} \mathsf{k}_i \cdot r_i$ is linear template with unknowns coefficients k_i
- base functions $\langle \cdot \rangle_i$ abstract stores as non-negative numbers

Constraint Solving

Bound

Automation

```
while (x > 0) {
  b = Coin(3/4);
  if (b = 1) {
    x = x - 1
  } else {
    x = x + 1
  };
  consume(1)
}
```

Constraint Extraction

1. Invariant Template Assignment

$$\mathcal{I}(\sigma) \triangleq k \cdot |x|$$

Constraint Solving

Bound

Automation

C

```
while (x > 0) {
    b = Coin(3/4);
    if (b = 1) {
      x = x - 1
    } else {
      x = x + 1
    };
    consume(1)
}
```

Constraint Extraction

1. Invariant Template Assignment

$$\mathcal{I}(\sigma) \triangleq \mathsf{k} \cdot |x|$$

2. Constraint Computation (recursive)

$$x > 0 \vDash \mathsf{ecost}[C] + \mathsf{k} \cdot \mathsf{evalue}[C](|x|) \leq \mathsf{k} \cdot |x|$$

$$x \leq 0 \vDash 0 \leq \mathsf{k} \cdot |x|$$

Constraint Solving

Bound

Automation

```
while (x > 0) {
  b = Coin(3/4);
  if (b = 1) {
    x = x - 1
  } else {
    x = x + 1
  };
  consume(1)
}
```

C

Constraint Extraction

1. Invariant Template Assignment

$$\mathcal{I}(\sigma) \triangleq k \cdot |x|$$

2. Constraint Computation (recursive)

$$x > 0 \vDash 1 + k \cdot (3/4|x-1| + 1/4|x+1|) \quad \leq k \cdot |x|$$
$$x \leq 0 \vDash 0 \leq k \cdot |x|$$

Constraint Solving

1. Reformulation as Non-Negativity Constraints

$$x - 1 \geq 0 \Rightarrow 1/2 k - 1 \geq 0 \qquad -x \geq 0 \Rightarrow -k \cdot x \geq 0$$

2. Reduction to SMT (QF_NRA) via Positivstellensatz (e.g. Handelman)

$$\exists k c, d \geq 0.\ 1/2 k - 1 = c(x-1) \wedge -kx = d(-x)$$

3. Invoke SMT solver $\qquad k = 2 \cdots$

Bound

Experimental Evaluation

Tools

1. eco-imp, implementing the described approach
2. Absynth by *[Ngo et al.'18]*, based on Hoare-style calculus
3. prototype of *[Wang et al.'19]*, based on supermartingale RFs
4. C^4B by *[Carbonneaux et al.'15]* *(for non-probabilistic programs)*

N. C. Ngo, Q. Carbonneaux, and J. Hoffmann. *"Bounded Expectations: Resource Analysis for Probabilistic Programs"*. In *Proc. of 39th PLDI*, pp. 496–512, 2018.

P. Wang et al. *"Cost Analysis of Nondeterministic Probabilistic Programs"*. In *Proc. of 40th PLDI*, pp. 204–220, 2019.

Q. Carbonneaux, J. Hoffmann, and Z. Shao. *"Compositional Certified Resource Bounds"*. In *Proc. of 36th PLDI*, pp. 467–478, 2015.

Experimental Evaluation

Tools

1. `eco-imp`, implementing the described approach
2. `Absynth` by *[Ngo et al.'18]*, based on Hoare-style calculus
3. prototype of *[Wang et al.'19]*, based on supermartingale RFs
4. $\mathtt{C^4B}$ by *[Carbonneaux et al.'15]* *(for non-probabilistic programs)*

Testbed

1. 3 new examples paper + 3 parameterised examples
2. 46 examples from benchmark of *[Ngo et al.'18]* (46 examples)
3. 2 additional examples from *[Wang et al.'19]*
4. 34 deterministic examples from *[Carbonneaux et al.'15]*

Experimental Evaluation
Conclusions

Precision and Strength

- ⋆ on pre-existing benchmarks, precision comparable to existing tools
- ⋆ competitive also on deterministic benchmarks
- ⋆ more advanced examples from the paper only solvable by `eco-imp`

Speed

- ⋆ on average, two orders of magnitude faster (factor 131)

Trader [Ngo et al.'18]

	inferred bound	time (secs)	factor
eco-imp	$10\langle 1+m\rangle\langle p-m\rangle + 5\langle p-m\rangle^2$	0.025	1
Absynth	$10(\langle m\rangle\langle p-m\rangle + \langle p-m\rangle^2) + 5\langle p-m\rangle$	3.638	146
Wang et al.	$5(p^2 - m^2 + p + m - 2)$	10.460	418

Table: Inferred cost and execution times for Trader.

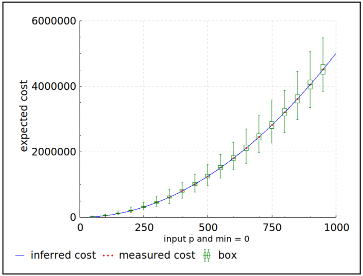


Figure: Inferred vs measured cost.

```
while (p > min ≥ 0) {
  if (Coin(1/4)) {
    p := p + 1
  } else {
    p := p - 1
  };

  n := Uniform(0,10);
  while (n > 0) {
    consume(p);
    n := n - 1
  }
}
```

Coupon Collector [Kaminski et al.'16]

	inferred bound	time (secs)	factor
eco-imp	$\langle n \rangle + 1/2 \langle n \rangle^2$	0.195	1
Absynth	not supported		—
Wang et al.	not supported		—

Table: Inferred cost and execution times for Coupons.

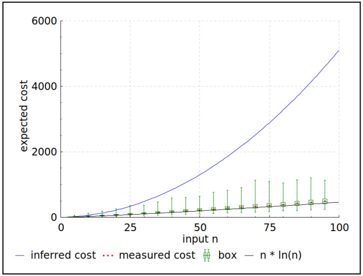


Figure: Inferred vs measured cost.

```
coupons := 0;
while (0 ≤ coupons < n) {
  draw := Uniform(1,n);
  if (draw > coupons) {
    coupons := coupons + 1
  }
  consume(1);
}
```

Nested Loops

	inferred bound	time (secs)	factor
eco-imp	$2\langle 2+n\rangle + 16\langle n\rangle^2 + 64\langle n\rangle^3$	0.068	1
Absynth	$2\langle 1+n\rangle + 4\langle 1+n\rangle^2 + 8\langle 1+n\rangle^3$	5.130	75
Wang et al.	—		—

Table: Inferred cost and execution times for Nest-3.

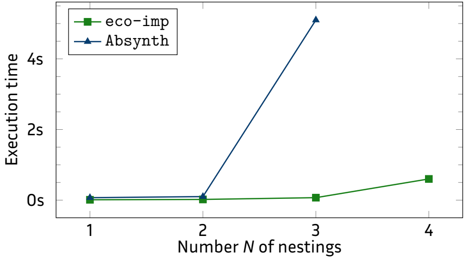


Figure: Execution times vs #nestings.

```
NEST(0) := skip

NEST(N + 1) := {
  x : = m
  while (x > 0) {
    x := x + Uniform(-2,1);
    consume(1);
    NEST(N)
  }
}
```

Conclusion

1. a novel expected cost analysis of probabilistic, imperative programs
   - modular: analysis bottom-up; inside out
   - local: focus on program fragments significantly simplifies constraint extraction and solving

2. proven sound in terms of a novel operational semantics

3. fully automated implementation `eco-imp`

`http://www-sop.inria.fr/members/Martin.Avanzini/software/eco-imp/`