universität
innsbruck

Expected Cost Analysis of Probabilistic Programs

Research Seminar WS21/22

Jonas Schöpf

November 24, 2021

Motivation

- model natural/physical processes $\Rightarrow$ “real” coin flip

Motivation

- model natural/physical processes $\Rightarrow$ “real” coin flip
- cryptography $\Rightarrow$ primality tests

Motivation

- model natural/physical processes $\Rightarrow$ “real” coin flip
- cryptography $\Rightarrow$ primality tests
- robotics

Motivation

- model natural/physical processes $\Rightarrow$ "real" coin flip
- cryptography $\Rightarrow$ primality tests
- robotics
- machine learning

Motivation

- model natural/physical processes $\Rightarrow$ "real" coin flip
- cryptography $\Rightarrow$ primality tests
- robotics
- machine learning
- improvement of algorithms, e.g., quicksort

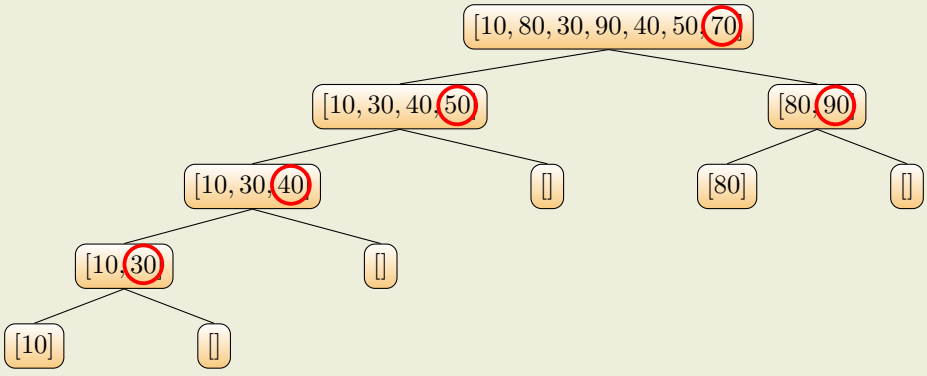
Motivation - Quicksort

- “standard” vs. randomized quicksort

Motivation - Quicksort

- "standard" vs. randomized quicksort

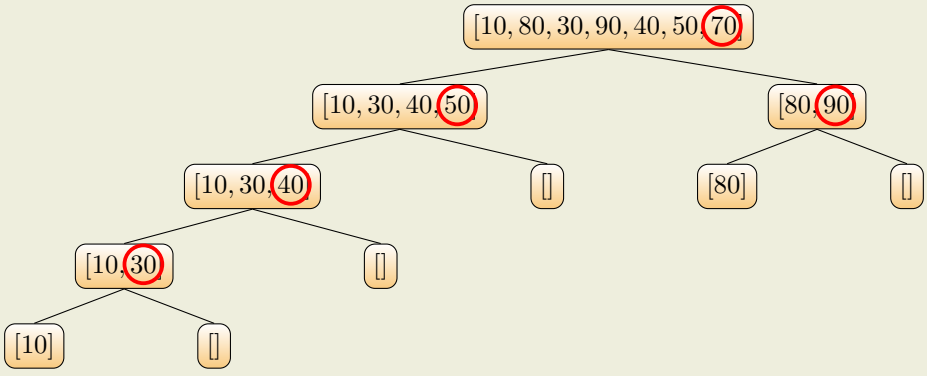
Example Quicksort



Motivation - Quicksort

- "standard" vs. randomized quicksort
- *first* vs. *last* vs. *random* vs. *median* pivot element

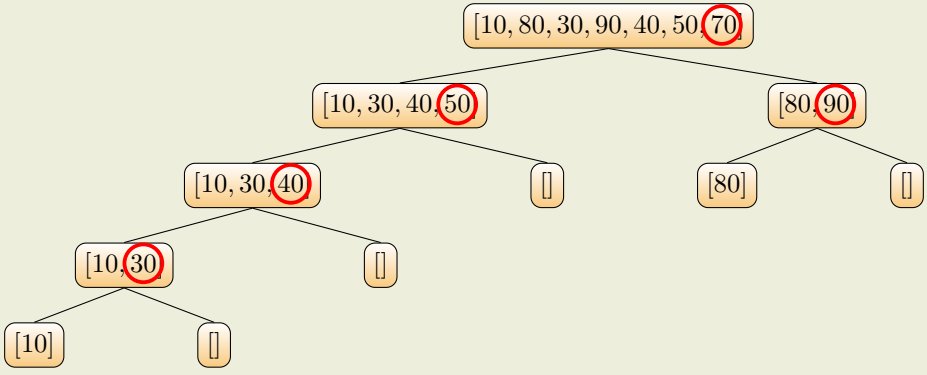
Example Quicksort



Motivation - Quicksort

- "standard" vs. randomized quicksort
- *first* vs. *last* vs. *random* vs. *median* pivot element
- worst case: $\mathcal{O}(n^2)$ vs. $\mathcal{O}(n^2)$ (BUT expected or average time complexity is $\mathcal{O}(n \log n)$)

Example Quicksort



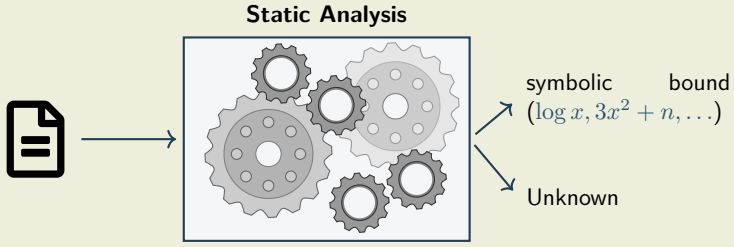
Overview

- Primer
- ERT
- ECT – Syntax & Semantic
- Automation
- Summary

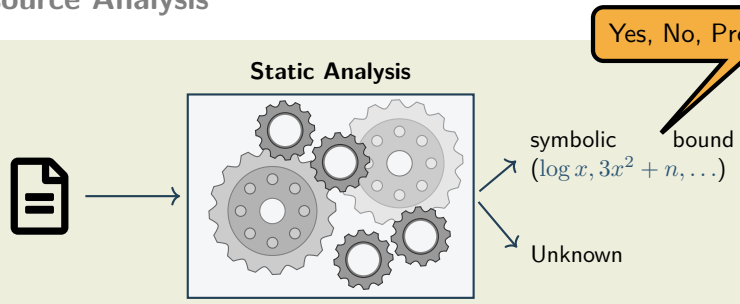
Overview

- Primer
- ERT
- ECT – Syntax & Semantic
- Automation
- Summary

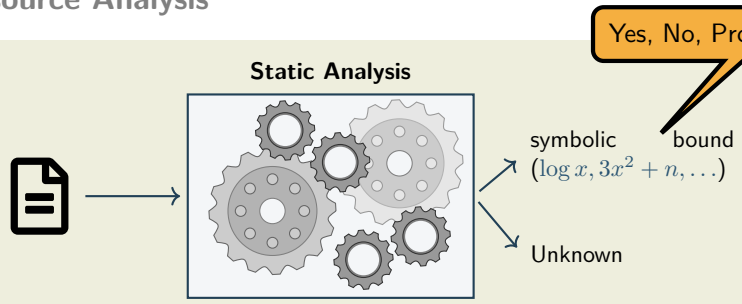
Static Resource Analysis



Static Resource Analysis

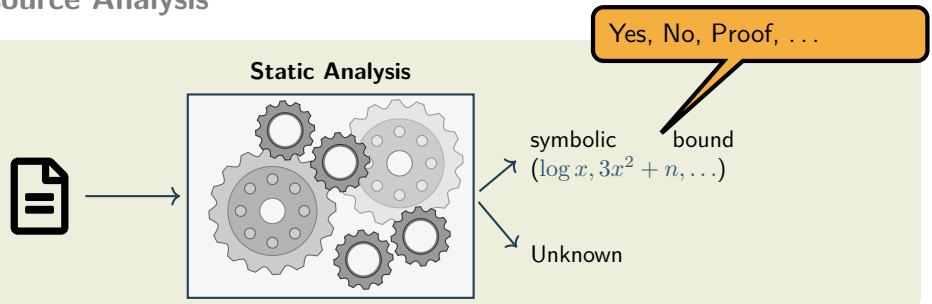


Static Resource Analysis



- integral part of formal verification
- improving the quality of complex software
- medical software, aviation software, nuclear software, . . .

Static Resource Analysis



- integral part of formal verification
- improving the quality of complex software
- medical software, aviation software, nuclear software, . . .

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

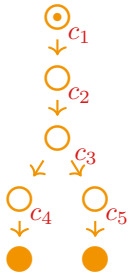
Non-/Deterministic vs. Probabilistic

	Non-/Determi.	Probabilistic
Dynamics		
Semantics		

Non-/Deterministic vs. Probabilistic

Non-/Determi. Probabilistic

Dynamics

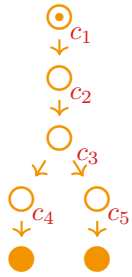


Semantics

Non-/Deterministic vs. Probabilistic

Non-/Determi. Probabilistic

Dynamics



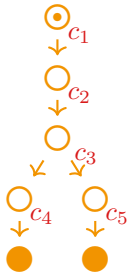
Semantics

- assign cost c_i to each operation
- overall cost is the sum of all operation costs

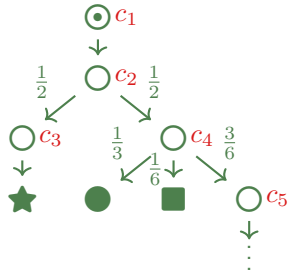
Non-/Deterministic vs. Probabilistic

Dynamics

Non-/Determi.



Probabilistic

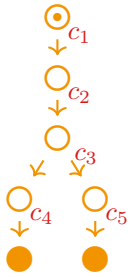


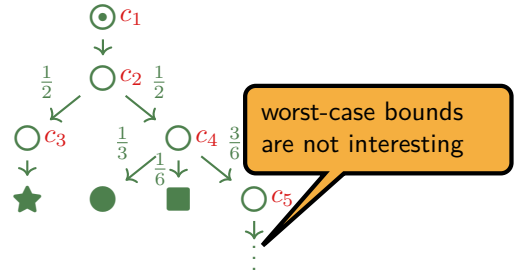
Semantics

- assign cost c_i to each operation
- overall cost is the sum of all operation costs
- deal with probabilities

Non-/Deterministic vs. Probabilistic

Dynamics



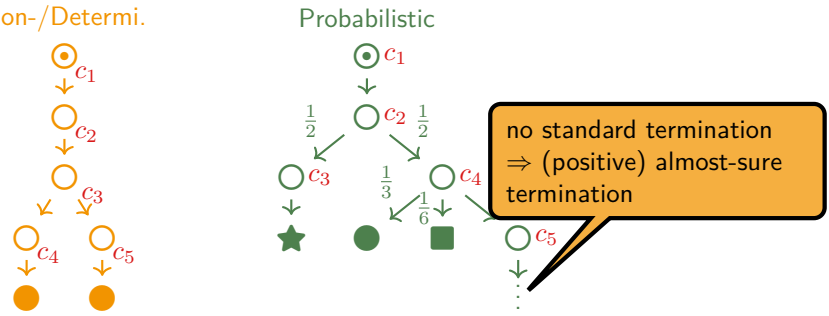


Semantics

- assign cost c_i to each operation
- overall cost is the sum of all operation costs
- deal with probabilities

Non-/Deterministic vs. Probabilistic

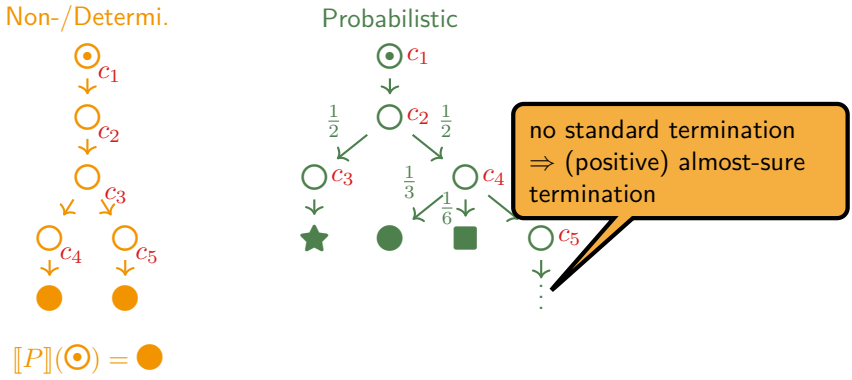
Dynamics



Semantics

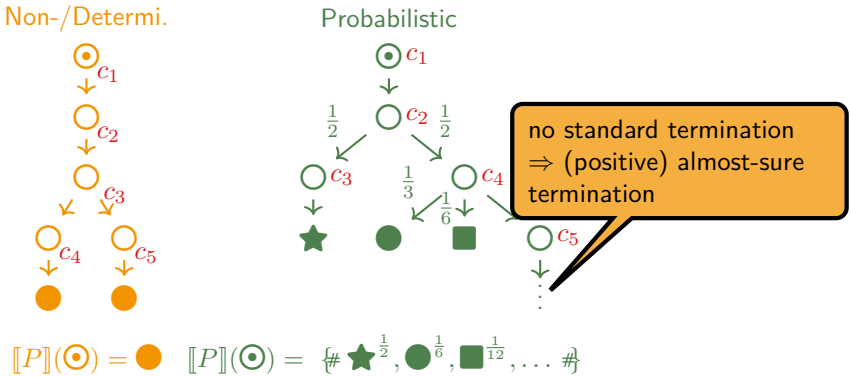
- assign cost c_i to each operation
- overall cost is the sum of all operation costs
- deal with probabilities

Non-/Deterministic vs. Probabilistic



- assign cost c_i to each operation
- overall cost is the sum of all operation costs
- deal with probabilities

Non-/Deterministic vs. Probabilistic



- assign cost c_i to each operation
- overall cost is the sum of all operation costs
- deal with probabilities

Overview

- Primer
- ERT
- ECT – Syntax & Semantic
- Automation
- Summary

Expected Runtime Transformer

- wp-style calculus
- obtains bounds for the expected runtime of randomized algorithms and proves positive almost-sure termination
- can deal with basic constructs, loops and recursion

Expected Runtime Transformer

- wp-style calculus
- obtains bounds for the expected runtime of randomized algorithms and proves positive almost-sure termination
- can deal with basic constructs, loops and recursion

"Assertions about programs" are predicates that are supposed to be "true at this point of the program".

Expected Runtime Transformer

- wp-style calculus
- obtains bounds for the expected runtime of randomized algorithms and proves positive almost-sure termination
- can deal with basic constructs, loops and recursion

"Assertions about programs" are predicates that are supposed to be "true at this point of the program".

Formalized — into logic — it looks as:

$$\{pre\}\ prog\ \{post\} \qquad \text{Hoare-style}$$

$$\text{or} \qquad pre \Rrightarrow \mathsf{wp}.prog.post \qquad \text{Dijkstra-style}$$

Expected Runtime Transformer

- wp-style calculus
- obtains bounds for the expected runtime of randomized algorithms and proves positive almost-sure termination
- can deal with basic constructs, loops and recursion

"Assertions about programs" are predicates that are supposed to be "true at this point of the program".

Formalized — into logic — it looks as:

$$\{pre\}\ prog\ \{post\} \qquad \text{Hoare-style}$$

$$\text{or} \qquad pre \Rrightarrow \mathit{wp}.prog.post \qquad \text{Dijkstra-style}$$

Example

$$\mathbb{C}^{\Sigma} \triangleq \{t \mid t : \Sigma \to \mathbb{R}_{\geq 0}\}$$

Expected Runtime Transformer

- wp-style calculus
- obtains bounds for the expected runtime of randomized algorithms and proves positive almost-sure termination
- can deal with basic constructs, loops and recursion

"Assertions about programs" are predicates that are supposed to be "true at this point of the program".

Formalized — into logic — it looks as:

$$\{pre\}\ prog\ \{post\} \qquad \text{Hoare-style}$$

$$\text{or} \qquad pre \Rrightarrow \mathit{wp}.prog.post \qquad \text{Dijkstra-style}$$

Example

$$\mathbb{C}^{\Sigma} \triangleq \{t \mid t : \Sigma \to \mathbb{R}_{\geq 0}\} \qquad \mathsf{ert}[\bullet] : \mathtt{C} \to (\mathbb{C}^{\Sigma} \to \mathbb{C}^{\Sigma}) \qquad \mathsf{ert}[\mathtt{C}](f)$$

Overview

- Primer
- ERT
- ECT – Syntax & Semantic
- Automation
- Summary

What Do We Want to Achieve?

We would like to have a calculus which to determine the expected runtime of a probabilistic program or algorithm.

- compositional
- modular
- precise

Furthermore it would be beneficial if termination follows from this calculus.

Probabilistic While (`pWhile`)

- inspired by Dijkstra's Guarded Command Language (GCL)

Probabilistic While (`pWhile`)

- inspired by Dijkstra's Guarded Command Language (GCL)
- simple ($\Rightarrow$ simplicity in reasoning helps)

Probabilistic While (`pWhile`)

- inspired by Dijkstra's Guarded Command Language (GCL)
- simple ($\Rightarrow$ simplicity in reasoning helps)
- extended with probabilistic behavior

Probabilistic While (pWhile)

- inspired by Dijkstra's Guarded Command Language (GCL)
- simple ($\Rightarrow$ simplicity in reasoning helps)
- extended with probabilistic behavior

Syntax of pWhile

$$
\begin{aligned}
\mathtt{C}, \mathtt{D} ::= \\
&\mid \mathtt{skip} \\
&\mid \mathtt{abort} \\
\\
&\mid \mathtt{C}; \mathtt{D} \\
&\mid \mathtt{if}(\,\phi\,)\ \{\mathtt{C}\}\ \{\mathtt{D}\} \\
&\mid \mathtt{while}(\phi)\ \{\mathtt{C}\}
\end{aligned}
$$

Probabilistic While (pWhile)

- inspired by Dijkstra's Guarded Command Language (GCL)
- simple ($\Rightarrow$ simplicity in reasoning helps)
- extended with probabilistic behavior

Syntax of pWhile

$$
\begin{aligned}
\mathtt{C}, \mathtt{D} ::= \quad & x := d \\
\mid\ & \mathtt{skip} \\
\mid\ & \mathtt{abort} \\
\\
\mid\ & \mathtt{C}; \mathtt{D} \\
\mid\ & \mathtt{if}(\,\phi\,)\ \{\mathtt{C}\}\ \{\mathtt{D}\} \\
\mid\ & \mathtt{while}(\phi)\ \{\mathtt{C}\}
\end{aligned}
$$

Probabilistic While (pWhile)

- inspired by Dijkstra's Guarded Command Language (GCL)
- simple ($\Rightarrow$ simplicity in reasoning helps)
- extended with probabilistic behavior

Syntax of pWhile

$$
\begin{aligned}
\mathtt{C}, \mathtt{D} ::= \quad & x := d \\
\mid\ & \mathtt{skip} \\
\mid\ & \mathtt{abort} \\
\mid\ & \mathtt{C}; \mathtt{D} \\
\mid\ & \mathtt{if}(\,\phi\,)\ \{\mathtt{C}\}\ \{\mathtt{D}\} \\
\mid\ & \mathtt{while}(\phi)\ \{\mathtt{C}\} \\
\mid\ & \{\mathtt{C}\} <> \{\mathtt{D}\}
\end{aligned}
$$

Probabilistic While (pWhile)

- inspired by Dijkstra's Guarded Command Language (GCL)
- simple ($\Rightarrow$ simplicity in reasoning helps)
- extended with probabilistic behavior

Syntax of pWhile

$$
\begin{aligned}
\texttt{C}, \texttt{D} ::= \quad & x := d \\
\mid\ & \texttt{skip} \\
\mid\ & \texttt{abort} \\
\mid\ & \texttt{consume}(e) \\
\mid\ & \texttt{C}; \texttt{D} \\
\mid\ & \texttt{if}(\,\phi\,)\ \{\texttt{C}\}\ \{\texttt{D}\} \\
\mid\ & \texttt{while}(\phi)\ \{\texttt{C}\} \\
\mid\ & \{\texttt{C}\} <> \{\texttt{D}\}
\end{aligned}
$$

Probabilistic While (pWhile)

- inspired by Dijkstra's Guarded Command Language (GCL)
- simple ($\Rightarrow$ simplicity in reasoning helps)
- extended with probabilistic behavior

Syntax of pWhile

$$
\begin{aligned}
\texttt{C}, \texttt{D} ::= \quad & x := d \\
\mid\ & \texttt{skip} \\
\mid\ & \texttt{abort} \\
\mid\ & \texttt{consume}(e) \\
\mid\ & \texttt{C};\texttt{D} \\
\mid\ & \texttt{if}(\phi)\ \{\texttt{C}\}\ \{\texttt{D}\} \\
\mid\ & \texttt{while}(\phi)\ \{\texttt{C}\} \\
\mid\ & \{\texttt{C}\} <> \{\texttt{D}\}
\end{aligned}
$$

Probabilistic While (pWhile)

- inspired by Dijkstra's Guarded Command Language (GCL)
- simple ($\Rightarrow$ simplicity in reasoning helps)
- extended w

$\texttt{rand}(e), \texttt{unif}(n,m), \texttt{ber}(n,m), \ldots$

Syntax of pWhile

$$
\begin{aligned}
\texttt{C}, \texttt{D} ::= \quad & x := d \\
\mid\ & \texttt{skip} \\
\mid\ & \texttt{abort} \\
\mid\ & \texttt{consume}(e) \\
\mid\ & \texttt{C};\texttt{D} \\
\mid\ & \texttt{if}(\,\phi\,)\ \{\texttt{C}\}\ \{\texttt{D}\} \\
\mid\ & \texttt{while}(\phi)\ \{\texttt{C}\} \\
\mid\ & \{\texttt{C}\} <> \{\texttt{D}\}
\end{aligned}
$$

Probabilistic While (pWhile)

- inspired by Dijkstra's Guarded Command Language (GCL)
- simple ($\Rightarrow$ simplicity in reasoning helps)
- extended with probabilistic behavior

Syntax of pWhile

$$
\begin{aligned}
\texttt{C}, \texttt{D} ::= \quad & x := d \\
\mid\ & \texttt{skip} \\
\mid\ & \texttt{abo} \\
\mid\ & \texttt{consume}(e) \\
\mid\ & \texttt{C};\texttt{D} \\
\mid\ & \texttt{if}(\phi)\ \{\texttt{C}\}\ \{\texttt{D}\} \\
\mid\ & \texttt{while}(\phi)\ \{\texttt{C}\} \\
\mid\ & \{\texttt{C}\} <> \{\texttt{D}\}
\end{aligned}
$$

$\texttt{prob}(n, m)\ \{\texttt{C}\}\ \{\texttt{D}\}$

Probabilistic While (pWhile)

- inspired by Dijkstra's Guarded Command Language (GCL)
- simple ($\Rightarrow$ simplicity in reasoning helps)
- extended with probabilistic behavior

Syntax of pWhile

$$
\begin{aligned}
\texttt{C}, \texttt{D} ::= \quad & x := d \\
\mid\ & \texttt{skip} \\
\mid\ & \texttt{abort} \\
\mid\ & \texttt{consume}(e) \\
\mid\ & \texttt{C};\texttt{D} \\
\mid\ & \texttt{if}(\,\phi\,)\ \{\texttt{C}\}\ \{\texttt{D}\} \\
\mid\ & \texttt{while}(\phi)\ \{\texttt{C}\} \\
\mid\ & \{\texttt{C}\} <> \{\texttt{D}\}
\end{aligned}
$$

Example – geo

$$
\begin{aligned}
& b := 1; \\
& x := 1; \\
& \texttt{while}(b = 1)\ \{ \\
& \quad \texttt{consume}(1); \\
& \quad x := x * 2; \\
& \quad b := \texttt{ber}(1,1)\}
\end{aligned}
$$

Expected Cost Transformer

We define the expected cost transformer (ECT) operating on cost functions over states. Thus $\mathsf{ect}[\mathtt{C}](f)$ can be seen as the cost of $\mathtt{C}$ w.r.t. a continuation cost f.

Expected Cost Transformer

We define the expected cost transformer (ECT) operating on cost functions over states. Thus $\mathsf{ect}[\mathtt{C}](f)$ can be seen as the cost of C w.r.t. a continuation cost f.

C	$\mathsf{ect}[\mathtt{C}](f)$	$\mathsf{evalue}[\mathtt{C}](f)$
$\mathtt{consume}(e)$	$\langle e \rangle + f$	
$\mathtt{skip}$	f	
$\mathtt{abort}$	0	
$x := d$		
$\mathtt{C};\mathtt{D}$		
$\mathtt{if}(\phi)\ \{\mathtt{C}\}\ \{\mathtt{D}\}$		
$\mathtt{while}(\phi)\ \{\mathtt{C}\}$		
$\{\mathtt{C}\} <> \{\mathtt{D}\}$		

Expected Cost Transformer

We define the expected cost transformer (ECT) operating on cost functions over states. Thus $\mathsf{ect}[\mathtt{C}](f)$ can be seen as the cost of C w.r.t. a continuation cost f.

C	$\mathsf{ect}[\mathtt{C}](f)$	$\mathsf{evalue}[\mathtt{C}](f)$
$\mathtt{consume}(e)$	$\langle e \rangle + f$	
$\mathtt{skip}$	f	
$\mathtt{abort}$	0	
$x := d$	$\lambda\sigma.\mathbb{E}_{d(\sigma)}(\lambda v.f[x/v](\sigma))$	
$\mathtt{C};\mathtt{D}$		
$\mathtt{if}(\phi)\ \{\mathtt{C}\}\ \{\mathtt{D}\}$		
$\mathtt{while}(\phi)\ \{\mathtt{C}\}$		
$\{\mathtt{C}\} <> \{\mathtt{D}\}$		

Expected Cost Transformer

We define the expected cost transformer (ECT) operating on cost functions over states. Thus $\mathsf{ect}[\mathtt{C}](f)$ can be seen as the cost of C w.r.t. a continuation cost f.

C	$\mathsf{ect}[\mathtt{C}](f)$	$\mathsf{evalue}[\mathtt{C}](f)$
$\mathtt{consume}(e)$	$\langle e \rangle + f$	
$\mathtt{skip}$	f	
$\mathtt{abort}$	0	
$x := d$	$\lambda\sigma.\mathbb{E}_{d(\sigma)}(\lambda v.f[x/v](\sigma))$	
$\mathtt{C};\mathtt{D}$	$\mathsf{ect}[\mathtt{C}](\mathsf{ect}[\mathtt{D}](f))$	
$\mathtt{if}(\phi)\ \{\mathtt{C}\}\ \{\mathtt{D}\}$		
$\mathtt{while}(\phi)\ \{\mathtt{C}\}$		
$\{\mathtt{C}\} <> \{\mathtt{D}\}$		

Expected Cost Transformer

We define the expected cost transformer (ECT) operating on cost functions over states. Thus $\mathsf{ect}[\mathtt{C}](f)$ can be seen as the cost of C w.r.t. a continuation cost f.

C	$\mathsf{ect}[\mathtt{C}](f)$	$\mathsf{evalue}[\mathtt{C}](f)$
$\mathtt{consume}(e)$	$\langle e \rangle + f$	
$\mathtt{skip}$	f	
$\mathtt{abort}$	0	
$x := d$	$\lambda\sigma.\mathbb{E}_{d(\sigma)}(\lambda v.f[x/v](\sigma))$	
$\mathtt{C};\mathtt{D}$	$\mathsf{ect}[\mathtt{C}](\mathsf{ect}[\mathtt{D}](f))$	
$\mathtt{if}(\phi)\ \{\mathtt{C}\}\ \{\mathtt{D}\}$	$[\phi] \cdot \mathsf{ect}[\mathtt{C}](f) + [\neg\phi] \cdot \mathsf{ect}[\mathtt{D}](f)$	
$\mathtt{while}(\phi)\ \{\mathtt{C}\}$		
$\{\mathtt{C}\} <> \{\mathtt{D}\}$		

Expected Cost Transformer

We define the expected cost transformer (ECT) operating on cost functions over states. Thus $\mathsf{ect}[\mathtt{C}](f)$ can be seen as the cost of C w.r.t. a continuation cost f.

C	$\mathsf{ect}[\mathtt{C}](f)$	$\mathsf{evalue}[\mathtt{C}](f)$
$\mathtt{consume}(e)$	$\langle e \rangle + f$	
$\mathtt{skip}$	f	
$\mathtt{abort}$	0	
$x := d$	$\lambda\sigma.\mathbb{E}_{d(\sigma)}(\lambda v.f[x/v](\sigma))$	
$\mathtt{C};\mathtt{D}$	$\mathsf{ect}[\mathtt{C}](\mathsf{ect}[\mathtt{D}](f))$	
$\mathtt{if}(\phi)\ \{\mathtt{C}\}\ \{\mathtt{D}\}$	$[\phi] \cdot \mathsf{ect}[\mathtt{C}](f) + [\neg\phi] \cdot \mathsf{ect}[\mathtt{D}](f)$	
$\mathtt{while}(\phi)\ \{\mathtt{C}\}$	$\mathtt{lfp}(\lambda F.[\phi] \cdot \mathsf{ect}[\mathtt{C}](F) + [\neg\phi] \cdot f)$	
$\{\mathtt{C}\} <> \{\mathtt{D}\}$		

Expected Cost Transformer

We define the expected cost transformer (ECT) operating on cost functions over states. Thus $\mathsf{ect}[\mathtt{C}](f)$ can be seen as the cost of C w.r.t. a continuation cost f.

C	$\mathsf{ect}[\mathtt{C}](f)$	$\mathsf{evalue}[\mathtt{C}](f)$
$\mathtt{consume}(e)$	$\langle e \rangle + f$	
$\mathtt{skip}$	f	
$\mathtt{abort}$	0	
$x := d$	$\lambda\sigma.\mathbb{E}_{d(\sigma)}(\lambda v.f[x/v](\sigma))$	
$\mathtt{C};\mathtt{D}$	$\mathsf{ect}[\mathtt{C}](\mathsf{ect}[\mathtt{D}](f))$	
$\mathtt{if}(\phi)\ \{\mathtt{C}\}\ \{\mathtt{D}\}$	$[\phi] \cdot \mathsf{ect}[\mathtt{C}](f) + [\neg\phi] \cdot \mathsf{ect}[\mathtt{D}](f)$	
$\mathtt{while}(\phi)\ \{\mathtt{C}\}$	$\mathtt{lfp}(\lambda F.[\phi] \cdot \mathsf{ect}[\mathtt{C}](F) + [\neg\phi] \cdot f)$	
$\{\mathtt{C}\} <> \{\mathtt{D}\}$	$\max(\mathsf{ect}[\mathtt{C}](f), \mathsf{ect}[\mathtt{D}](f))$	

Expected Cost Transformer

We define the expected cost transformer (ECT) operating on cost functions over states. Thus $\mathsf{ect}[\mathtt{C}](f)$ can be seen as the cost of C w.r.t. a continuation cost f.

C	$\mathsf{ect}[\mathtt{C}](f)$	$\mathsf{evalue}[\mathtt{C}](f)$
$\mathtt{consume}(e)$	$\langle e \rangle + f$	f
skip	f	f
abort	0	0
$x := d$	$\lambda\sigma.\mathbb{E}_{d(\sigma)}(\lambda v.f[x/v](\sigma))$	$\lambda\sigma.\mathbb{E}_{d(\sigma)}(\lambda v.f[x/v](\sigma))$
C;D	$\mathsf{ect}[\mathtt{C}](\mathsf{ect}[\mathtt{D}](f))$	
$\mathtt{if}(\phi)$ {C} {D}	$[\phi] \cdot \mathsf{ect}[\mathtt{C}](f) + [\neg\phi] \cdot \mathsf{ect}[\mathtt{D}](f)$	
$\mathtt{while}(\phi)$ {C}	$\mathtt{lfp}(\lambda F.[\phi] \cdot \mathsf{ect}[\mathtt{C}](F) + [\neg\phi] \cdot f)$	
{C} <> {D}	$\max(\mathsf{ect}[\mathtt{C}](f), \mathsf{ect}[\mathtt{D}](f))$	

Expected Cost Transformer

We define the expected cost transformer (ECT) operating on cost functions over states. Thus $\mathsf{ect}[\mathtt{C}](f)$ can be seen as the cost of C w.r.t. a continuation cost f.

C	$\mathsf{ect}[\mathtt{C}](f)$	$\mathsf{evalue}[\mathtt{C}](f)$
$\mathtt{consume}(e)$	$\langle e \rangle + f$	f
$\mathtt{skip}$	f	f
$\mathtt{abort}$	0	0
$x := d$	$\lambda\sigma.\mathbb{E}_{d(\sigma)}(\lambda v.f[x/v](\sigma))$	$\lambda\sigma.\mathbb{E}_{d(\sigma)}(\lambda v.f[x/v](\sigma))$
$\mathtt{C};\mathtt{D}$	$\mathsf{ect}[\mathtt{C}](\mathsf{ect}[\mathtt{D}](f))$	$\mathsf{evalue}[\mathtt{D}](\mathsf{evalue}[\mathtt{D}](f))$
$\mathtt{if}(\phi)\ \{\mathtt{C}\}\ \{\mathtt{D}\}$	$[\phi] \cdot \mathsf{ect}[\mathtt{C}](f) + [\neg\phi] \cdot \mathsf{ect}[\mathtt{D}](f)$	$[\phi] \cdot \mathsf{evalue}[\mathtt{C}](f) + [\neg\phi] \cdot \mathsf{evalue}[\mathtt{D}](f)$
$\mathtt{while}(\phi)\ \{\mathtt{C}\}$	$\mathtt{lfp}(\lambda F.[\phi] \cdot \mathsf{ect}[\mathtt{C}](F) + [\neg\phi] \cdot f)$	$\mathtt{lfp}(\lambda F.[\phi] \cdot \mathsf{evalue}[\mathtt{C}](F) + [\neg\phi] \cdot f)$
$\{\mathtt{C}\} <> \{\mathtt{D}\}$	$\max(\mathsf{ect}[\mathtt{C}](f), \mathsf{ect}[\mathtt{D}](f))$	$\max(\mathsf{evalue}[\mathtt{C}](f), \mathsf{evalue}[\mathtt{D}](f))$

Probabilistic Abstract Reduction Systems (PARS)

Probabilistic Abstract Reduction Systems (PARS)

A PARS over A is given by the ternary relation $\bullet \stackrel{\cdot}{\to} \bullet \subseteq A \times \mathbb{R}_{\geq 0} \times \mathcal{D}(A)$ where a rule is written as $a \stackrel{w}{\to} \delta$.

Probabilistic Abstract Reduction Systems (PARS)

A PARS over A is given by the ternary relation $\bullet \xrightarrow{\cdot} \bullet \subseteq A \times \mathbb{R}_{\geq 0} \times \mathcal{D}(A)$ where a rule is written as $a \xrightarrow{w} \delta$. For example the program geo is modeled by the PARS

$$\mathsf{geo}(x) \xrightarrow[\mathsf{geo}]{1} \{\# \frac{1}{2} : 2 \cdot x, \frac{1}{2} : \mathsf{geo}(2 \cdot x) \#\} \qquad \text{for all } x \in \mathbb{Z}$$

.

Probabilistic Abstract Reduction Systems (PARS)

A PARS over A is given by the ternary relation $\bullet \overset{\cdot}{\to} \bullet \subseteq A \times \mathbb{R}_{\geq 0} \times \mathcal{D}(A)$ where a rule is written as $a \overset{w}{\to} \delta$. For example the program geo is modeled by the PARS

$$\mathsf{geo}(x) \underset{\mathsf{geo}}{\overset{1}{\to}} \left\{\!\!\left\{ \frac{1}{2} : 2 \cdot x, \frac{1}{2} : \mathsf{geo}(2 \cdot x) \right\}\!\!\right\} \qquad \text{for all } x \in \mathbb{Z}$$

. The reduction relation $\bullet \overset{\cdot}{\twoheadrightarrow} \bullet \subseteq \mathcal{M}(A) \times \mathbb{R}_{\geq 0} \times \mathcal{M}(A)$ is defined as

$$\frac{}{\mu \overset{0}{\twoheadrightarrow} \mu} \qquad \frac{a \overset{w}{\to} \delta}{\left\{\!\!\left\{ 1 : a \right\}\!\!\right\} \overset{w}{\twoheadrightarrow} \delta} \qquad \frac{\mu_i \overset{w_i}{\twoheadrightarrow} v_i}{\biguplus_{i \in I} p_i \cdot \mu_i \overset{w}{\twoheadrightarrow} \biguplus_{i \in I} p_i \cdot v_i}$$

.

Probabilistic Abstract Reduction Systems (PARS)

A PARS over A is given by the ternary relation $\bullet \overset{\cdot}{\to} \bullet \subseteq A \times \mathbb{R}_{\geq 0} \times \mathcal{D}(A)$ where a rule is written as $a \overset{w}{\to} \delta$. For example the program geo is modeled by the PARS

$$\mathsf{geo}(x) \underset{\mathsf{geo}}{\overset{1}{\to}} \left\{\!\!\left\{ \frac{1}{2} : 2 \cdot x, \frac{1}{2} : \mathsf{geo}(2 \cdot x) \right\}\!\!\right\} \qquad \text{for all } x \in \mathbb{Z}$$

. The reduction relation $\bullet \overset{\cdot}{\twoheadrightarrow} \bullet \subseteq \mathcal{M}(A) \times \mathbb{R}_{\geq 0} \times \mathcal{M}(A)$ is defined as

$$\frac{}{\mu \overset{0}{\twoheadrightarrow} \mu} \qquad \frac{a \overset{w}{\to} \delta}{\left\{\!\!\left\{ 1 : a \right\}\!\!\right\} \overset{w}{\twoheadrightarrow} \delta} \qquad \frac{\mu_i \overset{w_i}{\twoheadrightarrow} v_i}{\biguplus_{i \in I} p_i \cdot \mu_i \overset{w}{\twoheadrightarrow} \biguplus_{i \in I} p_i \cdot v_i}$$

. Now a reduction using geo looks as follows:

$$\left\{\!\!\left\{ 1 : \mathsf{geo}(1) \right\}\!\!\right\} \underset{\mathsf{geo}}{\overset{1}{\twoheadrightarrow}}$$

Probabilistic Abstract Reduction Systems (PARS)

A PARS over A is given by the ternary relation $\bullet \xrightarrow{\cdot} \bullet \subseteq A \times \mathbb{R}_{\geq 0} \times \mathcal{D}(A)$ where a rule is written as $a \xrightarrow{w} \delta$. For example the program geo is modeled by the PARS

$$\mathsf{geo}(x) \xrightarrow[\mathsf{geo}]{1} \left\{\!\!\left\{ \frac{1}{2} : 2 \cdot x, \frac{1}{2} : \mathsf{geo}(2 \cdot x) \right\}\!\!\right\} \qquad \text{for all } x \in \mathbb{Z}$$

. The reduction relation $\bullet \xrightarrow{\cdot\cdot} \bullet \subseteq \mathcal{M}(A) \times \mathbb{R}_{\geq 0} \times \mathcal{M}(A)$ is defined as

$$\frac{}{\mu \xrightarrow{\cdot\cdot}{}^{0} \mu} \qquad \frac{a \xrightarrow{w} \delta}{\left\{\!\!\left\{ 1 : a \right\}\!\!\right\} \xrightarrow{\cdot\cdot}{}^{w} \delta} \qquad \frac{\mu_i \xrightarrow{\cdot\cdot}{}^{w_i} \nu_i}{\biguplus_{i \in I} p_i \cdot \mu_i \xrightarrow{\cdot\cdot}{}^{w} \biguplus_{i \in I} p_i \cdot \nu_i}$$

. Now a reduction using geo looks as follows:

$$\left\{\!\!\left\{ 1 : \mathsf{geo}(1) \right\}\!\!\right\} \xrightarrow[\mathsf{geo}]{1} \left\{\!\!\left\{ \frac{1}{2} : 2, \frac{1}{2} : \mathsf{geo}(2) \right\}\!\!\right\} \xrightarrow[\mathsf{geo}]{\frac{1}{2}}$$

Probabilistic Abstract Reduction Systems (PARS)

A PARS over A is given by the ternary relation $\bullet \xrightarrow{\cdot} \bullet \subseteq A \times \mathbb{R}_{\geq 0} \times \mathcal{D}(A)$ where a rule is written as $a \xrightarrow{w} \delta$. For example the program geo is modeled by the PARS

$$\mathsf{geo}(x) \xrightarrow[\mathsf{geo}]{1} \left\{\!\!\left\{ \frac{1}{2} : 2 \cdot x, \frac{1}{2} : \mathsf{geo}(2 \cdot x) \right\}\!\!\right\} \qquad \text{for all } x \in \mathbb{Z}$$

. The reduction relation $\bullet \xrightarrow{\cdot\!\cdot} \bullet \subseteq \mathcal{M}(A) \times \mathbb{R}_{\geq 0} \times \mathcal{M}(A)$ is defined as

$$\frac{}{\mu \xrightarrow{\cdot\!\cdot}{}^{0} \mu} \qquad \frac{a \xrightarrow{w} \delta}{\left\{\!\!\left\{ 1 : a \right\}\!\!\right\} \xrightarrow{\cdot\!\cdot}{}^{w} \delta} \qquad \frac{\mu_i \xrightarrow{\cdot\!\cdot}{}^{w_i} \nu_i}{\biguplus_{i \in I} p_i \cdot \mu_i \xrightarrow{\cdot\!\cdot}{}^{w} \biguplus_{i \in I} p_i \cdot \nu_i}$$

. Now a reduction using geo looks as follows:

$$\left\{\!\!\left\{ 1 : \mathsf{geo}(1) \right\}\!\!\right\} \xrightarrow[\mathsf{geo}]{1} \left\{\!\!\left\{ \frac{1}{2} : 2, \frac{1}{2} : \mathsf{geo}(2) \right\}\!\!\right\} \xrightarrow[\mathsf{geo}]{\frac{1}{2}} \left\{\!\!\left\{ \frac{1}{2} : 2, \frac{1}{4} : 4, \frac{1}{4} : \mathsf{geo}(4) \right\}\!\!\right\} \xrightarrow[\mathsf{geo}]{\frac{1}{4}}$$

Probabilistic Abstract Reduction Systems (PARS)

A PARS over A is given by the ternary relation $\bullet \xrightarrow{\cdot} \bullet \subseteq A \times \mathbb{R}_{\geq 0} \times \mathcal{D}(A)$ where a rule is written as $a \xrightarrow{w} \delta$. For example the program geo is modeled by the PARS

$$\mathsf{geo}(x) \xrightarrow[\mathsf{geo}]{1} \{\!\{\tfrac{1}{2} : 2 \cdot x, \tfrac{1}{2} : \mathsf{geo}(2 \cdot x)\}\!\} \qquad \text{for all } x \in \mathbb{Z}$$

. The reduction relation $\bullet \xrightarrow{\cdot\!\cdot} \bullet \subseteq \mathcal{M}(A) \times \mathbb{R}_{\geq 0} \times \mathcal{M}(A)$ is defined as

$$\frac{}{\mu \xrightarrow{\cdot\!\cdot}{}^{0} \mu} \qquad \frac{a \xrightarrow{w} \delta}{\{\!\{1 : a\}\!\} \xrightarrow{\cdot\!\cdot}{}^{w} \delta} \qquad \frac{\mu_i \xrightarrow{\cdot\!\cdot}{}^{w_i} \nu_i}{\biguplus_{i \in I} p_i \cdot \mu_i \xrightarrow{\cdot\!\cdot}{}^{w} \biguplus_{i \in I} p_i \cdot \nu_i}$$

. Now a reduction using geo looks as follows:

$$\{\!\{1 : \mathsf{geo}(1)\}\!\} \xrightarrow[\mathsf{geo}]{1} \{\!\{\tfrac{1}{2} : 2, \tfrac{1}{2} : \mathsf{geo}(2)\}\!\} \xrightarrow[\mathsf{geo}]{\frac{1}{2}} \{\!\{\tfrac{1}{2} : 2, \tfrac{1}{4} : 4, \tfrac{1}{4} : \mathsf{geo}(4)\}\!\} \xrightarrow[\mathsf{geo}]{\frac{1}{4}} \cdots$$

Soundness & Completeness

1. $\mathtt{ect}[\to](f)(\bullet \rhd \mathtt{C}) = \mathtt{ect}[\mathtt{C}](f)$
2. $\mathtt{ecost}[\to](\bullet \rhd \mathtt{C}) = \mathtt{ecost}[\mathtt{C}]$

Soundness & Completeness

1. $\texttt{ect}[\to](f)(\bullet \rhd \mathtt{C}) = \texttt{ect}[\mathtt{C}](f)$
2. $\texttt{ecost}[\to](\bullet \rhd \mathtt{C}) = \texttt{ecost}[\mathtt{C}]$

Upper Invariants

In order to find closed-forms of a loop we need to find a upper invariants. Such invariants are prefix points of the loops characteristic function.

Soundness & Completeness

1. $\texttt{ect}[\to](f)(\bullet \rhd \texttt{C}) = \texttt{ect}[\texttt{C}](f)$
2. $\texttt{ecost}[\to](\bullet \rhd \texttt{C}) = \texttt{ecost}[\texttt{C}]$

Upper Invariants

In order to find closed-forms of a loop we need to find a upper invariants. Such invariants are prefix points of the loops characteristic function. For every cost function I over a state Σ

$$\phi \models \texttt{ect}[\texttt{C}](I) \leq I \land \neg\phi \models f \leq I$$

Soundness & Completeness

1. $\mathtt{ect}[\to](f)(\bullet \rhd \mathtt{C}) = \mathtt{ect}[\mathtt{C}](f)$
2. $\mathtt{ecost}[\to](\bullet \rhd \mathtt{C}) = \mathtt{ecost}[\mathtt{C}]$

Upper Invariants

In order to find closed-forms of a loop we need to find a upper invariants. Such invariants are prefix points of the loops characteristic function. For every cost function I over a state Σ

$$\phi \models \mathtt{ect}[\mathtt{C}](I) \leq I \land \neg\phi \models f \leq I \Longrightarrow \mathtt{ect}[\mathtt{while}(\phi)\ \{\mathtt{C}\}](f) \leq I$$

Soundness & Completeness

1. $\mathtt{ect}[\to](f)(\bullet \rhd \mathtt{C}) = \mathtt{ect}[\mathtt{C}](f)$
2. $\mathtt{ecost}[\to](\bullet \rhd \mathtt{C}) = \mathtt{ecost}[\mathtt{C}]$

Upper Invariants

In order to find closed-forms of a loop we need to find a upper invariants. Such invariants are prefix points of the loops characteristic function. For every cost function I over a state Σ

$$\phi \models \mathtt{ect}[\mathtt{C}](I) \leq I \wedge \neg\phi \models f \leq I \Longrightarrow \mathtt{ect}[\mathtt{while}(\phi)\ \{\mathtt{C}\}](f) \leq I$$

Example – geo

Define $I = [b = 1] \cdot 2$.

Soundness & Completeness

1. $\mathtt{ect}[\to](f)(\bullet \rhd \mathtt{C}) = \mathtt{ect}[\mathtt{C}](f)$
2. $\mathtt{ecost}[\to](\bullet \rhd \mathtt{C}) = \mathtt{ecost}[\mathtt{C}]$

Upper Invariants

In order to find closed-forms of a loop we need to find a upper invariants. Such invariants are prefix points of the loops characteristic function. For every cost function I over a state Σ

$$\phi \models \mathtt{ect}[\mathtt{C}](I) \leq I \wedge \neg\phi \models f \leq I \Longrightarrow \mathtt{ect}[\mathtt{while}(\phi)\ \{\mathtt{C}\}](f) \leq I$$

Example – geo

Define $I = [b = 1] \cdot 2$. Thus we have to check $b = 1 \models \mathtt{ect}[\mathtt{C}_{\mathsf{geo}}](I) \leq I$ and $b \neq 1 \models 0 \leq I$.

Consider $\mathtt{C}_{\mathtt{geo}}$ denoting the body of geo and $I = [b = 1] \cdot 2$ the upper invariant:

Example – geo

$$
\begin{array}{l}
b := 1; \\
x := 1; \\
\mathtt{while}(b = 1)\ \{ \\
\quad \mathtt{consume}(1); \\
\quad x := x * 2; \\
\quad b := \mathtt{ber}(1, 1)\}
\end{array}
$$

Consider $\mathtt{C}_{\mathtt{geo}}$ denoting the body of geo and $I = [b = 1] \cdot 2$ the upper invariant:

$\mathsf{ect}[\mathtt{C}_{\mathtt{geo}}](I) =$

Example – geo

$b := 1;$
$x := 1;$
$\mathtt{while}(b = 1)\ \{$
  $\mathtt{consume}(1);$
  $x := x * 2;$
  $b := \mathtt{ber}(1, 1)\}$

Consider $\mathtt{C}_{\mathtt{geo}}$ denoting the body of geo and $I = [b = 1] \cdot 2$ the upper invariant:

$$\begin{aligned} &\mathsf{ect}[\mathtt{C}_{\mathtt{geo}}](I) = \\ &= \mathsf{ect}[\mathtt{consume}(1)](\mathsf{ect}[x := x * 2](\mathsf{ect}[b := \mathtt{ber}(1,1)](I))) \end{aligned}$$

Example – geo

$$\begin{aligned} &b := 1; \\ &x := 1; \\ &\mathtt{while}(b = 1)\ \{ \\ &\quad \mathtt{consume}(1); \\ &\quad x := x * 2; \\ &\quad b := \mathtt{ber}(1,1)\} \end{aligned}$$

Consider $\mathtt{C}_{\mathtt{geo}}$ denoting the body of geo and $I = [b = 1] \cdot 2$ the upper invariant:

$$\begin{aligned}
&\mathsf{ect}[\mathtt{C}_{\mathtt{geo}}](I) = \\
&= \mathsf{ect}[\mathtt{consume}(1)](\mathsf{ect}[x := x * 2](\mathsf{ect}[b := \mathtt{ber}(1,1)](I))) \\
&= \mathsf{ect}[\mathtt{consume}(1)](\mathsf{ect}[x := x * 2](\frac{1}{2} \cdot I[b/1] + \frac{1}{2} \cdot I[b/0]))
\end{aligned}$$

Example – geo

$$\begin{aligned}
&b := 1; \\
&x := 1; \\
&\mathtt{while}(b = 1)\ \{ \\
&\quad \mathtt{consume}(1); \\
&\quad x := x * 2; \\
&\quad b := \mathtt{ber}(1,1)\}
\end{aligned}$$

Consider $\mathtt{C}_{\mathtt{geo}}$ denoting the body of geo and $I = [b = 1] \cdot 2$ the upper invariant:

$$
\begin{aligned}
&\mathsf{ect}[\mathtt{C}_{\mathtt{geo}}](I) = \\
&= \mathsf{ect}[\mathtt{consume}(1)](\mathsf{ect}[x := x * 2](\mathsf{ect}[b := \mathtt{ber}(1,1)](I))) \\
&= \mathsf{ect}[\mathtt{consume}(1)](\mathsf{ect}[x := x * 2](\frac{1}{2} \cdot I[b/1] + \frac{1}{2} \cdot I[b/0])) \\
&= \mathsf{ect}[\mathtt{consume}(1)](\frac{1}{2} \cdot I[b/1, x/2 \cdot x] + \frac{1}{2} \cdot I[b/0, x/2 \cdot x])
\end{aligned}
$$

Example – geo

$$
\begin{aligned}
&b := 1; \\
&x := 1; \\
&\mathtt{while}(b = 1)\ \{ \\
&\quad \mathtt{consume}(1); \\
&\quad x := x * 2; \\
&\quad b := \mathtt{ber}(1,1)\}
\end{aligned}
$$

Consider $\mathtt{C}_{\mathtt{geo}}$ denoting the body of geo and $I = [b = 1] \cdot 2$ the upper invariant:

$$
\begin{aligned}
&\mathsf{ect}[\mathtt{C}_{\mathtt{geo}}](I) = \\
&= \mathsf{ect}[\mathtt{consume}(1)](\mathsf{ect}[x := x * 2](\mathsf{ect}[b := \mathtt{ber}(1,1)](I))) \\
&= \mathsf{ect}[\mathtt{consume}(1)](\mathsf{ect}[x := x * 2](\frac{1}{2} \cdot I[b/1] + \frac{1}{2} \cdot I[b/0])) \\
&= \mathsf{ect}[\mathtt{consume}(1)](\frac{1}{2} \cdot I[b/1, x/2 \cdot x] + \frac{1}{2} \cdot I[b/0, x/2 \cdot x]) \\
&= 1 + \frac{1}{2} \cdot I[b/1, x/2 \cdot x] + \frac{1}{2} \cdot I[b/0, x/2 \cdot x]
\end{aligned}
$$

Example – geo

$$
\begin{aligned}
&b := 1; \\
&x := 1; \\
&\mathtt{while}(b = 1)\ \{ \\
&\quad \mathtt{consume}(1); \\
&\quad x := x * 2; \\
&\quad b := \mathtt{ber}(1,1)\}
\end{aligned}
$$

Consider $\mathtt{C}_{\mathtt{geo}}$ denoting the body of geo and $I = [b = 1] \cdot 2$ the upper invariant:

$$
\begin{aligned}
&\mathsf{ect}[\mathtt{C}_{\mathtt{geo}}](I) = \\
&= \mathsf{ect}[\mathtt{consume}(1)](\mathsf{ect}[x := x * 2](\mathsf{ect}[b := \mathtt{ber}(1,1)](I))) \\
&= \mathsf{ect}[\mathtt{consume}(1)](\mathsf{ect}[x := x * 2](\frac{1}{2} \cdot I[b/1] + \frac{1}{2} \cdot I[b/0])) \\
&= \mathsf{ect}[\mathtt{consume}(1)](\frac{1}{2} \cdot I[b/1, x/2 \cdot x] + \frac{1}{2} \cdot I[b/0, x/2 \cdot x]) \\
&= 1 + \frac{1}{2} \cdot I[b/1, x/2 \cdot x] + \frac{1}{2} \cdot I[b/0, x/2 \cdot x] \\
&= 1 + \frac{1}{2} \cdot [1 = 1] \cdot 2 + \frac{1}{2} \cdot [0 = 1] \cdot 2
\end{aligned}
$$

Example – geo

$$
\begin{aligned}
&b := 1; \\
&x := 1; \\
&\mathtt{while}(b = 1)\ \{ \\
&\quad \mathtt{consume}(1); \\
&\quad x := x * 2; \\
&\quad b := \mathtt{ber}(1,1)\}
\end{aligned}
$$

Consider $\mathtt{C}_{\mathtt{geo}}$ denoting the body of geo and $I = [b = 1] \cdot 2$ the upper invariant:

$$
\begin{aligned}
&\mathsf{ect}[\mathtt{C}_{\mathtt{geo}}](I) = \\
&= \mathsf{ect}[\mathtt{consume}(1)](\mathsf{ect}[x := x * 2](\mathsf{ect}[b := \mathtt{ber}(1,1)](I))) \\
&= \mathsf{ect}[\mathtt{consume}(1)](\mathsf{ect}[x := x * 2](\frac{1}{2} \cdot I[b/1] + \frac{1}{2} \cdot I[b/0])) \\
&= \mathsf{ect}[\mathtt{consume}(1)](\frac{1}{2} \cdot I[b/1, x/2 \cdot x] + \frac{1}{2} \cdot I[b/0, x/2 \cdot x]) \\
&= 1 + \frac{1}{2} \cdot I[b/1, x/2 \cdot x] + \frac{1}{2} \cdot I[b/0, x/2 \cdot x] \\
&= 1 + \frac{1}{2} \cdot [1 = 1] \cdot 2 + \frac{1}{2} \cdot [0 = 1] \cdot 2 \\
&= 2
\end{aligned}
$$

Example – geo

$$
\begin{aligned}
&b := 1; \\
&x := 1; \\
&\mathtt{while}(b = 1)\ \{ \\
&\quad \mathtt{consume}(1); \\
&\quad x := x * 2; \\
&\quad b := \mathtt{ber}(1,1)\}
\end{aligned}
$$

Why Do We Need Modularity?

For two nested while loops $\texttt{while}(\phi)\ \{\texttt{while}(\psi)\ \{\texttt{C}\}\}$, we cannot synthesize the inner (I) and outer (O) upper invariant independently.

Why Do We Need Modularity?

For two nested while loops $\texttt{while}(\phi)\ \{\texttt{while}(\psi)\ \{\texttt{C}\}\}$, we cannot synthesize the inner (I) and outer (O) upper invariant independently. We get the following constraints in order to synthesize those:

$$\phi \models I \leq O \land \neg\phi \models f \leq O$$

Why Do We Need Modularity?

For two nested while loops $\texttt{while}(\phi)\ \{\texttt{while}(\psi)\ \{\texttt{C}\}\}$, we cannot synthesize the inner (I) and outer (O) upper invariant independently. We get the following constraints in order to synthesize those:

$$\phi \models I \leq O \land \neg\phi \models f \leq O \land \psi \models \mathsf{ect}[\texttt{C}](I) \leq I \land \neg\psi \models O \leq I$$

Why Do We Need Modularity?

For two nested while loops $\texttt{while}(\phi)\ \{\texttt{while}(\psi)\ \{\texttt{C}\}\}$, we cannot synthesize the inner (I) and outer (O) upper invariant independently. We get the following constraints in order to synthesize those:

$$\phi \models I \leq O \land \neg\phi \models f \leq O \land \psi \models \mathsf{ect}[\texttt{C}](I) \leq I \land \neg\psi \models O \leq I$$

Modularity

We achieve modularity by first calculating the cost of one loop iteration and then analyze how possible values in a state evolve during execution.

Separating Expected Cost and Value

First we have to separate the analysis into a cost and value analysis:

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

Separating Expected Cost and Value

First we have to separate the analysis into a cost and value analysis:

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

Modular Upper Invariants

$$1)\ \phi \models \mathtt{ecost}[\mathtt{C}] + \kappa(\mathsf{evalue}[\mathtt{C}](b_1), \ldots, \mathsf{evalue}[\mathtt{C}](b_n)) \leq \kappa(b_1, \ldots, b_n)$$
$$\Rightarrow \mathtt{ecost}[\mathtt{while}(\phi)\ \{\mathtt{C}\}] \leq \kappa(b_1, \ldots, b_n)$$

Separating Expected Cost and Value

First we have to separate the analysis into a cost and value analysis:

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

Modular Upper Invariants

1) $\phi \models \mathtt{ecost}[\mathtt{C}] + \kappa(\mathsf{evalue}[\mathtt{C}](b_1), \ldots, \mathsf{evalue}[\mathtt{C}](b_n)) \leq \kappa(b_1, \ldots, b_n)$
   $\Rightarrow \mathtt{ecost}[\mathtt{while}(\phi)\ \{\mathtt{C}\}] \leq \kappa(b_1, \ldots, b_n)$
2) $\phi \models \kappa(\mathsf{evalue}[\mathtt{C}](b_1), \ldots, \mathsf{evalue}[\mathtt{C}](b)_n)) \leq \kappa(b_1, \ldots, b_n) \wedge \neg\phi \models f \leq \kappa(b_1, \ldots, b_n)$
   $\Rightarrow \mathsf{evalue}[\mathtt{while}(\phi)\ \{\mathtt{C}\}](f) \leq \kappa(b_1, \ldots, b_n)$

Separating Expected Cost and Value

First we have to separate the analysis into a cost and value analysis:

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

Modular Upper Invariants

1) $\phi \models \mathtt{ecost}[\mathtt{C}] + \kappa(\mathsf{evalue}[\mathtt{C}](b_1), \ldots, \mathsf{evalue}[\mathtt{C}](b_n)) \leq \kappa(b_1, \ldots, b_n)$
   $\Rightarrow \mathtt{ecost}[\mathtt{while}(\phi)\ \{\mathtt{C}\}] \leq \kappa(b_1, \ldots, b_n)$
2) $\phi \models \kappa(\mathsf{evalue}[\mathtt{C}](b_1), \ldots, \mathsf{evalue}[\mathtt{C}](b)_n)) \leq \kappa(b_1, \ldots, b_n) \land \neg\phi \models f \leq \kappa(b_1, \ldots, b_n)$
   $\Rightarrow \mathsf{evalue}[\mathtt{while}(\phi)\ \{\mathtt{C}\}](f) \leq \kappa(b_1, \ldots, b_n)$

We need all this for the automation of ECT.

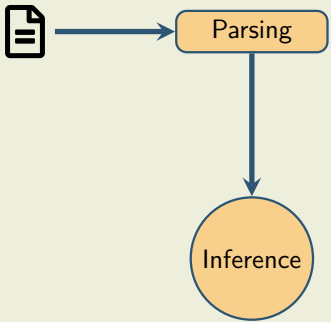
Overview

- Primer
- ERT
- ECT – Syntax & Semantic
- Automation
- Summary

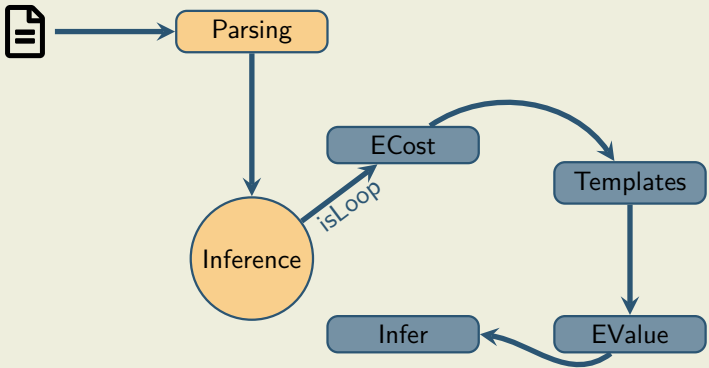
ecoimp


Parsing

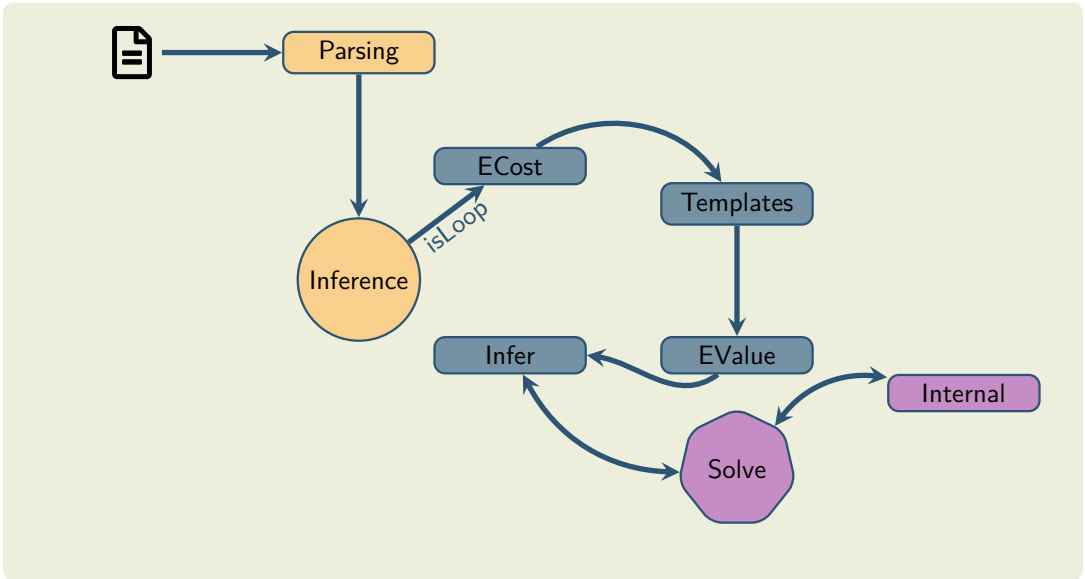
ecoimp



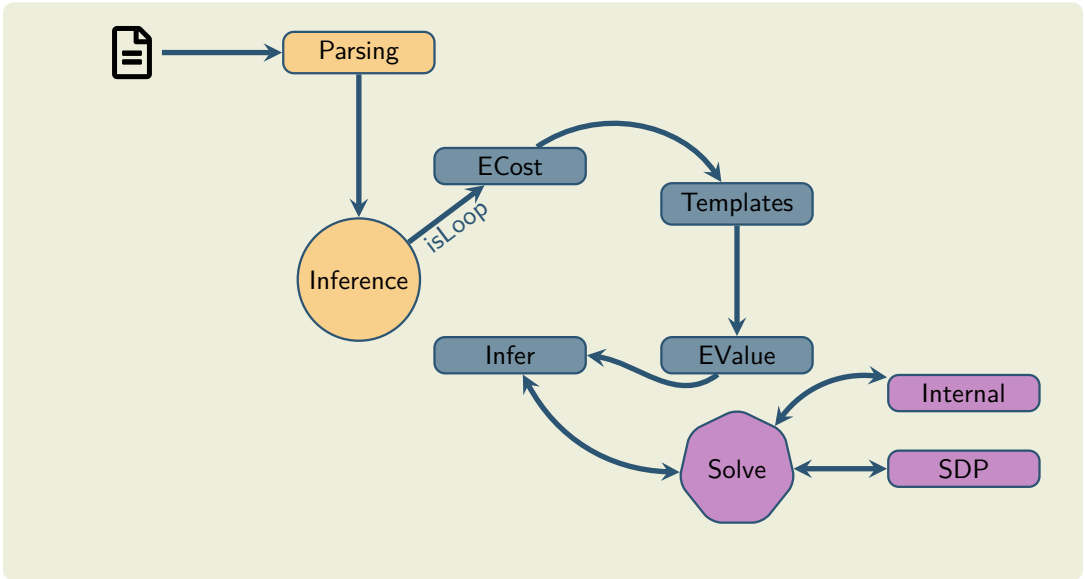
ecoimp



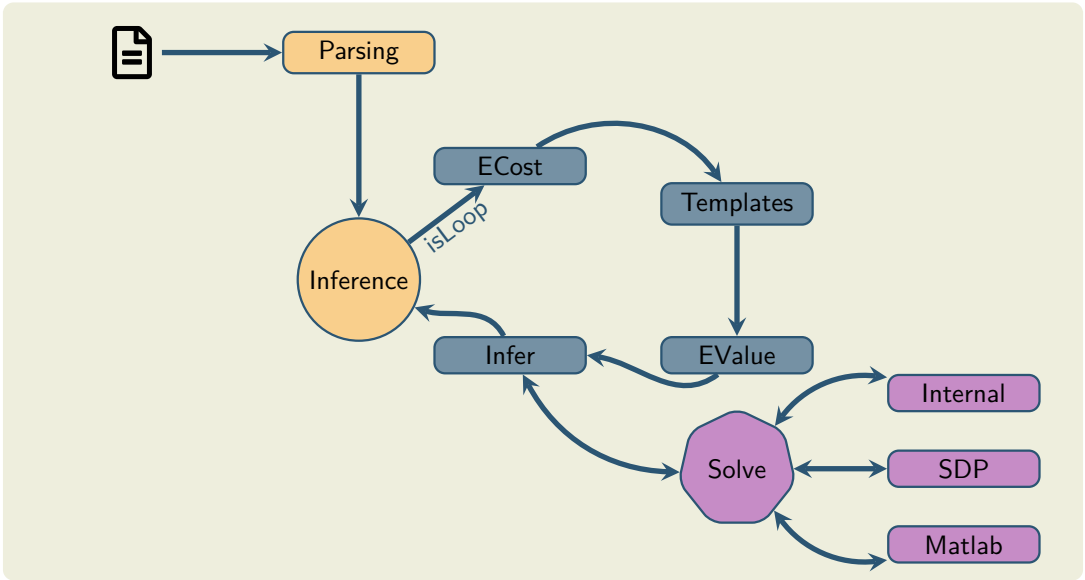
ecoimp


Parsing
ECost
Templates
isLoop
Inference
Infer
EValue
Internal
Solve

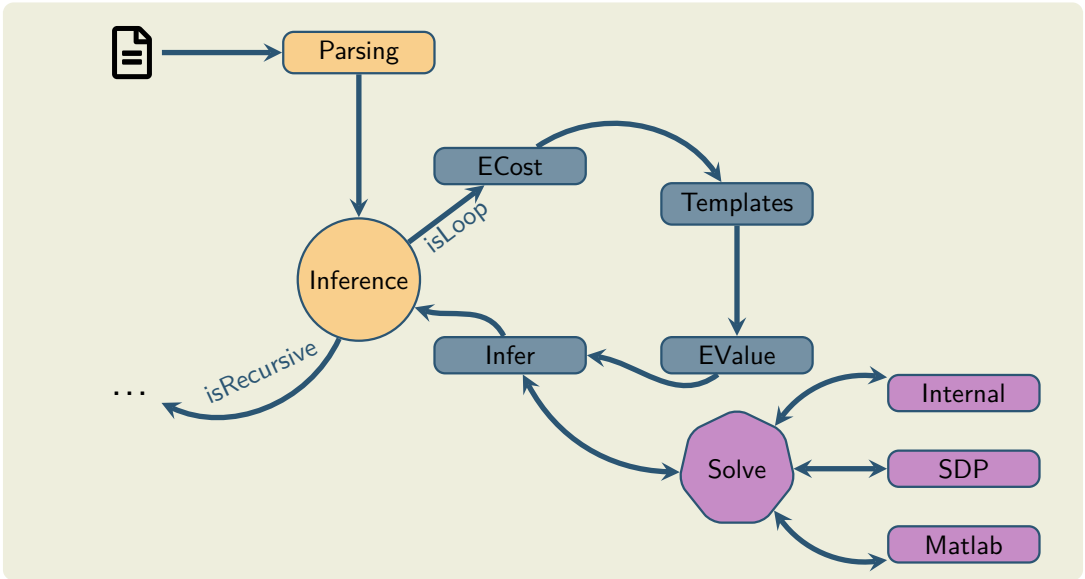
ecoimp


Parsing
ECost
Templates
isLoop
Inference
Infer
EValue
Internal
Solve
SDP

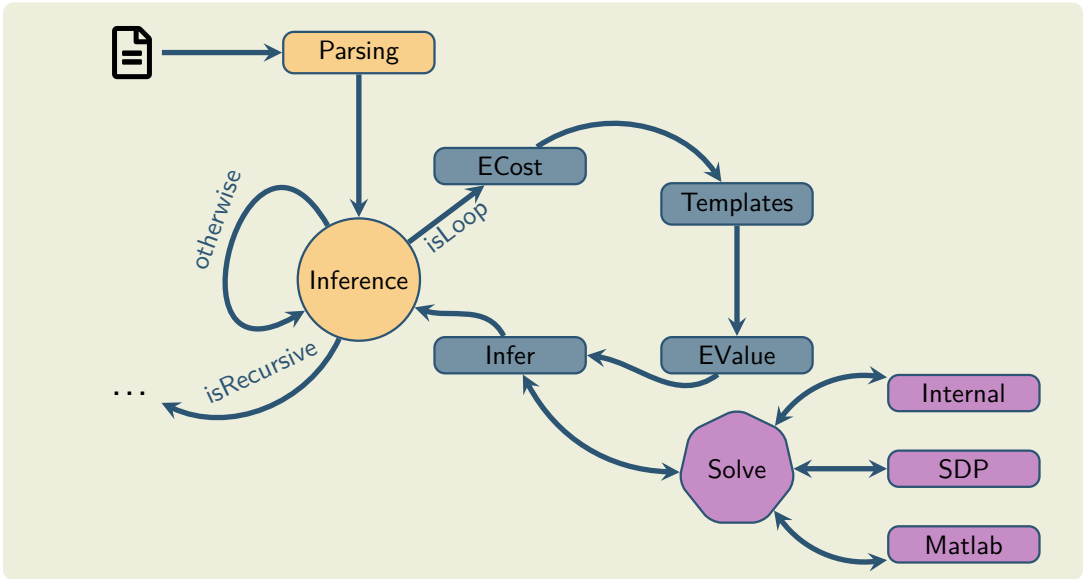
ecoimp


Parsing
ECost
Templates
isLoop
Inference
Infer
EValue
Internal
Solve
SDP
Matlab

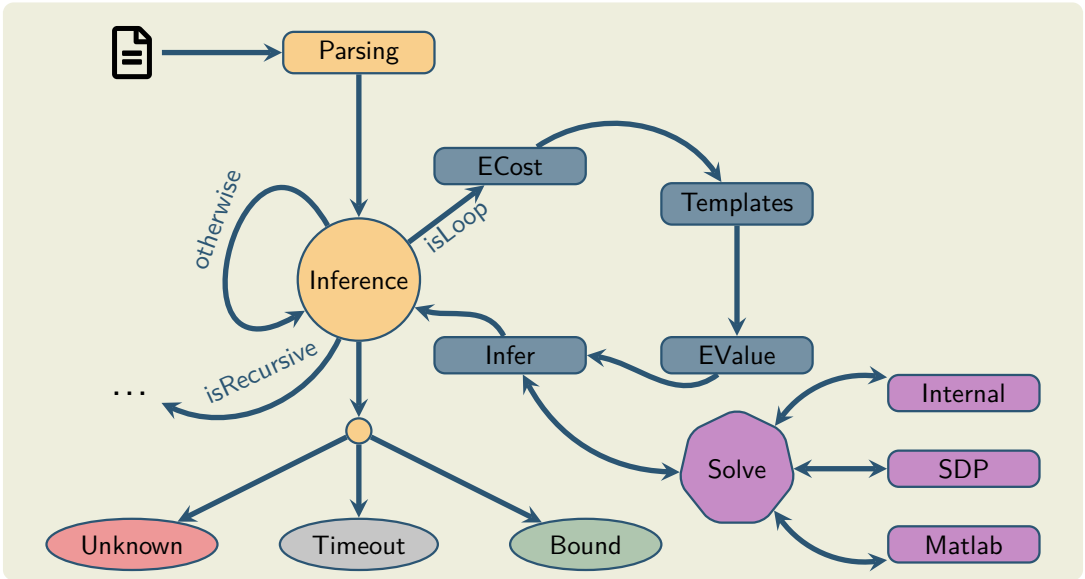
ecoimp


Parsing
ECost
Templates
isLoop
Inference
Infer
EValue
isRecursive
...
Internal
Solve
SDP
Matlab

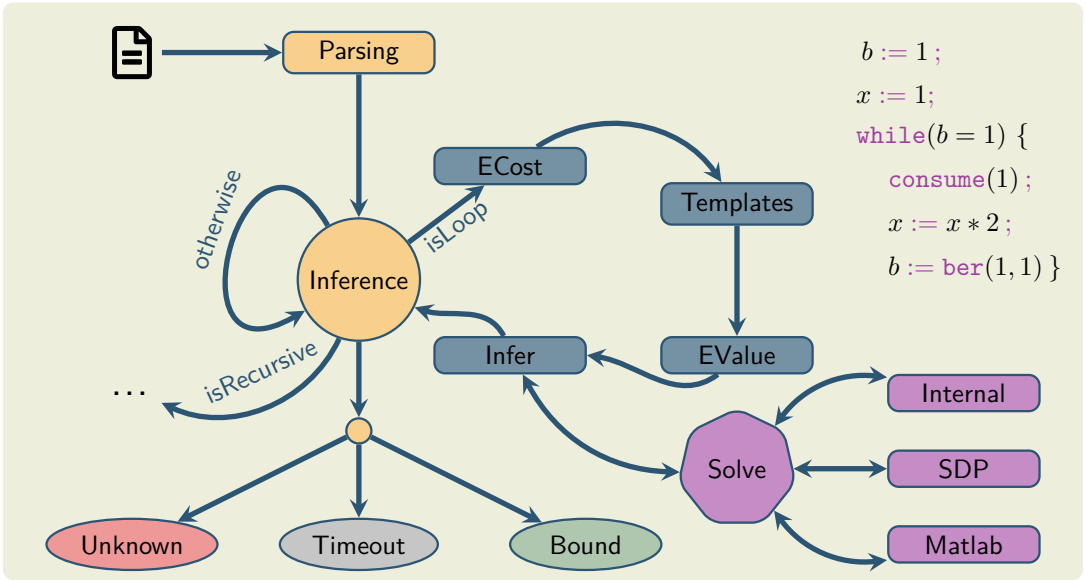
ecoimp


Parsing
ECost
Templates
otherwise
isLoop
Inference
Infer
EValue
isRecursive
...
Internal
Solve
SDP
Matlab

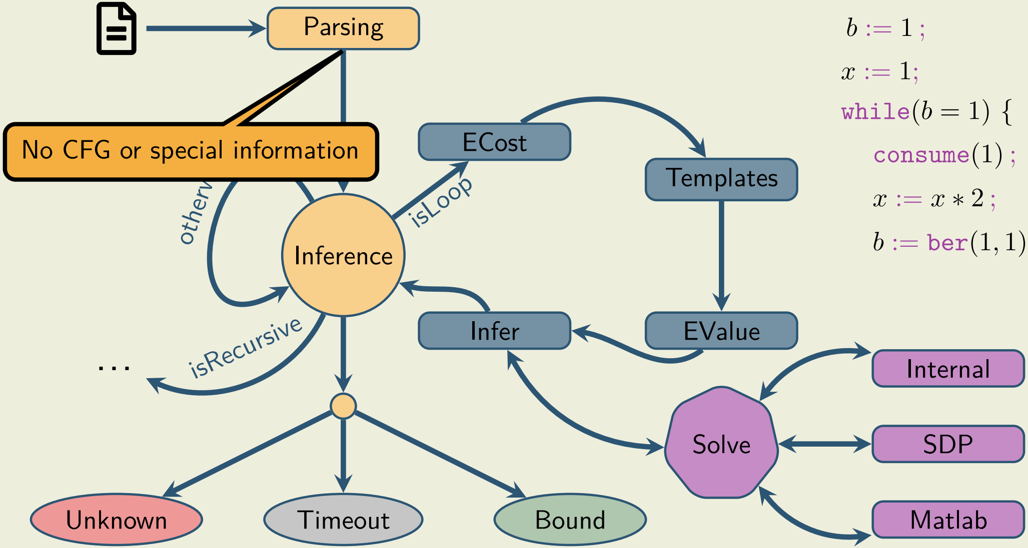
ecoimp


Parsing
ECost
Templates
otherwise
isLoop
Inference
...
isRecursive
Infer
EValue
Internal
Solve
SDP
Matlab
Unknown
Timeout
Bound

ecoimp


Parsing
ECost
Templates
otherwise
isLoop
Inference
isRecursive
...
Infer
EValue
Internal
Solve
SDP
Matlab
Unknown
Timeout
Bound
$b := 1\,;$
$x := 1;$
$\texttt{while}(b = 1)\ \{$
$\texttt{consume}(1)\,;$
$x := x * 2\,;$
$b := \texttt{ber}(1,1)\ \}$

ecoimp



$b := 1\,;$

$x := 1;$

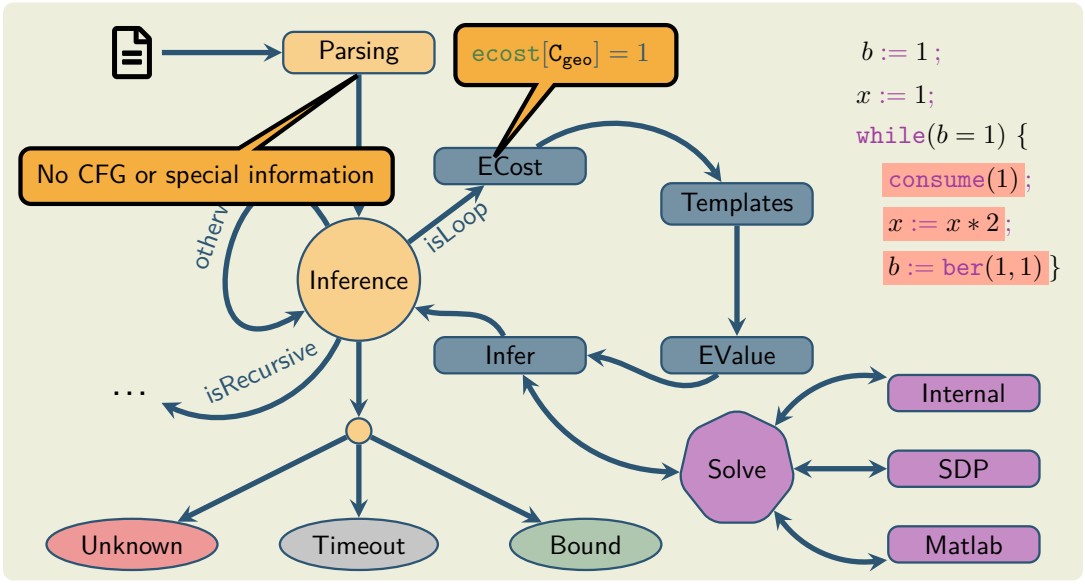
$\texttt{while}(b = 1)\ \{$

$\quad \texttt{consume}(1)\,;$

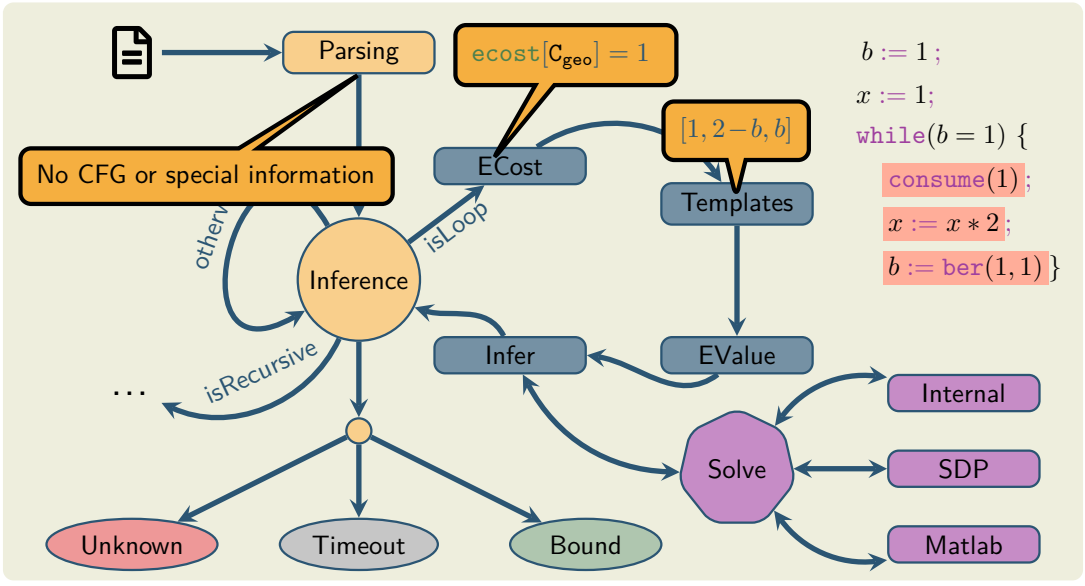
$\quad x := x * 2\,;$

$\quad b := \texttt{ber}(1,1)\ \}$

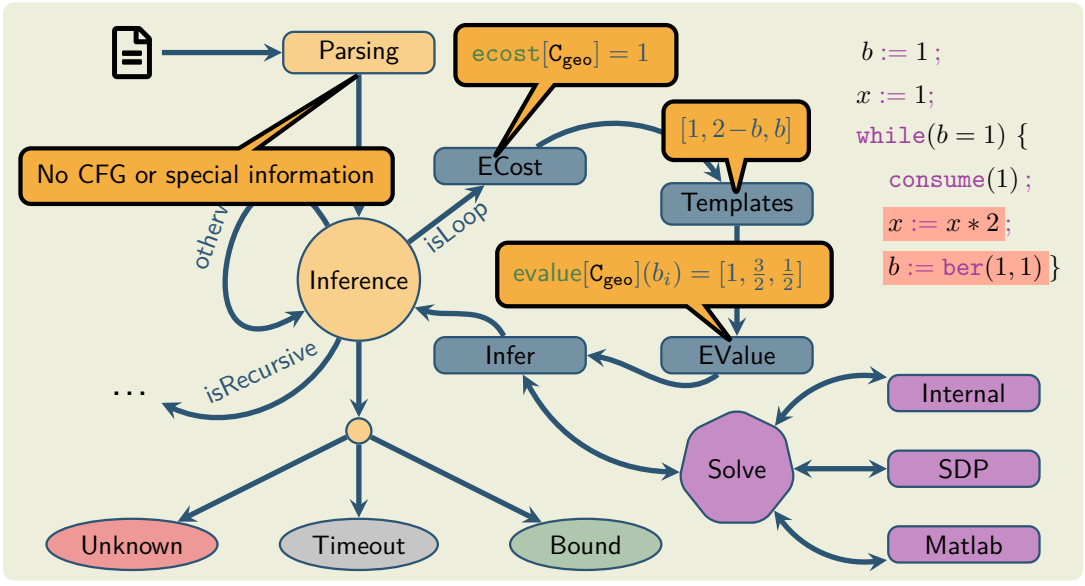
ecoimp


Parsing
ecost[C_geo] = 1
No CFG or special information
ECost
Templates
otherw
isLoop
Inference
isRecursive
...
Infer
EValue
Internal
Solve
SDP
Matlab
Unknown
Timeout
Bound
b := 1;
x := 1;
while(b = 1) {
consume(1);
x := x * 2;
b := ber(1,1)}

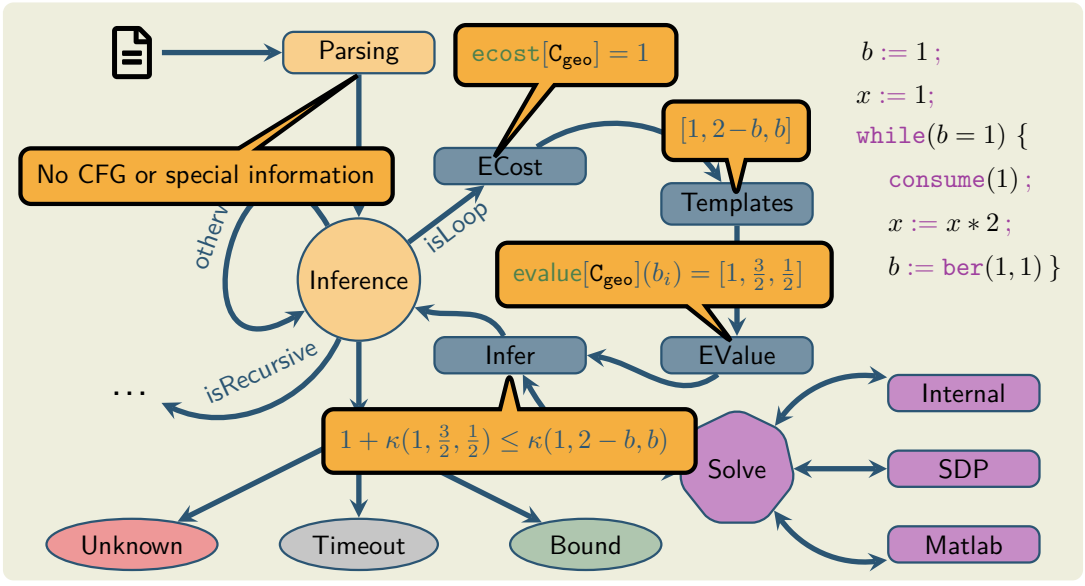
ecoimp


Parsing
ecost[C_geo] = 1
No CFG or special information
ECost
[1, 2−b, b]
Templates
otherw
isLoop
Inference
isRecursive
...
Infer
EValue
Internal
Solve
SDP
Matlab
Unknown
Timeout
Bound
b := 1;
x := 1;
while(b = 1) {
consume(1);
x := x ∗ 2;
b := ber(1, 1)}

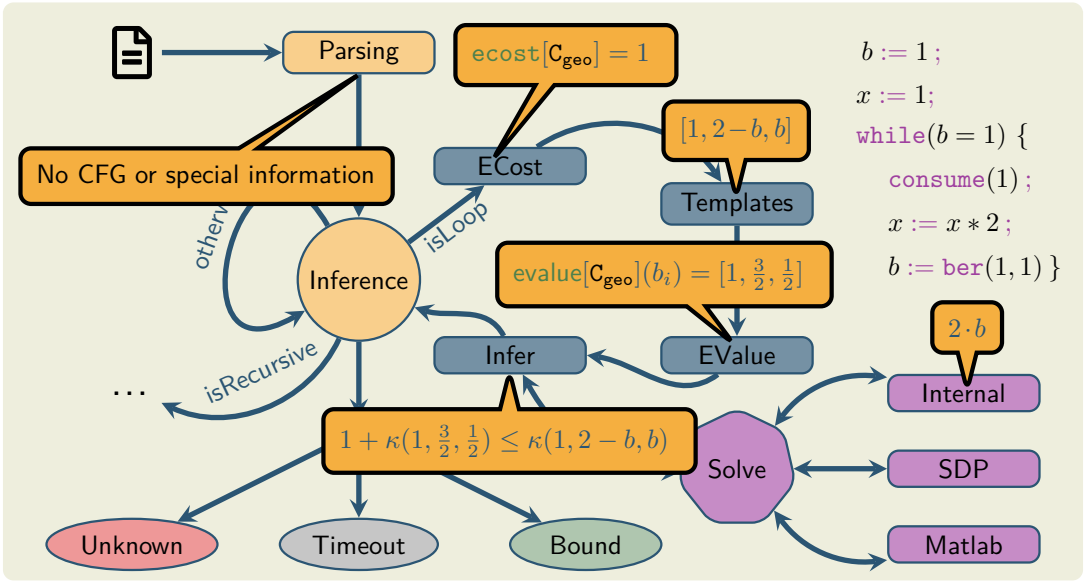
ecoimp

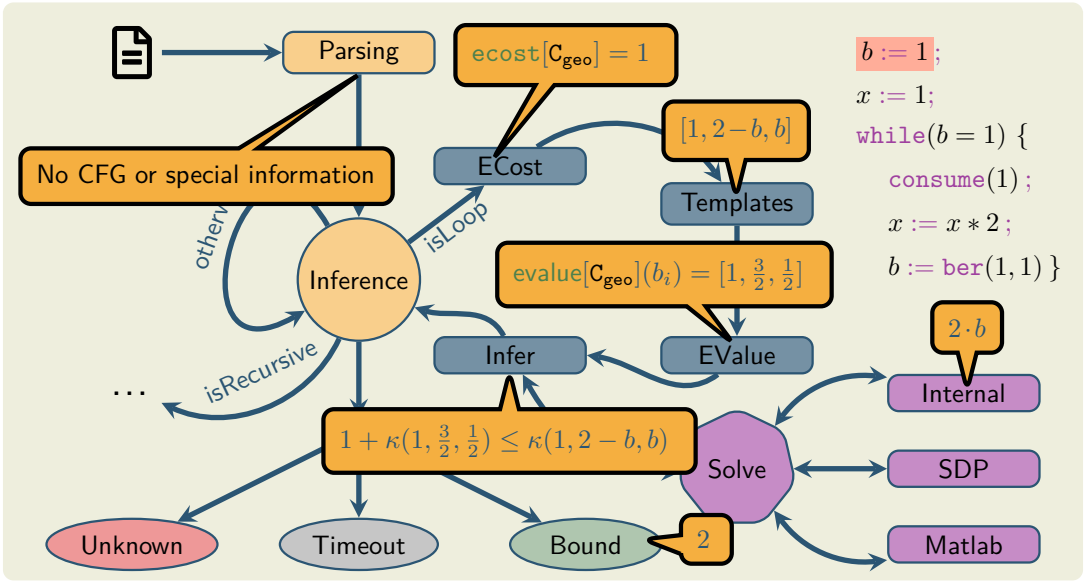

Parsing
ecost[C_geo] = 1
No CFG or special information
ECost
[1, 2−b, b]
Templates
otherw
isLoop
Inference
evalue[C_geo](b_i) = [1, 3/2, 1/2]
EValue
Infer
isRecursive
...
Internal
Solve
SDP
Matlab
Unknown
Timeout
Bound
b := 1 ;
x := 1;
while(b = 1) {
consume(1) ;
x := x ∗ 2;
b := ber(1, 1) }

ecoimp


Parsing
ecost[C_geo] = 1
No CFG or special information
ECost
[1, 2−b, b]
Templates
otherw
isLoop
Inference
evalue[C_geo](b_i) = [1, 3/2, 1/2]
Infer
EValue
isRecursive
...
1 + κ(1, 3/2, 1/2) ≤ κ(1, 2 − b, b)
Solve
Internal
SDP
Matlab
Unknown
Timeout
Bound
b := 1 ;
x := 1;
while(b = 1) {
consume(1) ;
x := x ∗ 2 ;
b := ber(1, 1) }

ecoimp


Parsing
ecost[C_geo] = 1
No CFG or special information
ECost
[1, 2−b, b]
Templates
otherw
isLoop
Inference
evalue[C_geo](b_i) = [1, 3/2, 1/2]
EValue
Infer
isRecursive
...
1 + κ(1, 3/2, 1/2) ≤ κ(1, 2 − b, b)
Solve
2·b
Internal
SDP
Matlab
Unknown
Timeout
Bound
b := 1 ;
x := 1;
while(b = 1) {
consume(1) ;
x := x ∗ 2 ;
b := ber(1, 1) }


Parsing
ecost[C_geo] = 1
No CFG or special information
ECost
[1, 2−b, b]
Templates
otherw
isLoop
Inference
evalue[C_geo](b_i) = [1, 3/2, 1/2]
EValue
Infer
isRecursive
...
1 + κ(1, 3/2, 1/2) ≤ κ(1, 2 − b, b)
Solve
Internal
SDP
Matlab
Unknown
Timeout
Bound
2
2·b
b := 1;
x := 1;
while(b = 1) {
consume(1);
x := x ∗ 2;
b := ber(1, 1) }

Summary

- static program analysis for probabilistic programs

Summary

- static program analysis for probabilistic programs
- ERT calculus for compositional analysis
- ECT calculus for compositional/modular analysis

Summary

- static program analysis for probabilistic programs
- ERT calculus for compositional analysis
- ECT calculus for compositional/modular analysis
- automation in ecoimp

Summary

- static program analysis for probabilistic programs
- ERT calculus for compositional analysis
- ECT calculus for compositional/modular analysis
- automation in ecoimp

Current/Future Research

- finishing recursion (modularity?)
- SDP solving

Summary

- static program analysis for probabilistic programs
- ERT calculus for compositional analysis
- ECT calculus for compositional/modular analysis
- automation in ecoimp

Current/Future Research

- finishing recursion (modularity?)
- SDP solving
- logarithmic bounds
- abstractions via coupling

Thank you for your attention!

References

Martin Avanzini, Georg Moser, and Michael Schaper.
A Modular Cost Analysis for Probabilistic Programs.
Proc. ACM Program. Lang., 4(OOPSLA), nov 2020.

E.W. Dijkstra.
Guarded Commands, Nondeterminacy and Formal Derivation of Programs.
Communications of the ACM, 1975.

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo.
Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms.
J. ACM, 65(5), aug 2018.