universität innsbruck

Expected Cost Analysis of Probabilistic Programs

Lunchtime Seminar

Martin Avanzini Georg Moser Michael Schaper Jonas Schöpf

February 03, 2022

Probabilistic Programming

Probabilistic Programming is a programming paradigm where probabilistic models can be specified and inference for these is done automatically. Languages in this class, e.g., incorporate random events as primitives or probabilistic branching.

Probabilistic Programming

Probabilistic Programming is a programming paradigm where probabilistic models can be specified and inference for these is done automatically. Languages in this class, e.g., incorporate random events as primitives or probabilistic branching.

Motivation

- model natural/physical processes $\Rightarrow$ "real" coin flip
- expressivity to model unavoidable application specifics (i.e. fault tolerance)

Probabilistic Programming

Probabilistic Programming is a programming paradigm where probabilistic models can be specified and inference for these is done automatically. Languages in this class, e.g., incorporate random events as primitives or probabilistic branching.

Motivation

- model natural/physical processes $\Rightarrow$ "real" coin flip
- expressivity to model unavoidable application specifics (i.e. fault tolerance)
- cryptography $\Rightarrow$ primality tests
- robotics/machine learning algorithms

Probabilistic Programming

Probabilistic Programming is a programming paradigm where probabilistic models can be specified and inference for these is done automatically. Languages in this class, e.g., incorporate random events as primitives or probabilistic branching.

Motivation

- model natural/physical processes ⇒ "real" coin flip
- expressivity to model unavoidable application specifics (i.e. fault tolerance)
- cryptography ⇒ primality tests
- robotics/machine learning algorithms
- improvement of algorithms, e.g., quicksort

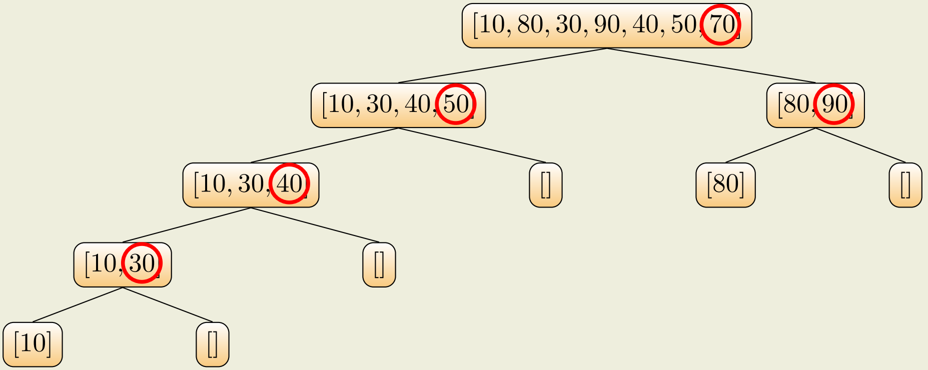
Motivation - Quicksort

- "standard" vs. randomized quicksort

Motivation - Quicksort

- "standard" vs. randomized quicksort

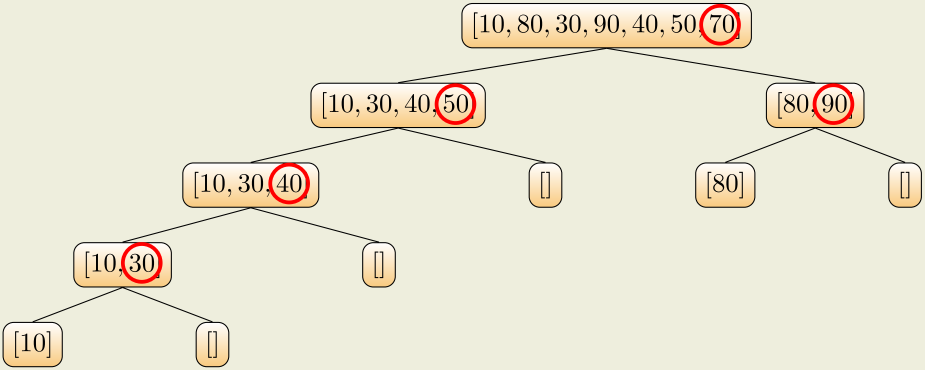
Example Quicksort



Motivation - Quicksort

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

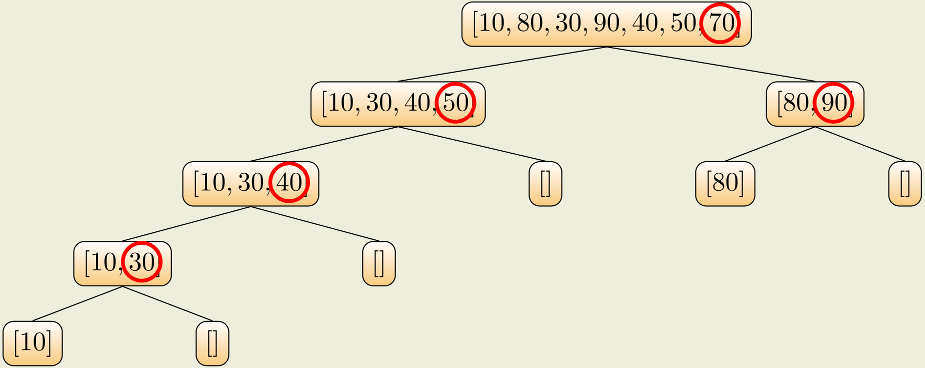
Example Quicksort



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



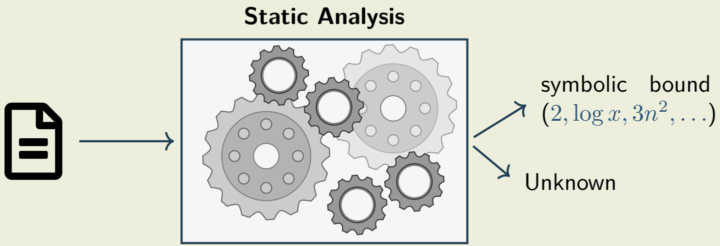
Overview

- Primer
- Syntax & Semantic
- Automation
- Constraint Solving
- Summary

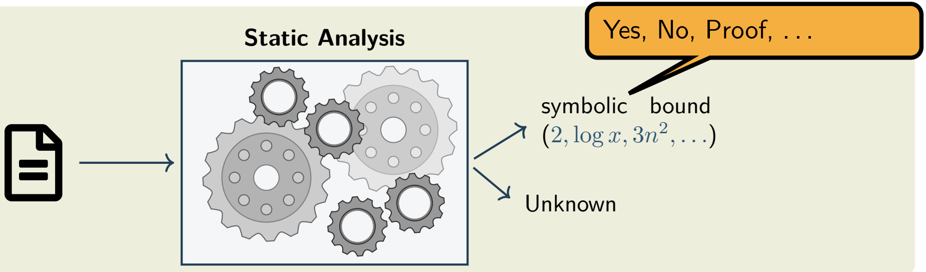
Overview

- Primer
- Syntax & Semantic
- Automation
- Constraint Solving
- Summary

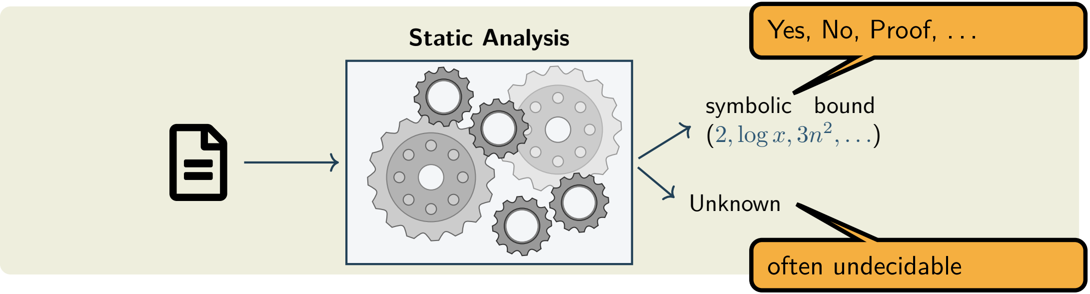
Static Resource Analysis



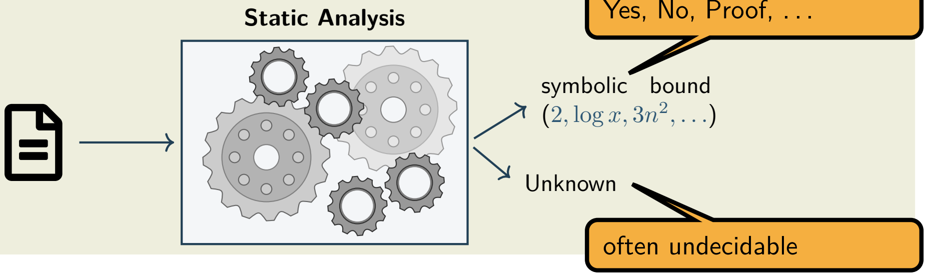
Static Resource Analysis



Static Resource Analysis



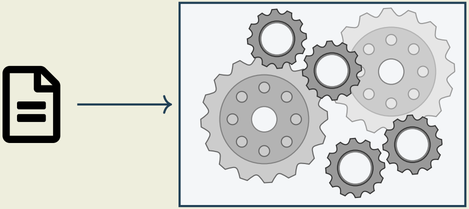
Static Resource Analysis



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

Static Resource Analysis

Static Analysis



symbolic bound $(2, \log x, 3n^2, \ldots)$

Yes, No, Proof, ...

Unknown

often undecidable

- 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

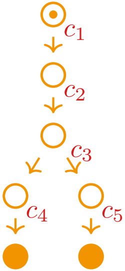
Dynamics

Semantics

Non-/Deterministic vs. Probabilistic

Non-/Determi. | Probabilistic

Dynamics

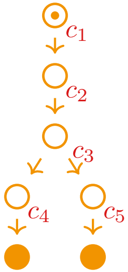


Semantics

Non-/Deterministic vs. Probabilistic

Non-/Determi. | Probabilistic

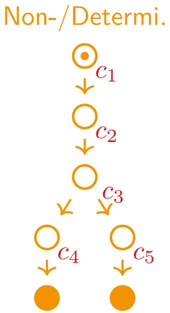
Dynamics

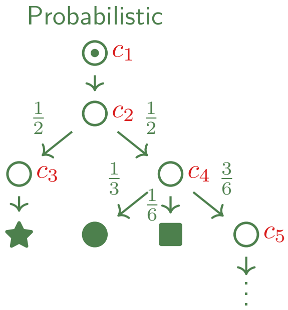


Semantics

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

Non-/Deterministic vs. Probabilistic





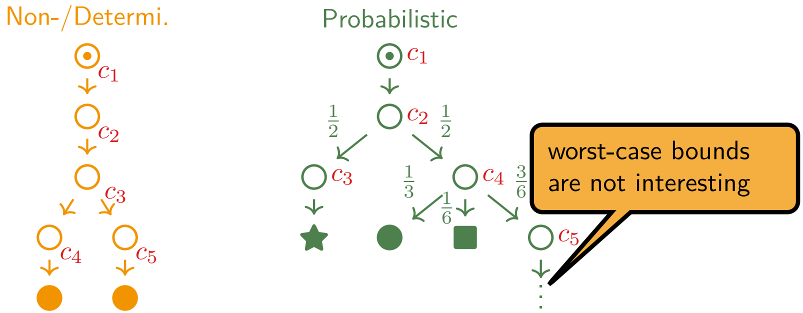
Dynamics

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

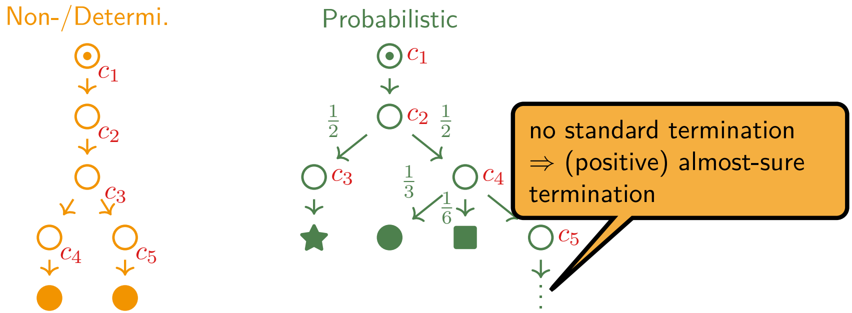


Semantics

- assign cost c_i to each operation
- overall cost is the sum of all operation costs
- deal with probabilities
- focus on average case complexity

Non-/Deterministic vs. Probabilistic

Dynamics



Semantics

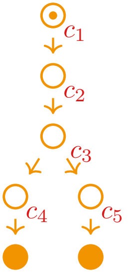
- assign cost c_i to each operation
- overall cost is the sum of all operation costs
- deal with probabilities

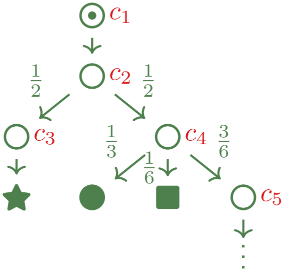
- focus on average case complexity
- program terminates with probability 1 (in a finite amount of time)

Non-/Deterministic vs. Probabilistic

Non-/Determi. | **Probabilistic**

Dynamics



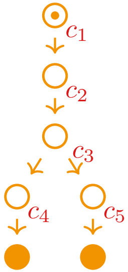
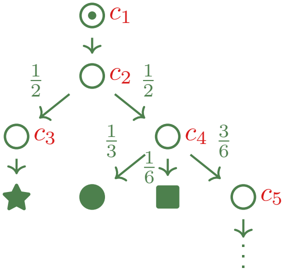


Semantics

$[\![P]\!](\odot) = \bullet$

- assign cost c_i to each operation
- overall cost is the sum of all operation costs
- deal with probabilities
- focus on average case complexity
- program terminates with probability 1 (in a finite amount of time)

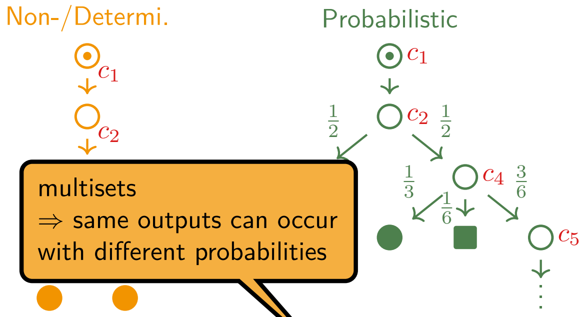
Non-/Deterministic vs. Probabilistic

	Non-/Determi.	Probabilistic
Dynamics		
Semantics	$[\![P]\!](\odot) = \bullet$	$[\![P]\!](\odot) = \{\!\!\{ \bigstar^{\frac{1}{2}}, \bullet^{\frac{1}{6}}, \blacksquare^{\frac{1}{12}}, \ldots \}\!\!\}$

- assign cost c_i to each operation
- overall cost is the sum of all operation costs
- deal with probabilities
- focus on average case complexity
- program terminates with probability 1 (in a finite amount of time)

Non-/Deterministic vs. Probabilistic

Dynamics



Semantics $[\![P]\!](\odot) = \bullet$ $\quad [\![P]\!](\odot) = \{\!\# \bigstar^{\frac{1}{2}}, \bullet^{\frac{1}{6}}, \blacksquare^{\frac{1}{12}}, \dots \#\!\}$

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

- focus on average case complexity
- program terminates with probability 1 (in a finite amount of time)

Overview

- Primer
- Syntax & Semantic
- Automation
- Constraint Solving
- 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

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

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

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

Probabilistic While (pWhile)

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

Syntax of pWhile

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

Probabilistic While ($\texttt{pWhile}$)

- 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 $\texttt{pWhile}$

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

Probabilistic While (`pWhile`)

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

Syntax of `pWhile`

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

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

Probabilistic While (pWhile)

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

Syntax of pWhile

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

Example – geo

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

Probabilistic While (pWhile)

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

Syntax of pWhile

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

Example – geo

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

$\mathsf{ect}[\mathsf{geo}](0) =$

Probabilistic While (pWhile)

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

Syntax of pWhile

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

Example – geo

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

$$\mathsf{ect}[\mathsf{geo}](0) = 1 + \frac{1}{2} \cdot (1 + \frac{1}{2} \cdot (1 + \ldots$$

Probabilistic While (pWhile)

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

Syntax of pWhile

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

Example – geo

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

$$
\begin{aligned}
\mathsf{ect}[\mathsf{geo}](0) &= 1 + \frac{1}{2} \cdot (1 + \frac{1}{2} \cdot (1 + \ldots \\
&= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots
\end{aligned}
$$

Probabilistic While (pWhile)

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

Syntax of pWhile

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

Example – geo

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

$$
\begin{aligned}
\mathsf{ect}[\mathsf{geo}](0) &= 1 + \frac{1}{2} \cdot (1 + \frac{1}{2} \cdot (1 + \dots \\
&= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots = 2
\end{aligned}
$$

Experiments Stable Version ecoimp

Problem	ecoimp	Absynth	Wang et al. 2019
linear			
2drwalk	0.026	0.286	−−
bayesian_network	0.002	0.127	−−
ber	0.001	0.014	6.684
C4B_t13	0.005	0.025	8.527
non-linear			
2drobot	1.760	−−	11.621
queueing-network	2.215	1.286	78.191
nest-4	0.554	−−	−−
trader-10	0.025	3.638	10.460
trader-20	0.030	119.464	10.420
trader-100000	2.113	−−	20.332
coupons-n	0.195	−−	−−

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.

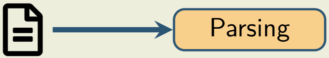
C	$\mathsf{ect}[\mathtt{C}](f)$	$\mathsf{evalue}[\mathtt{C}](f)$
$\mathtt{consume}(e)$	$\langle e\rangle + f$	f
$\mathtt{skip}$	f	f
$\mathtt{abort}$	0	0
$x := d$	$\lambda\sigma.\mathbb{E}_{d(\sigma)}(\lambda v.f[x/v](\sigma))$	$\lambda\sigma.\mathbb{E}_{d(\sigma)}(\lambda v.f[x/v](\sigma))$
$\mathtt{C};\mathtt{D}$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

Overview

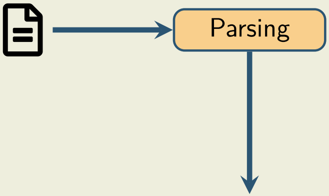
- Primer
- Syntax & Semantic
- Automation
- Constraint Solving
- Summary

ecoimp

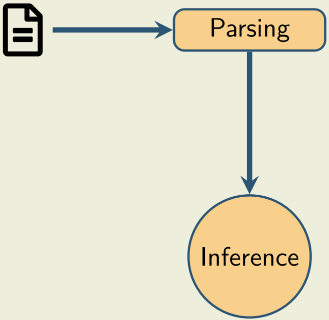
ecoimp



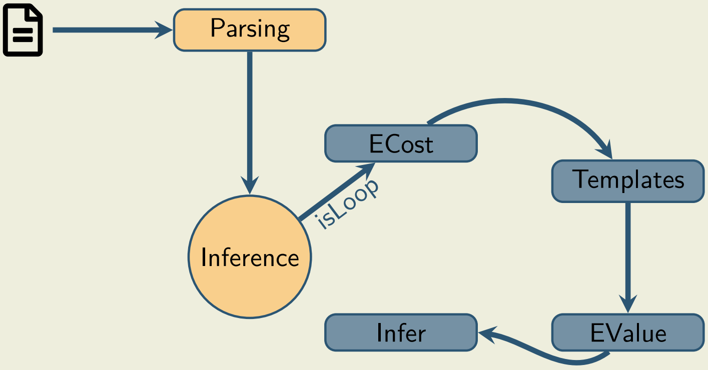
ecoimp



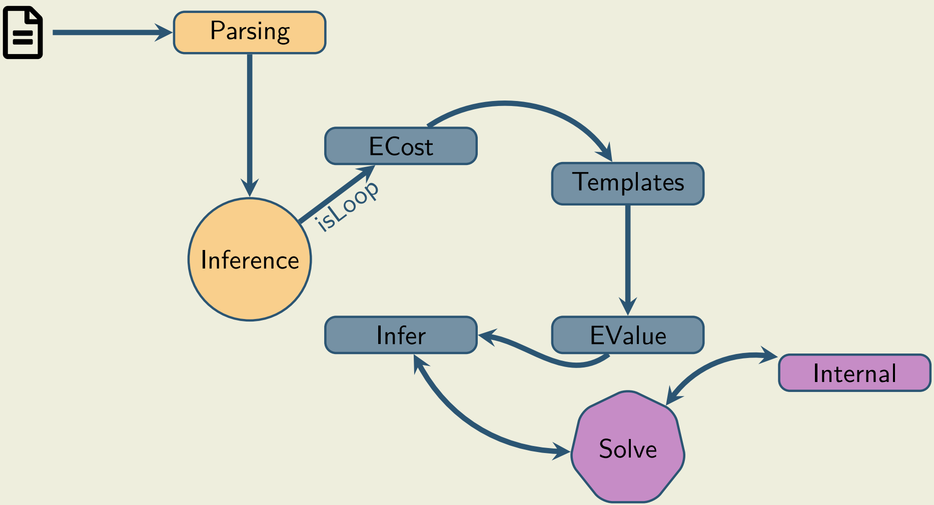
ecoimp



ecoimp

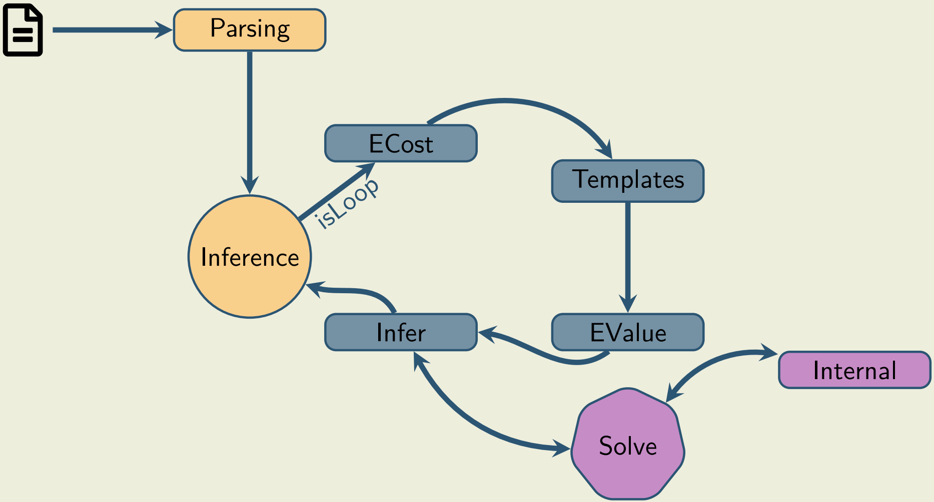


ecoimp



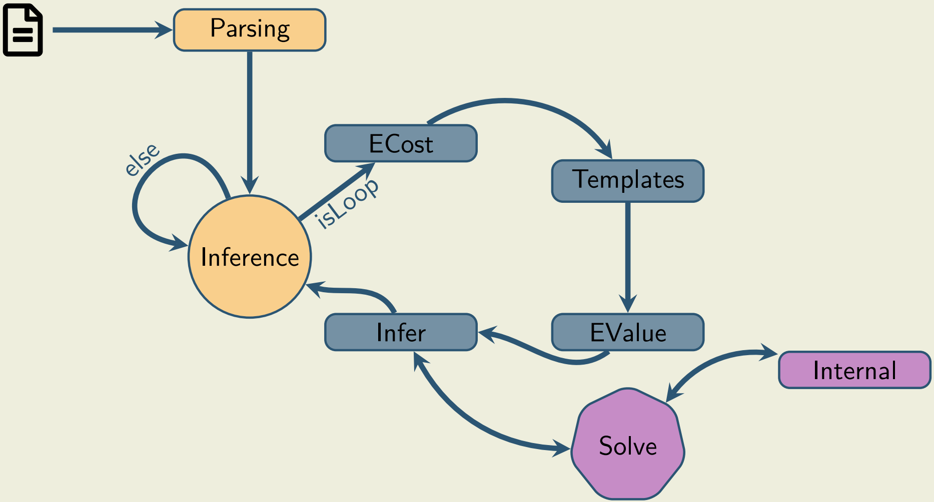
Parsing
Inference
isLoop
ECost
Templates
EValue
Infer
Internal
Solve

ecoimp



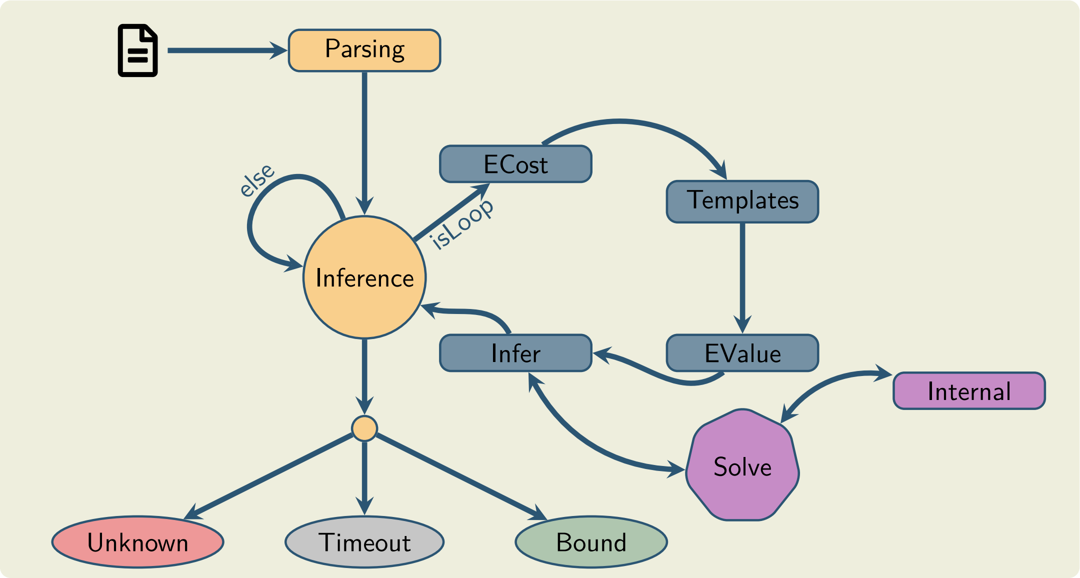
Parsing
ECost
Templates
isLoop
Inference
Infer
EValue
Internal
Solve

ecoimp



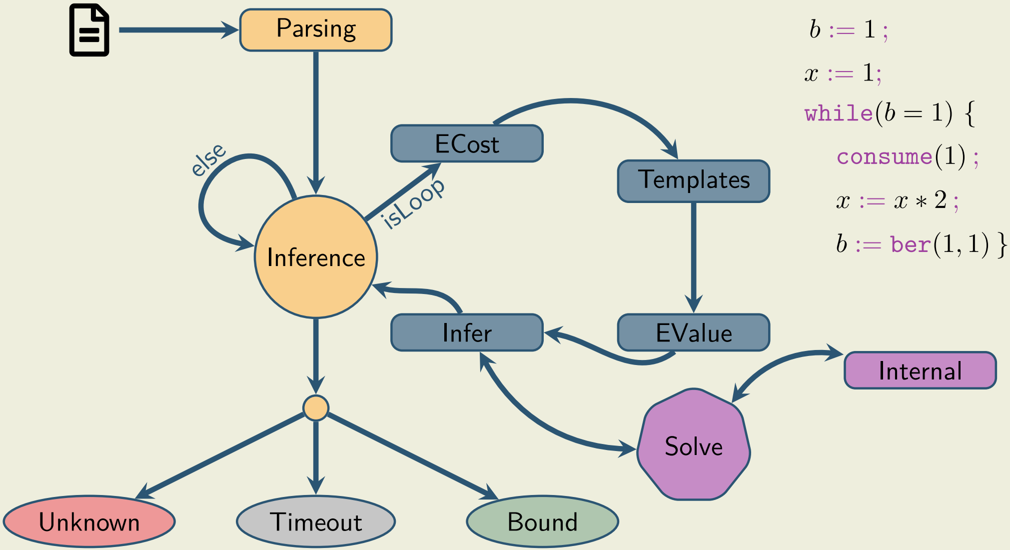
Parsing
else
Inference
isLoop
ECost
Templates
EValue
Infer
Internal
Solve

ecoimp



Parsing
else
ECost
isLoop
Templates
Inference
Infer
EValue
Internal
Solve
Unknown
Timeout
Bound

ecoimp



$b := 1\,;$

$x := 1;$

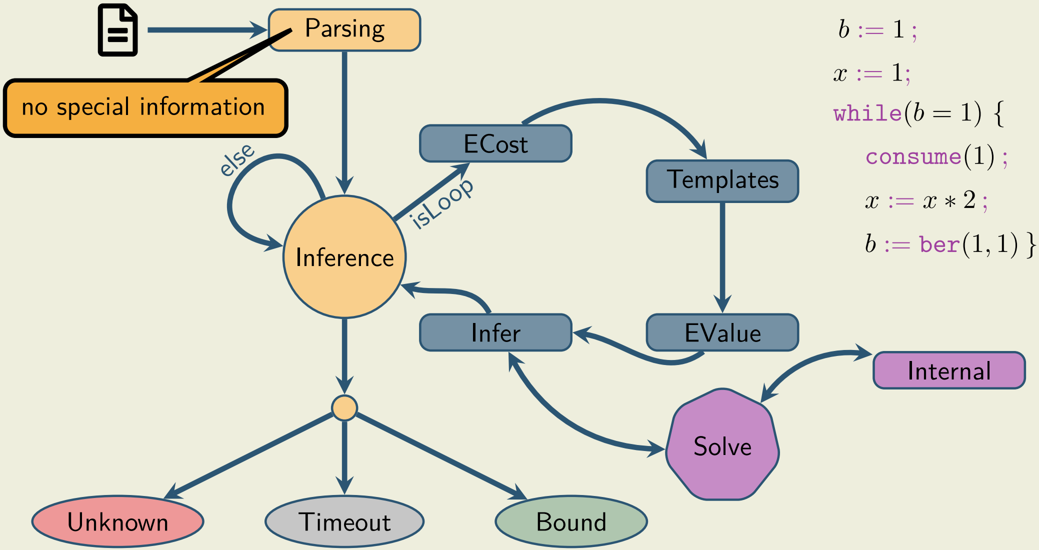
$\texttt{while}(b = 1)\ \{$

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

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

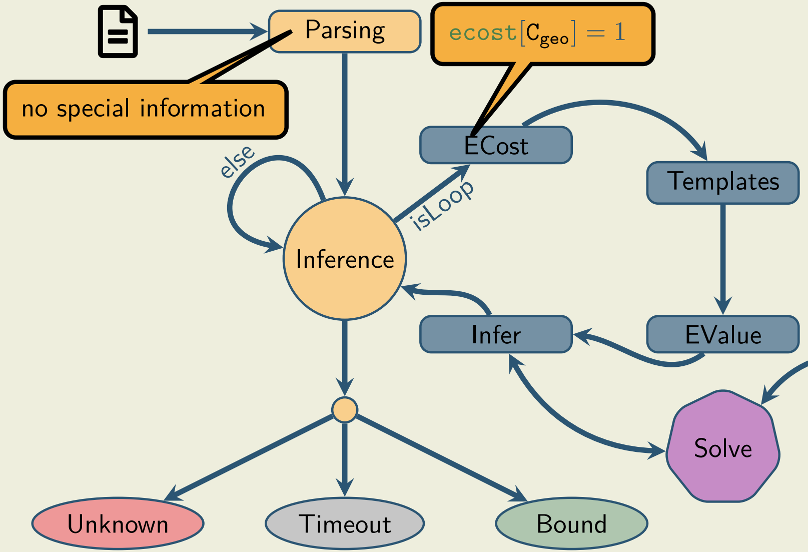
$\quad b := \texttt{ber}(1, 1)\ \}$

ecoimp



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

ecoimp



$b := 1\,;$

$x := 1;$

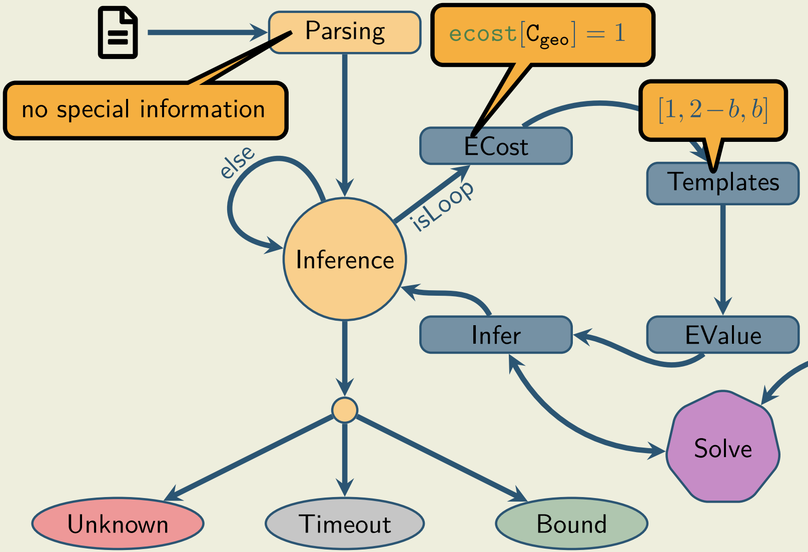
$\texttt{while}(b = 1)\ \{$

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

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

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

ecoimp



$b := 1;$

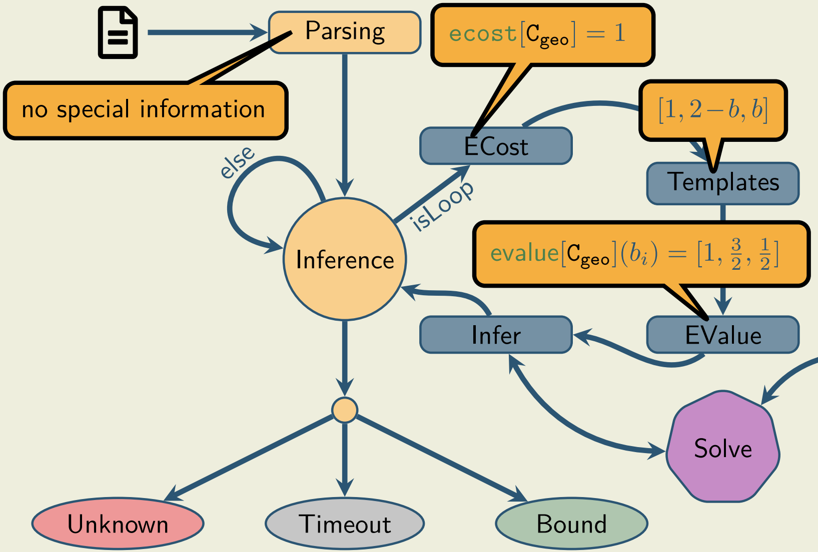
$x := 1;$

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

$\texttt{consume}(1);$

$x := x * 2;$

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



$b := 1;$

$x := 1;$

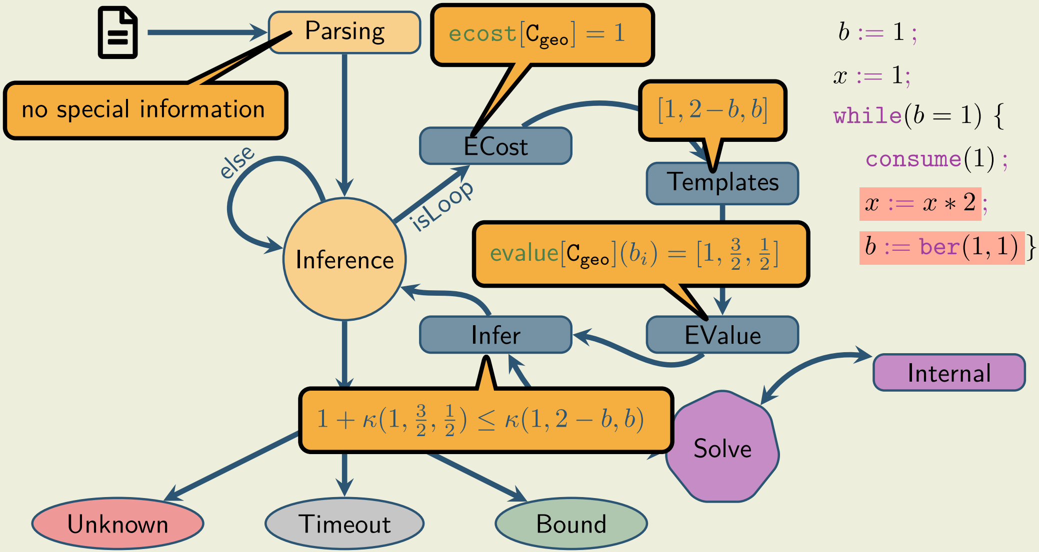
$\texttt{while}(b = 1)$ {

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

$\quad x := x * 2;$

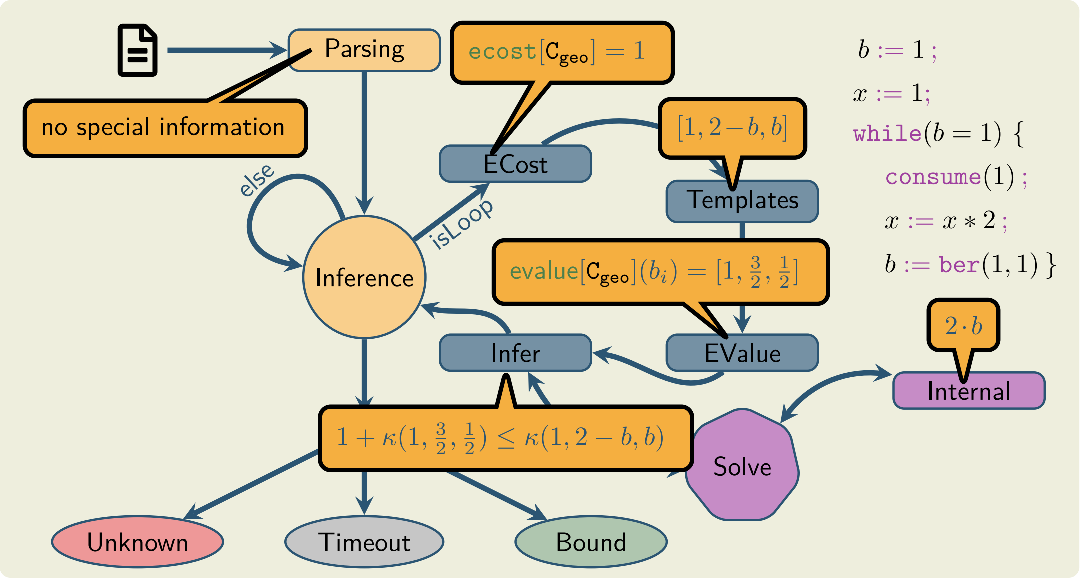
$\quad b := \texttt{ber}(1, 1)$ }

ecoimp

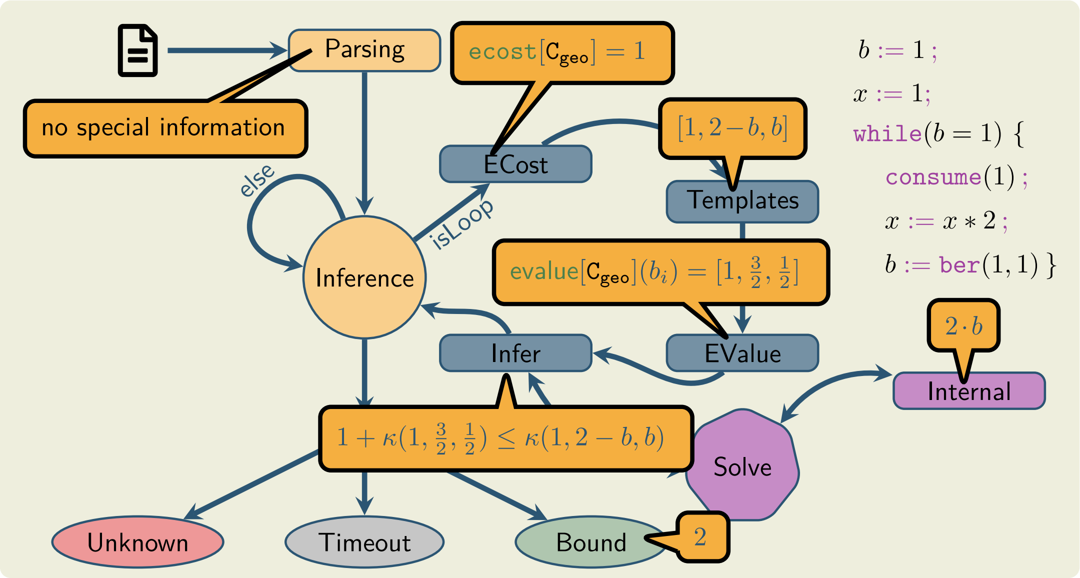



```
b := 1 ;
x := 1;
while(b = 1) {
   consume(1) ;
   x := x * 2;
   b := ber(1, 1) }
```

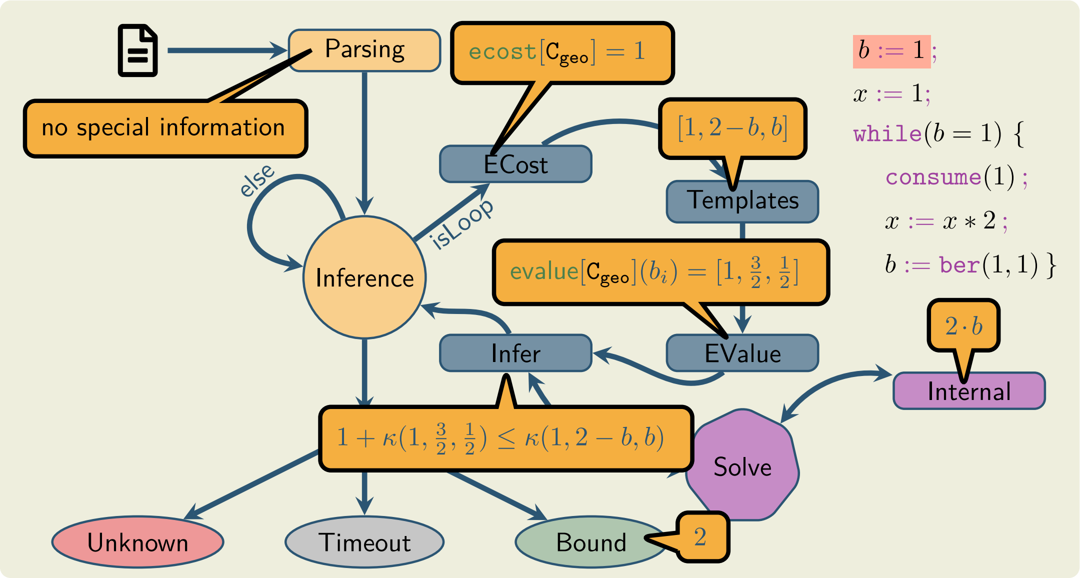
ecoimp


Parsing
no special information
$\texttt{ecost}[\mathsf{C}_{\mathsf{geo}}] = 1$
ECost
$[1, 2-b, b]$
Templates
else
isLoop
Inference
$\texttt{evalue}[\mathsf{C}_{\mathsf{geo}}](b_i) = [1, \frac{3}{2}, \frac{1}{2}]$
Infer
EValue
$2 \cdot b$
Internal
$1 + \kappa(1, \frac{3}{2}, \frac{1}{2}) \leq \kappa(1, 2-b, b)$
Solve
Unknown
Timeout
Bound
$b := 1\,;$
$x := 1;$
$\texttt{while}(b = 1)\ \{$
$\texttt{consume}(1)\,;$
$x := x * 2\,;$
$b := \texttt{ber}(1,1)\ \}$

ecoimp



Parsing
no special information
$\texttt{ecost}[\mathsf{C_{geo}}] = 1$
$[1, 2-b, b]$
ECost
Templates
else
isLoop
Inference
$\texttt{evalue}[\mathsf{C_{geo}}](b_i) = [1, \frac{3}{2}, \frac{1}{2}]$
Infer
EValue
$2 \cdot b$
Internal
$1 + \kappa(1, \frac{3}{2}, \frac{1}{2}) \leq \kappa(1, 2-b, b)$
Solve
Unknown
Timeout
Bound
2
$b := 1\,;$
$x := 1;$
$\texttt{while}(b = 1)\ \{$
$\texttt{consume}(1)\,;$
$x := x * 2\,;$
$b := \texttt{ber}(1,1)\ \}$


Parsing
no special information
$\texttt{ecost}[\mathsf{C_{geo}}] = 1$
$[1, 2-b, b]$
ECost
Templates
else
isLoop
Inference
$\texttt{evalue}[\mathsf{C_{geo}}](b_i) = [1, \frac{3}{2}, \frac{1}{2}]$
Infer
EValue
$2 \cdot b$
Internal
$1 + \kappa(1, \frac{3}{2}, \frac{1}{2}) \leq \kappa(1, 2-b, b)$
Solve
Unknown
Timeout
Bound
2
$b := 1$;
$x := 1$;
while$(b = 1)$ {
consume(1) ;
$x := x * 2$;
$b :=$ ber$(1,1)$ }

Recursion

We extend the syntax of `pWhile` to function definitions and a call statement to a function. A program is now a sequence of functions.

$$\mathtt{F} ::= \mathtt{def}\ fun\ \mathtt{:}\ \{\mathtt{C}\} \qquad\qquad \mathtt{call}\ fun$$

Recursion

We extend the syntax of `pWhile` to function definitions and a call statement to a function. A program is now a sequence of functions.

$$\mathtt{F} ::= \mathtt{def}\ fun : \{\mathtt{C}\} \qquad\qquad \mathtt{call}\ fun$$

This is semantically interpreted as:

- the entry point to a program is the `main` function

Recursion

We extend the syntax of pWhile to function definitions and a call statement to a function. A program is now a sequence of functions.

$$\texttt{F} ::= \texttt{def}\ fun : \{\texttt{C}\} \qquad\qquad \texttt{call}\ fun$$

This is semantically interpreted as:

- the entry point to a program is the main function
- a function is analyzed based on the SCC analysis

Recursion

We extend the syntax of `pWhile` to function definitions and a call statement to a function. A program is now a sequence of functions.

$$\texttt{F} ::= \texttt{def}\ fun : \{\texttt{C}\} \qquad\qquad \texttt{call}\ fun$$

This is semantically interpreted as:

- the entry point to a program is the `main` function
- a function is analyzed based on the SCC analysis
- we extend our theory to handle recursive calls

Recursion

We extend the syntax of `pWhile` to function definitions and a call statement to a function. A program is now a sequence of functions.

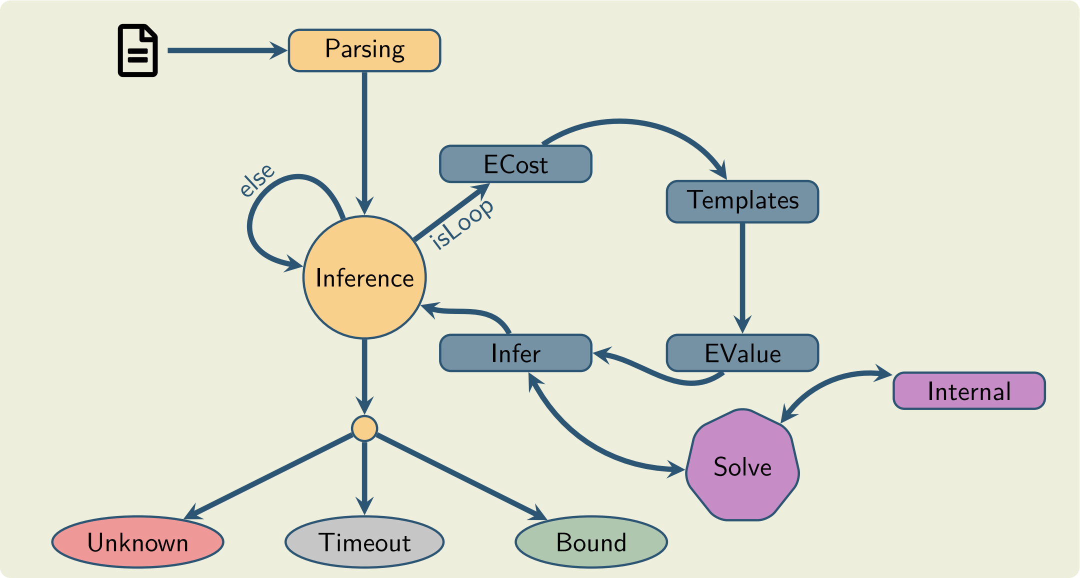
$$\texttt{F} ::= \texttt{def}\ fun : \{\texttt{C}\} \qquad\qquad \texttt{call}\ fun$$

This is semantically interpreted as:

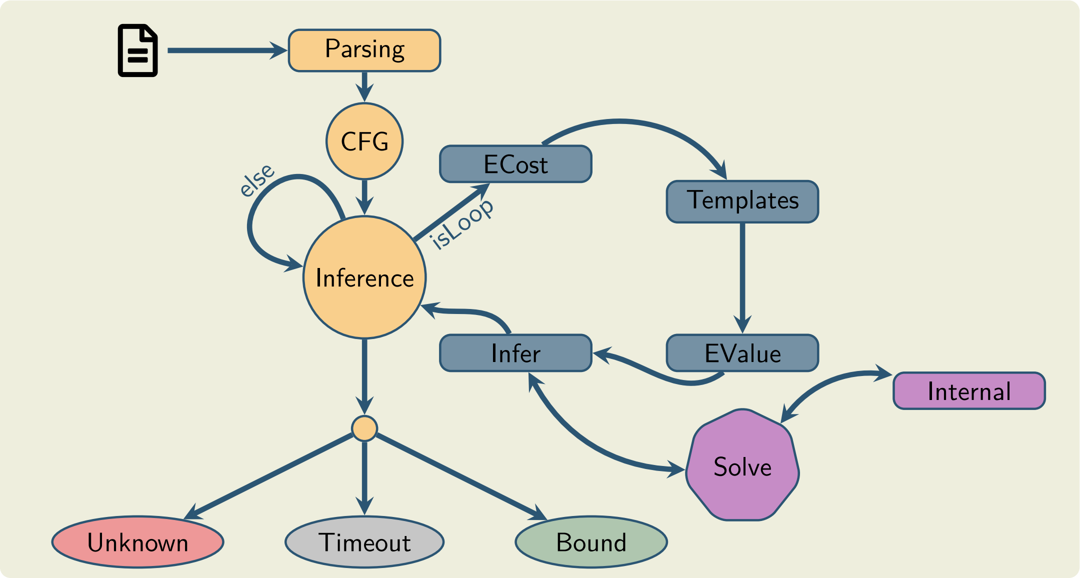
- the entry point to a program is the `main` function
- a function is analyzed based on the SCC analysis
- we extend our theory to handle recursive calls

```
def geo : {
  consume(1);
  b := ber(1,1);
  if(b = 1) {
    x := x * 2;
    call geo
  } {
    skip
  }}
```

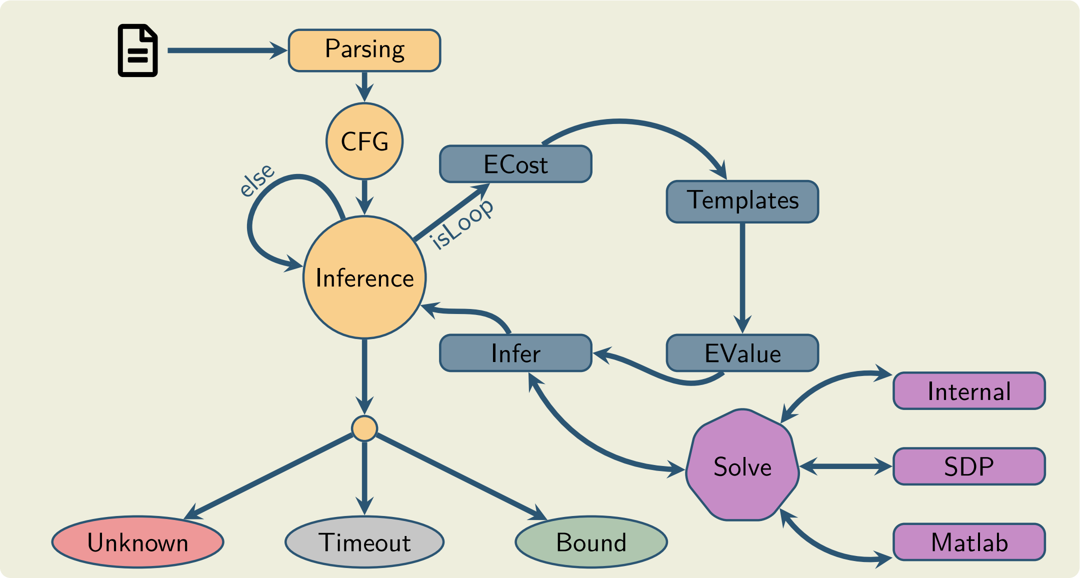
ecoimp


Parsing
else
ECost
Templates
isLoop
Inference
Infer
EValue
Internal
Solve
Unknown
Timeout
Bound

ecoimp

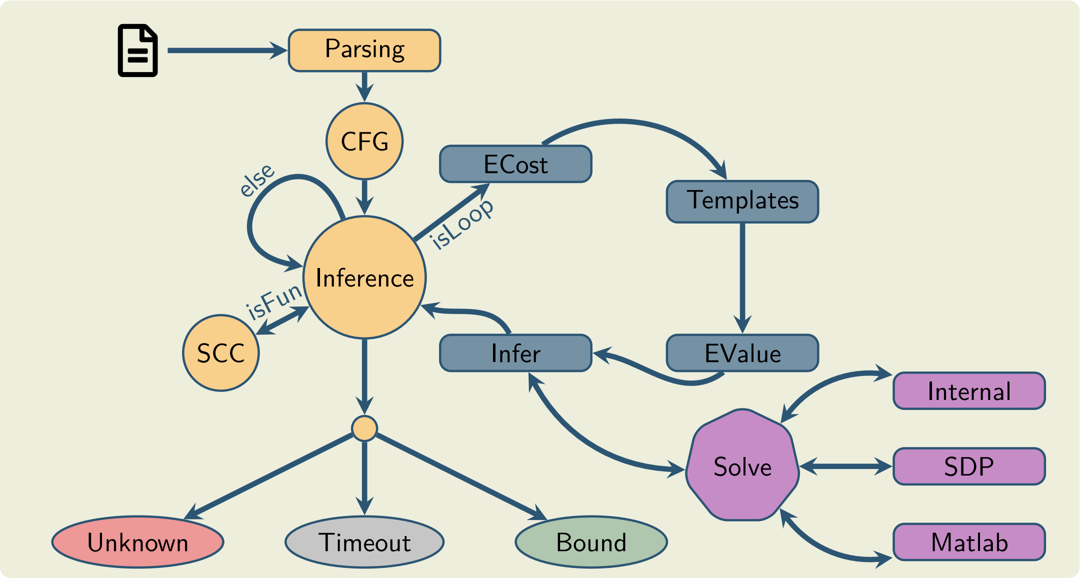

Parsing
CFG
else
Inference
isLoop
ECost
Templates
EValue
Infer
Internal
Solve
Unknown
Timeout
Bound

ecoimp

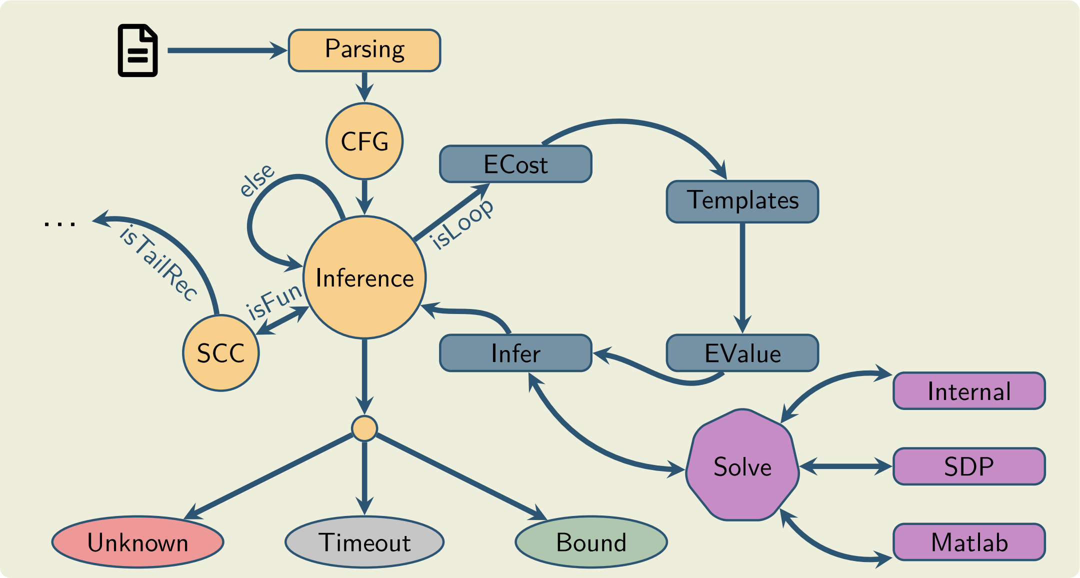


Parsing
CFG
else
Inference
isLoop
ECost
Templates
EValue
Infer
Internal
Solve
SDP
Matlab
Unknown
Timeout
Bound

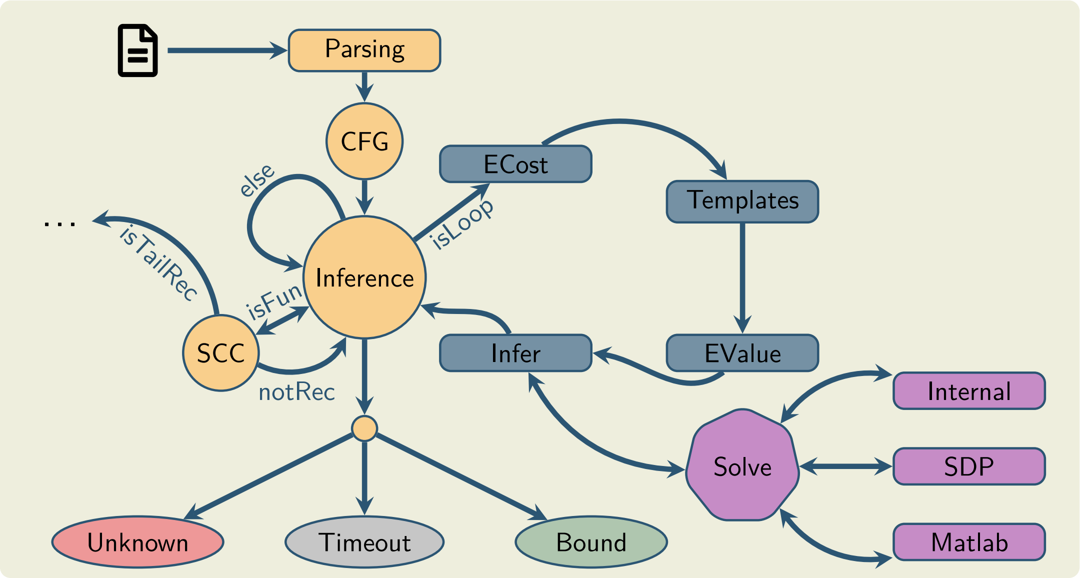
ecoimp


Parsing
CFG
else
Inference
isLoop
ECost
Templates
isFun
SCC
Infer
EValue
Internal
Solve
SDP
Matlab
Unknown
Timeout
Bound

ecoimp

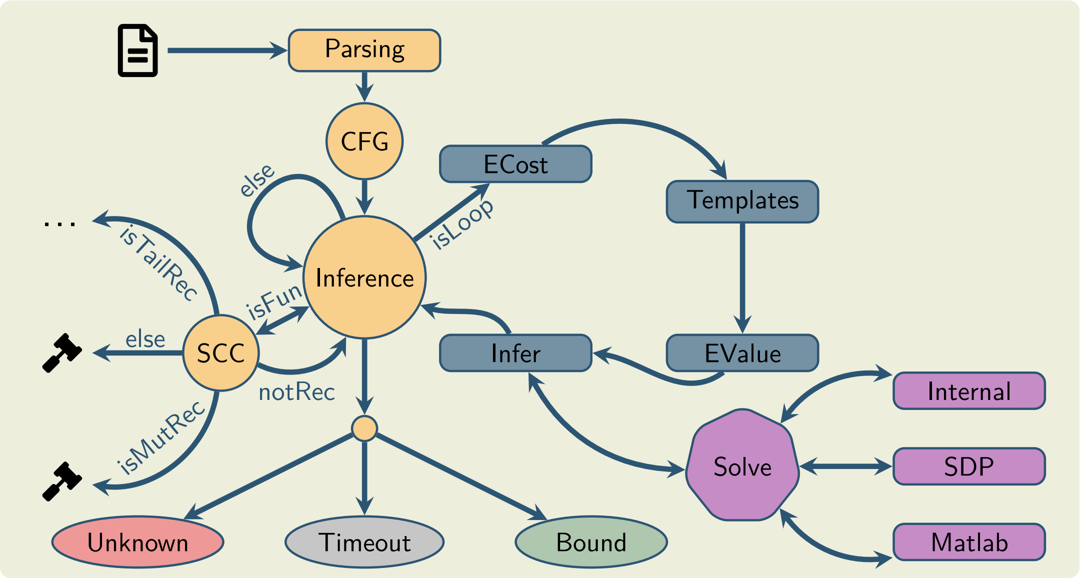

Parsing
CFG
else
Inference
isLoop
ECost
Templates
EValue
Infer
…
isTailRec
isFun
SCC
Internal
Solve
SDP
Matlab
Unknown
Timeout
Bound

ecoimp



Parsing
CFG
else
Inference
isLoop
ECost
Templates
EValue
Infer
...
isTailRec
isFun
SCC
notRec
Internal
Solve
SDP
Matlab
Unknown
Timeout
Bound

ecoimp


Parsing
CFG
else
ECost
Templates
. . .
isTailRec
isLoop
Inference
isFun
else
SCC
Infer
EValue
notRec
Internal
isMutRec
Solve
SDP
Matlab
Unknown
Timeout
Bound

Quickselect

Program Abstraction

- no arrays or pointer structures

Program Abstraction

- no arrays or pointer structures
- model complexity of original algorithm
- correct resource consumption

Quickselect

Program Abstraction

- no arrays or pointer structures
- model complexity of original algorithm
- correct resource consumption
- show correspondence between original and abstracted program with coupling

Program Abstraction

- no arrays or pointer structures
- model complexity of original algorithm
- correct resource consumption
- show correspondence between original and abstracted program with coupling

Quickselect

```
def qselect : {
  lo := 0;
  hi := N − 1;
  while(lo < hi) {
    consume(hi − lo);
    p := unif(lo, hi) ;
    if(p = pos) {
      lo := hi
    } {
      if(p < pos) {
        lo := p + 1
      } {
        hi := p − 1}}}}
```

Program Abstraction

- no arrays or pointer structures
- model complexity of original algorithm
- correct resource consumption
- show correspondence between original and abstracted program with coupling

Quickselect

```
def qselect : {
  lo := 0;
  hi := N − 1;
  while(lo < hi) {
    consume(hi − lo);
    p := unif(lo, hi);
    if(p = pos) {
      lo := hi
    } {
      if(p < pos) {
        lo := p + 1
      } {
        hi := p − 1}}}}
```

Program Abstraction

- no arrays or pointer structures
- model complexity of original algorithm
- correct resource consumption
- show correspondence between original and abstracted program with coupling

Analysis

We would like to automatically derive an upper bound for the quicksort algorithm, but our initial approach can't even handle quickselect.
Our Implementation fails to solve the resulting constraints of the quickselect algorithm as equating coefficients is to weak.

Quickselect

```
def qselect : {
  lo := 0;
  hi := N − 1;
  while(lo < hi) {
    consume(hi − lo);
    p := unif(lo, hi) ;
    if(p = pos) {
      lo := hi
    } {
      if(p < pos) {
        lo := p + 1
      } {
        hi := p − 1}}}}
```

Constraint Solving

- reduce the problem of finding a bound to checking for polynomials l,r that $l \geq r$

Constraint Solving

- reduce the problem of finding a bound to checking for polynomials l,r that $l \geq r$
- this check is done by equating coefficients
- for quickselect this is to weak

Constraint Solving

- reduce the problem of finding a bound to checking for polynomials l,r that $l \geq r$
- this check is done by equating coefficients
- for quickselect this is to weak
- instead check $l - r \geq 0$ (in general NP-hard) $\Rightarrow$ show that this polynomial is a sum of squares

Constraint Solving

- reduce the problem of finding a bound to checking for polynomials l,r that $l \geq r$
- this check is done by equating coefficients
- for quickselect this is to weak
- instead check $l - r \geq 0$ (in general NP-hard) $\Rightarrow$ show that this polynomial is a sum of squares

Sum-of-Squares (SOS)

We can show positivity of a polynomial by showing that it is a sum of squares. Let p be a polynomial, then p has an SOS decomposition if

$$p = \sum_i f_i^2$$

for polynomials f_i.

Constraint Solving

- r… polynomials l,r that $l \geq r$
- t…
- f…
- instead check … (in general NP-hard) … that this polynomial is a sum of squares

> NB: For a polynomial p the following holds
>
> p has an SOS decomposition $\implies p$ is positive

Sum-of-Squares (SOS)

We can show positivity of a polynomial by showing that it is a sum of squares. Let p be a polynomial, then p has an SOS decomposition if

$$p = \sum_i f_i^2$$

for polynomials f_i.

Automation using Semi-Definite Programming (SDP)

Such an SOS decomposition can be found using semi-definite programming. A polynomial p has an SOS decomposition if

$$p = v^T Q v$$

Automation using Semi-Definite Programming (SDP)

Such an SOS decomposition can be found using semi-definite programming. A polynomial p has an SOS decomposition if

$$p = v^T Q v$$

where matrix Q is positive semi-definite and v is a vector of possible monomials. The vector v is chosen from the variables in p according to specific heuristics.
A matrix Q which is positive semi-definite can be found by SDP.

Automation using Semi-Definite Programming (SDP)

Such an SOS decomposition can be found using semi-definite programming. A polynomial p has an SOS decomposition if

$$p = v^T Q v$$

where matrix Q is positive semi-definite and v is a vector of possible monomials. The vector v is chosen from the variables in p according to specific heuristics.
A matrix Q which is positive semi-definite can be found by SDP.

Context

We still do one of the most basic forms, we show that $l - r$ is positive.

Automation using Semi-Definite Programming (SDP)

Such an SOS decomposition can be found using semi-definite programming. A polynomial p has an SOS decomposition if

$$p = v^T Q v$$

where matrix Q is positive semi-definite and v is a vector of possible monomials. The vector v is chosen from the variables in p according to specific heuristics.
A matrix Q which is positive semi-definite can be found by SDP.

Context

We still do one of the most basic forms, we show that $l - r$ is positive.

- analysis on a program with information about variables

Automation using Semi-Definite Programming (SDP)

Such an SOS decomposition can be found using semi-definite programming. A polynomial p has an SOS decomposition if

$$p = v^T Q v$$

where matrix Q is positive semi-definite and v is a vector of possible monomials. The vector v is chosen from the variables in p according to specific heuristics.
A matrix Q which is positive semi-definite can be found by SDP.

Context

We still do one of the most basic forms, we show that $l - r$ is positive.

- analysis on a program with information about variables
- incorporate available information into constraint solving

Automation using Semi-Definite Programming (SDP)

Such an SOS decomposition can be found using semi-definite programming. A polynomial p has an SOS decomposition if

$$p = v^T Q v$$

where matrix Q is positive semi-definite and v is a vector of possible monomials. The vector v is chosen from the variables in p according to specific heuristics.
A matrix Q which is positive semi-definite can be found by SDP.

Context

We still do one of the most basic forms, we show that $l - r$ is positive.

- analysis on a program with information about variables
- incorporate available information into constraint solving
- we maintain a context of positive polynomials in our implementation

Experiments

Problem	ecoimp		ecoimp(v1.0)		Absynth		KoAT2		Amber	
miner	.1753	✱	.0482	✔	.1274	✔	2.7567	✔		⊘
qselect	13.6977	✔	.0200	⧗		⊘		⊘		⊘
qselect_rec	14.8312	✔		⊘		⊘		⊘		⊘
coupons-10	.0754	✔	.0662	✔	32.7563	✱	.3496	⧗	.0465	✔
coupons-N	13.4197	✱	.2900	✔		⊘	.3769	⧗		⊘
pol05	25.2870	✔	.0575	✔	.3191	✔	.9076	⧗		⊘
geo	.0310	✔	.0118	✔	.0309	⧗	.5874	✔	.0461	✔
nest-4	60.1884	⧗	1.2368	✔	60.0697	⧗	1.9257	⧗		⊘
rdbub	25.4311	✔	.0569	✔	.3551	✔	.8865	⧗		⊘
complex_past	56.3168	✱	.1106	⧗	.8738	⧗	1.3813	⧗	5.1363	✔
polynomial_past_1	60.3788	⧗	.1715	⧗	.4316	⧗	1.1733	⧗	1.2191	✔

Table: Here ✔, ✱, ⧗ or ⊘ denote that a bound was found, an imprecise bound was found, no bound was found or the problem is not applicable respectively.

Summary

- static program analysis for probabilistic programs

Summary

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

Summary

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

Summary

- static program analysis for probabilistic programs
- ECT calculus for compositional/modular analysis
- automation in ecoimp
- partly extension to recursion
- SDP solving using Matlab

Summary

- static program analysis for probabilistic programs
- ECT calculus for compositional/modular analysis
- automation in ecoimp
- partly extension to recursion
- SDP solving using Matlab

Current/Future Research

- finishing recursion
- standalone SDP solving using Csdp

Summary

- static program analysis for probabilistic programs
- ECT calculus for compositional/modular analysis
- automation in ecoimp
- partly extension to recursion
- SDP solving using Matlab

Current/Future Research

- finishing recursion
- standalone SDP solving using Csdp
- logarithmic bounds
- abstractions via coupling

Thank you for your attention!