Analyzing Expected Outcomes and Almost–Sure Termination of Probabilistic Programs is Hard

Benjamin Kaminski

Rheinisch–Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik 2

28.5.2014
Motivation

- Probabilistic Programs are like ordinary programs, except:
Motivation

- Probabilistic Programs are like ordinary programs, except:
  - Allow for random choice on how to continue the execution
  - Random choice is done with some specified probability $p$
Motivation

- Probabilistic Programs are like ordinary programs, except:
  - Allow for random choice on how to continue the execution
  - Random choice is done with some specified probability $p$

- **Analysis problems** we consider:
Motivation

- Probabilistic Programs are like ordinary programs, except:
  - Allow for random choice on how to continue the execution
  - Random choice is done with some specified probability \( p \)

- **Analysis problems** we consider:
  - Determine the value of a variable after program execution
Motivation

- Probabilistic Programs are like ordinary programs, except:
  - Allow for random choice on how to continue the execution
  - Random choice is done with some specified probability $p$
- Analysis problems we consider:
  - Determine the value of a variable after program execution
  - Decide whether the program terminates
Motivation

- Probabilistic Programs are like ordinary programs, except:
  - Allow for random choice on how to continue the execution
  - Random choice is done with some specified probability $p$

- Analysis problems we consider:
  - Determine the value of a variable after program execution
    - Determine expected values
  - Decide whether the program terminates
Motivation

- Probabilistic Programs are like ordinary programs, except:
  - Allow for random choice on how to continue the execution
  - Random choice is done with some specified probability $p$

- **Analysis problems** we consider:
  - Determine the value of a variable after program execution
    - Determine expected values (expected outcomes)
  - Decide whether the program terminates
Motivation

- Probabilistic Programs are like ordinary programs, except:
  - Allow for random choice on how to continue the execution
  - Random choice is done with some specified probability $p$

- Analysis problems we consider:
  - Determine the value of a variable after program execution
    - Determine expected values (expected outcomes)
  - Decide whether the program terminates
    - Decide almost–sure termination!
Motivation

- Probabilistic Programs are like ordinary programs, except:
  - Allow for random choice on how to continue the execution
  - Random choice is done with some specified probability $p$

- **Analysis problems** we consider:
  - Determine the value of a variable after program execution
    - Determine *expected* values (*expected outcomes*)
  - Decide whether the program terminates
    - Decide almost–sure termination!

How hard is it to solve these analysis problems?
The arithmetical hierarchy
Outline

1  The arithmetical hierarchy
2  Probabilistic programs
Outline

1. The arithmetical hierarchy
2. Probabilistic programs
   - Syntax
   - Semantics
   - Notations
1. The arithmetical hierarchy
2. Probabilistic programs
   - Syntax
   - Semantics
   - Notations
3. Hardness of determining expected outcomes
Outline

1. The arithmetical hierarchy
2. Probabilistic programs
   - Syntax
   - Semantics
   - Notations
3. Hardness of determining expected outcomes
4. Hardness of deciding almost–sure termination
The Arithmetical Hierarchy

Definition

Class $\mathcal{C}_0^n$ is defined as

$$\mathcal{C}_0^n = A = \{x : \exists y_1 \exists y_2 \exists y_3 \cdots \exists y_n : (\neg x, y_1, y_2, y_3, \ldots, y_n) \in R\},$$

where $R$ is a decidable relation.

Class $\mathcal{C}_0^n$ is defined as

$$\mathcal{C}_0^n = \mathcal{C}_0^n \setminus \mathcal{C}_0^n,$$

where $R$ is a decidable relation.
The Arithmetical Hierarchy

Class $\Sigma^0_n$ is defined as

$$\Sigma^0_n = \left\{ A \mid A = \{ \vec{x} \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists \forall y_n : (\vec{x}, y_1, y_2, y_3, \ldots, y_n) \in R \} \right\},$$

$R$ is a decidable relation.
The Arithmetical Hierarchy

Class $\Sigma^0_n$ is defined as

$$\Sigma^0_n = \left\{ A \mid A = \{ \vec{x} \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists / \forall y_n : (\vec{x}, y_1, y_2, y_3, \ldots, y_n) \in R \} \right\},$$

$R$ is a decidable relation

Class $\Pi^0_n$ is defined as

$$\Pi^0_n = \left\{ A \mid A = \{ \vec{x} \mid \forall y_1 \exists y_2 \forall y_3 \cdots \exists / \forall y_n : (\vec{x}, y_1, y_2, y_3, \ldots, y_n) \in R \} \right\},$$

$R$ is a decidable relation
The Arithmetical Hierarchy

- **Class $\Sigma^0_n$** is defined as
  
  \[
  \Sigma^0_n = \left\{ A \mid A = \{ \vec{x} \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists/\forall y_n : (\vec{x}, y_1, y_2, y_3, \ldots, y_n) \in R \} \right\},
  \]

  $R$ is a decidable relation

- **Class $\Pi^0_n$** is defined as
  
  \[
  \Pi^0_n = \left\{ A \mid A = \{ \vec{x} \mid \forall y_1 \exists y_2 \forall y_3 \cdots \exists/\forall y_n : (\vec{x}, y_1, y_2, y_3, \ldots, y_n) \in R \} \right\},
  \]

  $R$ is a decidable relation

- **Class $\Delta^0_n$** is defined as $\Delta^0_n = \Sigma^0_n \cap \Pi^0_n$
The Arithmetical Hierarchy Revisited
The Arithmetical Hierarchy Revisited

Reconsider class $\Sigma^0_n$:

$$\Sigma^0_n = \left\{ A \mid A = \{ \bar{x} \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists / \forall y_n : \right. \\
\left. (\bar{x}, y_1, y_2, y_3, \cdots, y_n) \in R \right\},$$

$R$ is a decidable relation
The Arithmetical Hierarchy Revisited

- Reconsider class $\Sigma_n^0$:

$$\Sigma_n^0 = \left\{ A \mid A = \{ \vec{x} \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists / \forall y_n : (\vec{x}, y_1, y_2, y_3, \ldots, y_n) \in R \} \right\},$$

$R$ is a decidable relation

- For our convenience: domain of the variables shall be $\mathbb{Q}^+$
The Arithmetical Hierarchy Revisited

- Reconsider class $\Sigma^0_n$:

$$\Sigma^0_n = \{ A \mid A = \{ \vec{x} \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists/\forall y_n : (\vec{x}, y_1, y_2, y_3, \ldots, y_n) \in R \},$$

$R$ is a decidable relation

- For our convenience: domain of the variables shall be $\mathbb{Q}^+$
The Arithmetical Hierarchy Revisited

- Reconsider class $\Sigma_n^0$:

  $$\Sigma_n^0 = \left\{ A \mid A = \{ \bar{x} \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists / \forall y_n : (\bar{x}, y_1, y_2, y_3, \ldots, y_n) \in R \} \right\},$$

  $R$ is a decidable relation

- For our convenience: domain of the variables shall be $\mathbb{Q}^+$

- Consecutive quantifiers of the same type can be contracted:
The Arithmetical Hierarchy Revisited

- Reconsider class $\Sigma^0_n$:

$$\Sigma^0_n = \left\{ A \mid A = \left\{ \vec{x} \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists / \forall y_n :\right. \right.$$  

$$\left. (\vec{x}, y_1, y_2, y_3, \ldots, y_n) \in R \right\},$$

$R$ is a decidable relation

- For our convenience: domain of the variables shall be $\mathbb{Q}^+$

- Consecutive quantifiers of the same type can be contracted:

$$A = \left\{ \vec{x} \mid \exists y_1 \exists y_2 \exists y_3 \forall y_4 \forall y_5 : (x, y_1, \ldots, y_5) \in R \right\}$$
The Arithmetical Hierarchy Revisited

- Reconsider class $\Sigma_0^0$:

$$\Sigma_0^0 = \left\{ A \mid A = \{ \vec{x} \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists / \forall y_n : (\vec{x}, y_1, y_2, y_3, \ldots, y_n) \in R \}, \right. \\
R \text{ is a decidable relation} \right\}$$

- For our convenience: domain of the variables shall be $\mathbb{Q}^+$

- Consecutive quantifiers of the same type can be contracted:

$$A = \{ \vec{x} \mid \exists y_1 \exists y_2 \exists y_3 \forall y_4 \forall y_5 : (x, y_1, \ldots, y_5) \in R \}$$
The Arithmetical Hierarchy Revisited

- Reconsider class $\Sigma^0_n$:

$$\Sigma^0_n = \left\{ A \mid A = \{ \bar{x} \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists /\forall y_n : \right.$$  

$$(\bar{x}, y_1, y_2, y_3, \ldots, y_n) \in R \right\},$$

$R$ is a decidable relation

- For our convenience: domain of the variables shall be $\mathbb{Q}^+$

- Consecutive quantifiers of the same type can be contracted:

$$A = \{ \bar{x} \mid \exists y_1 \exists y_2 \exists y_3 \forall y_4 \forall y_5 : (x, y_1, \ldots, y_5) \in R \}$$

$$= \{ \bar{x} \mid \exists y_{1,2,3} \forall y_{4,5} : (x, y_{1,2,3}, y_{4,5}) \in R' \}$$
The Arithmetical Hierarchy Revisited

- Reconsider class $\Sigma^0_n$:

$$\Sigma^0_n = \left\{ A \mid A = \{ \vec{x} \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists / \forall y_n : (\vec{x}, y_1, y_2, y_3, \ldots, y_n) \in R \}, R \text{ is a decidable relation} \right\}$$

- For our convenience: domain of the variables shall be $\mathbb{Q}^+$

- Consecutive quantifiers of the same type can be contracted:

$$A = \{ \vec{x} \mid \exists y_1 \exists y_2 \exists y_3 \forall y_4 \forall y_5 : (x, y_1, \ldots, y_5) \in R \}$$

$$= \{ \vec{x} \mid \exists y_{1,2,3} \forall y_{4,5} : (x, y_{1,2,3}, y_{4,5}) \in R' \}$$

$$\in \Sigma^0_2$$
The Arithmetical Hierarchy Revisited

- Reconsider class $\Sigma^0_n$:

$$\Sigma^0_n = \left\{ A \mid A = \{ \vec{x} \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists/\forall y_n : (\vec{x}, y_1, y_2, y_3, \ldots, y_n) \in R \} \right\},$$

where $R$ is a decidable relation.

- For our convenience: domain of the variables shall be $\mathbb{Q}^+$

- Consecutive quantifiers of the same type can be contracted:

$$A = \{ \vec{x} \mid \exists y_1 \exists y_2 \exists y_3 \forall y_4 \forall y_5 : (x, y_1, \ldots, y_5) \in R \}$$

$$= \{ \vec{x} \mid \exists y_{1,2,3} \forall y_{4,5} : (x, y_{1,2,3}, y_{4,5}) \in R' \}$$

$$\in \Sigma^0_2$$

- $n$ is really about the number of alternating quantifiers
The Arithmetical Hierarchy — The Bigger Picture
The Arithmetical Hierarchy — The Bigger Picture

- The following inclusion diagram holds:
The Arithmetical Hierarchy — The Bigger Picture

- The following inclusion diagram holds:

  ![Arithmetical Hierarchy Diagram]

- All inclusions are strict
The Arithmetical Hierarchy — The Bigger Picture

- The following inclusion diagram holds:

- All inclusions are strict
Probabilistic Programs — Syntax

- Syntax of probabilistic programs:
Probabilistic Programs — Syntax

Syntax of probabilistic programs:

- **Assignment**: \( var := expr \)
Probabilistic Programs — Syntax

- Syntax of probabilistic programs:
  - Assignment: `var := expr`
  - Concatenation: `P_1; P_2`
Probabilistic Programs — Syntax

Syntax of probabilistic programs:

- **Assignment**: $\text{var} := \text{expr}$
- **Concatenation**: $P_1; P_2$
- **While–loop**: $\text{WHILE } (bexpr) \{ P \}$
## Probabilistic Programs — Syntax

- **Syntax of probabilistic programs:**
  - **Assignment:** $\textit{var} := \textit{expr}$
  - **Concatenation:** $P_1; P_2$
  - **While–loop:** $\textsc{While } (\textit{bexpr}) \{ P \}$
  - **Probabilistic choice:** $\{P_1\} [p] \{P_2\}$, for $p \in [0, 1] \subseteq \mathbb{Q}$
Syntax of probabilistic programs:

- **Assignment**: `var := expr`
- **Concatenation**: `P_1; P_2`
- **While–loop**: `WHILE (bexpr) {P}`
- **Probabilistic choice**: `{P_1} [p] {P_2}`, for `p ∈ [0, 1] ⊆ Q`

Denote the set of probabilistic programs by `Prog`. 
Probabilistic Programs — Syntax

- Syntax of probabilistic programs:
  - Assignment: \( \text{var} := \text{expr} \)
  - Concatenation: \( P_1 ; P_2 \)
  - While-loop: \( \text{WHILE } (\text{bexpr}) \{P\} \)
  - Probabilistic choice: \( \{P_1\} [p] \{P_2\}, \text{ for } p \in [0, 1] \subseteq \mathbb{Q} \)

- Denote the set of probabilistic programs by \( \text{Prog} \).

- Denote the set of ordinary programs (programs that contain no probabilistic choice) by \( \text{ordProg} \).
Probabilistic Programs — Semantics

- Set of variable valuations: \( \mathcal{V} = \{ \eta \mid \eta : \text{Var} \rightarrow \mathbb{Q}^+ \} \)
Probabilistic Programs — Semantics

- Set of variable valuations: $\mathbb{V} = \{\eta \mid \eta: \text{Var} \rightarrow \mathbb{Q}^+\}$

- Set of program states: $\mathbb{S} = (\text{Prog} \cup \{\downarrow\}) \times \mathbb{V} \times I \times \{L, R\}^*$, for $I = [0, 1] \subseteq \mathbb{Q}^+$
Probabilistic Programs — Semantics

- Set of variable valuations: \( \mathcal{V} = \{ \eta \mid \eta: \text{Var} \to \mathbb{Q}^+ \} \)
- Set of program states: \( \mathcal{S} = (\text{Prog} \cup \{\downarrow\}) \times \mathcal{V} \times I \times \{L, R\}^* \), for \( I = [0, 1] \subseteq \mathbb{Q}^+ \)
- A closer look at a program state \( \sigma = \langle P, \eta, a, \theta \rangle \in \mathcal{S} \):
Probabilistic Programs — Semantics

- Set of variable valuations: $\mathbb{V} = \{ \eta \mid \eta : \text{Var} \to \mathbb{Q}^+ \}$

- Set of program states: $\mathbb{S} = (\text{Prog} \cup \{\downarrow\}) \times \mathbb{V} \times I \times \{L, R\}^*$, for $I = [0, 1] \subseteq \mathbb{Q}^+$

- A closer look at a program state $\sigma = \langle P, \eta, a, \theta \rangle \in \mathbb{S}$:
  - $P$ is the program remaining to be executed
Probabilistic Programs — Semantics

- Set of variable valuations: $\mathcal{V} = \{\eta \mid \eta: \text{Var} \rightarrow \mathbb{Q}^+\}$
- Set of program states: $\mathcal{S} = (\text{Prog} \cup \{\downarrow\}) \times \mathcal{V} \times I \times \{L, R\}^*$, for $I = [0, 1] \subseteq \mathbb{Q}^+$

A closer look at a program state $\sigma = \langle P, \eta, a, \theta \rangle \in \mathcal{S}$:
- $P$ is the program remaining to be executed
  - If $P = \downarrow$ then $\sigma$ is a terminal state
Probabilistic Programs — Semantics

- Set of variable valuations: \( \mathbb{V} = \{ \eta \mid \eta: \text{Var} \to \mathbb{Q}^+ \} \)

- Set of program states: \( \mathbb{S} = (\text{Prog} \cup \{ \downarrow \}) \times \mathbb{V} \times I \times \{ L, R \}^* \), for \( I = [0, 1] \subseteq \mathbb{Q}^+ \)

- A closer look at a program state \( \sigma = \langle P, \eta, a, \theta \rangle \in \mathbb{S} \):
  - \( P \) is the program remaining to be executed
    - If \( P = \downarrow \) then \( \sigma \) is a terminal state
  - \( \eta \) is the current variable valuation
Probabilistic Programs — Semantics

- Set of variable valuations: \( \mathcal{V} = \{ \eta \mid \eta: \text{Var} \to \mathbb{Q}^+ \} \)

- Set of program states: \( \mathcal{S} = (\text{Prog} \cup \{ \downarrow \}) \times \mathcal{V} \times I \times \{L, R\}^* \), for \( I = [0, 1] \subseteq \mathbb{Q}^+ \)

A closer look at a program state \( \sigma = \langle P, \eta, a, \theta \rangle \in \mathcal{S} \):

- \( P \) is the program remaining to be executed
  - If \( P = \downarrow \) then \( \sigma \) is a terminal state

- \( \eta \) is the current variable valuation

- \( \theta \) is a string over \( \{L, R\} \) that encodes which probabilistic choices were made to reach \( \sigma \)
Probabilistic Programs — Semantics

- Set of variable valuations: \( \mathcal{V} = \{ \eta \mid \eta: \text{Var} \rightarrow \mathbb{Q}^+ \} \)

- Set of program states: \( \mathcal{S} = (\text{Prog} \cup \{\downarrow\}) \times \mathcal{V} \times I \times \{L, R\}^* \), for \( I = [0, 1] \subseteq \mathbb{Q}^+ \)

- A closer look at a program state \( \sigma = \langle P, \eta, a, \theta \rangle \in \mathcal{S} \):
  - \( P \) is the program remaining to be executed
  - If \( P = \downarrow \) then \( \sigma \) is a terminal state
  - \( \eta \) is the current variable valuation
  - \( \theta \) is a string over \( \{L, R\} \) that encodes which probabilistic choices were made to reach \( \sigma \)
  - \( a \) is the probability that those choices were made
Semantics of probabilistic programs:

- Semantics of probabilistic programs:

  1. Semantics of probabilistic programs:
Probabilistic Programs — Semantics continued

- **Semantics of probabilistic programs**: smallest relation $\vdash \subseteq S \times S$ which satisfies the following inference rules:
Semantics of probabilistic programs: smallest relation \( \models \subseteq \mathcal{S} \times \mathcal{S} \) which satisfies the following inference rules:

(assign)

\[
\langle v := e, \eta, a, \theta \rangle \models \langle \downarrow, \eta[v \mapsto \max\{\llbracket e \rrbracket_{\eta}, 0\}], a, \theta \rangle
\]
Probabilistic Programs — Semantics continued

**Semantics of probabilistic programs**: smallest relation $\vdash \subseteq S \times S$ which satisfies the following inference rules:

\[
\begin{align*}
\text{(assign)} & \quad \frac{\langle v := e, \eta, a, \theta \rangle \vdash \langle \downarrow, \eta[v \mapsto \max\{\llbracket e \rrbracket_\eta, 0\}], a, \theta \rangle}{\vdash}
\end{align*}
\]
Probabilistic Programs — Semantics continued

- **Semantics of probabilistic programs**: smallest relation $\vdash \subseteq \mathcal{S} \times \mathcal{S}$ which satisfies the following inference rules:

  (assign) $\frac{\langle v := e, \eta, a, \theta \rangle \vdash \langle \downarrow, \eta[v \mapsto \max\{\|e\|_\eta, 0\}], a, \theta \rangle}{\vdash}$

  (concat1) $\frac{\langle P_1, \eta, a, \theta \rangle \vdash \langle P_1', \eta', a', \theta' \rangle}{\langle P_1; P_2, \eta, a, \theta \rangle \vdash \langle P_1'; P_2, \eta', a', \theta' \rangle}$

  (concat2) $\langle \downarrow; P_2, \eta, a, \theta \rangle \vdash \langle P_2, \eta, a, \theta \rangle$
Probabilistic Programs — Semantics continued

Semantics of probabilistic programs: smallest relation $\vdash \subseteq S \times S$ which satisfies the following inference rules:

(assign) $\vdash v := e, \eta, a, \theta \vdash \downarrow, \eta[v \mapsto \text{max}\{\llbracket e \rrbracket_\eta, 0\}], a, \theta$

(concat1) $\vdash P_1, \eta, a, \theta \vdash P'_1, \eta', a', \theta' \vdash P_1 ; P_2, \eta, a, \theta \vdash P'_1 ; P_2, \eta', a', \theta'$

(concat2) $\vdash \downarrow ; P_2, \eta, a, \theta \vdash P_2, \eta, a, \theta$

(while1) $\llbracket b \rrbracket_\eta = \text{True} \quad \vdash \text{WHILE} (b) \{P\}, \eta, a, \theta \vdash \langle P ; \text{WHILE} (b) \{P\}, \eta, a, \theta \rangle$

(while2) $\llbracket b \rrbracket_\eta = \text{False} \quad \vdash \text{WHILE} (b) \{P\}, \eta, a, \theta \vdash \downarrow, \eta, a, \theta$
(prob1)  \[
\langle \{P_1\} [p] \{P_2\}, \eta, a, \theta \rangle \vdash \langle P_1, \eta, a \cdot p, \theta \cdot L \rangle
\]
(prob1) \[
\langle \{P_1\}[p]\{P_2\}, \eta, a, \theta \rangle \vdash \langle P_1, \eta, a \cdot p, \theta \cdot L \rangle
\]
Probabilistic Programs — Semantics continued

\[(\text{prob1})\]
\[
\langle\{P_1\}[p]\{P_2\}, \eta, a, \theta \rangle \vdash \langle P_1, \eta, a \cdot p, \theta \cdot L \rangle
\]

\[(\text{prob2})\]
\[
\langle\{P_1\}[p]\{P_2\}, \eta, a, \theta \rangle \vdash \langle P_2, \eta, a \cdot (1 - p), \theta \cdot R \rangle
\]
Probabilistic Programs — Semantics continued

(prob1) \[ \langle \{P_1\} [p] \{P_2\}, \eta, a, \theta \rangle \vdash \langle P_1, \eta, a \cdot p, \theta \cdot L \rangle \]

(prob2) \[ \langle \{P_1\} [p] \{P_2\}, \eta, a, \theta \rangle \vdash \langle P_2, \eta, a \cdot (1 - p), \theta \cdot R \rangle \]
Probabilistic Programs — Semantics continued

(prob1) \[ \langle \{ P_1 \} \mid \{ P_2 \}, \eta, a, \theta \rangle \vdash \langle P_1, \eta, a \cdot p, \theta \cdot L \rangle \]

(prob2) \[ \langle \{ P_1 \} \mid \{ P_2 \}, \eta, a, \theta \rangle \vdash \langle P_2, \eta, a \cdot (1 - p), \theta \cdot R \rangle \]

- Use \( \sigma \models^k \tau \) in the usual sense
Probabilistic Programs — Semantics continued

\[
\frac{\langle \{P_1\} \ [p] \ {P_2}\rangle, \eta, a, \theta}{\langle P_1, \eta, a \cdot p, \theta \cdot L \rangle}
\]

\[
\frac{\langle \{P_1\} \ [p] \ {P_2}\rangle, \eta, a, \theta}{\langle P_2, \eta, a \cdot (1 - p), \theta \cdot R \rangle}
\]

- Use $\sigma \vdash^k \tau$ in the usual sense
- Write $\sigma \vdash_{(\text{name})} \tau$
Probabilistic Programs — Semantics continued

\begin{align*}
\text{(prob1)} & \quad \langle \{ P_1 \} [p] \{ P_2 \}, \eta, a, \theta \rangle \vdash \langle P_1, \eta, a \cdot p, \theta \cdot L \rangle \\
\text{(prob2)} & \quad \langle \{ P_1 \} [p] \{ P_2 \}, \eta, a, \theta \rangle \vdash \langle P_2, \eta, a \cdot (1 - p), \theta \cdot R \rangle
\end{align*}

- Use $\sigma \vdash^k \tau$ in the usual sense
- Write $\sigma \vdash^{\text{name}} \tau$ if $\tau$ is inferred by the use of the \text{name}–rule (for \text{name} $\in \{ \text{assign, concat1, \ldots } \}$)
State Successors — The Classical Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$
State Successors — The Classical Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$

- Assume next instruction in $P$ is a classical instruction
State Successors — The Classical Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$
- Assume next instruction in $P$ is a classical instruction

Ex. function $T$ that computes the (unique) successor of $\sigma$: 
State Successors — The Classical Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$
- Assume next instruction in $P$ is a classical instruction

Example function $T$ that computes the (unique) successor of $\sigma$:

$$T(\sigma) = \begin{cases} \tau, & \text{if } \sigma \vdash \tau \\ T, & \text{else} \end{cases}$$
State Successors — The Classical Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$
- Assume next instruction in $P$ is a classical instruction

Ex. function $T$ that computes the (unique) successor of $\sigma$:

$$T(\sigma) = \begin{cases} \tau, & \text{if } \sigma \vdash \tau \\ \top, & \text{else (i.e. if } \sigma = \langle \downarrow, \eta, a, \theta \rangle \rangle \end{cases}$$
State Successors — The Classical Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$
- Assume next instruction in $P$ is a classical instruction

Ex. function $T$ that computes the (unique) successor of $\sigma$:

$$T(\sigma) = \begin{cases} \tau, & \text{if } \sigma \vdash \tau \\ \top, & \text{else (i.e. if } \sigma = \langle \downarrow, \eta, a, \theta \rangle) \end{cases}$$

How do we deal with probabilistic choice?
State Successors — The Classical Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$
- Assume next instruction in $P$ is a classical instruction

Ex. function $T$ that computes the (unique) successor of $\sigma$:

$$T(\sigma) = \begin{cases} 
\tau, & \text{if } \sigma \vdash \tau \\
\top, & \text{else (i.e. if } \sigma = \langle \downarrow, \eta, a, \theta \rangle \rangle
\end{cases}$$

How do we deal with probabilistic choice? Basically, we simply tell $T$ how to resolve it!
State Successors — The Probabilistic Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$
State Successors — The Probabilistic Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$
- Assume next instruction in $P$ is a probabilistic choice
State Successors — The Probabilistic Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$
- Assume next instruction in $P$ is a probabilistic choice
- $\sigma$ has two successors according to $\vdash$
State Successors — The Probabilistic Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$
- Assume next instruction in $P$ is a **probabilistic choice**
- $\sigma$ has two successors according to $\sqsubseteq$
- Provide a symbol $s \in \{L, R\}$ resolving the probabilistic
State Successors — The Probabilistic Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$
- Assume next instruction in $P$ is a probabilistic choice
- $\sigma$ has two successors according to $\vdash$
- Provide a symbol $s \in \{L, R\}$ resolving the probabilistic

There exists a function $T_{prob}$ that computes the successor of $\sigma$: 
State Successors — The Probabilistic Case

- Let $\sigma = \langle P, \eta, a, \theta \rangle$
- Assume next instruction in $P$ is a probabilistic choice
- $\sigma$ has two successors according to $\vdash$
- Provide a symbol $s \in \{L, R\}$ resolving the probabilistic

There exists a function $T_{prob}$ that computes the successor of $\sigma$:

$$T_{prob}(\sigma, s) = \begin{cases} 
\tau_L, & \text{if } s = L \text{ and } \sigma \vdash_{(prob1)} \tau_L \\
\tau_R, & \text{if } s = R \text{ and } \sigma \vdash_{(prob2)} \tau_R 
\end{cases}$$
State Successors — The Probabilistic Case

Let $\sigma = \langle P, \eta, a, \theta \rangle$

Assume next instruction in $P$ is a probabilistic choice

$\sigma$ has two successors according to $\vdash$

Provide a symbol $s \in \{L, R\}$ resolving the probabilistic

There exists a function $T_{prob}$ that computes the successor of $\sigma$:

$$T_{prob}(\sigma, s) = \begin{cases} 
\tau_L, & \text{if } s = L \text{ and } \sigma \vdash_{(prob1)} \tau_L \\
\tau_R, & \text{if } s = R \text{ and } \sigma \vdash_{(prob2)} \tau_R 
\end{cases}$$

Provided $s$, the successor of $\sigma$ is unique!
$k$–th State Successors — The Combined Case
$k$–th State Successors — The Combined Case

There exists a computable function $T^*_{prob}$, such that:
$k$–th State Successors — The Combined Case

There exists a computable function $T^*_{\text{prob}}$, such that:

$$T^*_{\text{prob}}(\sigma, k, w) = \begin{cases} \tau, & \text{if } \sigma = \langle P, \eta, a, \theta \rangle \vdash^k \langle P', \eta', a', \theta \cdot w \rangle = \tau \\ T, & \text{else} \end{cases}$$
There exists a computable function $T_{prob}^*$, such that:

$$T_{prob}^*(\sigma, k, w) = \begin{cases} \tau, & \text{if } \sigma = \langle P, \eta, a, \theta \rangle \vdash^k \langle P', \eta', a', \theta \cdot w \rangle = \tau \\ \top, & \text{else} \end{cases}$$

- $T_{prob}^*$ returns a successor state $\tau$, if
There exists a computable function $T^*_{prob}$, such that:

$$T^*_{prob}(\sigma, k, w) = \begin{cases} \tau, & \text{if } \sigma = \langle P, \eta, a, \theta \rangle \vdash^k \langle P', \eta', a', \theta \cdot w \rangle = \tau \\ \top, & \text{else} \end{cases}$$

- $T^*_{prob}$ returns a successor state $\tau$, if $
- $\sigma \vdash^k \tau$
There exists a computable function $T^*_\text{prob}$, such that:

$$T^*_\text{prob}(\sigma, k, w) = \begin{cases} \tau, & \text{if } \sigma = \langle P, \eta, a, \theta \rangle \mid^k \langle P', \eta', a', \theta \cdot w \rangle = \tau \\ \top, & \text{else} \end{cases}$$

- $T^*_\text{prob}$ returns a successor state $\tau$, if
  - $\sigma \mid^k \tau$
  - Exactly $|w|$ probabilistic choices occur
There exists a computable function $T^{*}_{\text{prob}}$, such that:

$$T^{*}_{\text{prob}}(\sigma, k, w) = \begin{cases} \tau, & \text{if } \sigma = \langle P, \eta, a, \theta \rangle \vdash^{k} \langle P', \eta', a', \theta \cdot w \rangle = \tau \\ \bot, & \text{else} \end{cases}$$

- $T^{*}_{\text{prob}}$ returns a successor state $\tau$, if
  - $\sigma \vdash^{k} \tau$
  - **Exactly** $|w|$ probabilistic choices occur
  - The probabilistic choices are resolved according to $w$
\(k\)-th State Successors — The Combined Case

There exists a computable function \(T_{prob}^*\), such that:

\[
T_{prob}^*(\sigma, k, w) = \begin{cases} 
\tau, & \text{if } \sigma = \langle P, \eta, a, \theta \rangle \vdash^k \langle P', \eta', a', \theta \cdot w \rangle = \tau \\
\top, & \text{else}
\end{cases}
\]

- \(T_{prob}^*\) returns a successor state \(\tau\), if
  - \(\sigma \vdash^k \tau\)
  - Exactly \(|w|\) probabilistic choices occur
  - The probabilistic choices are resolved according to \(w\)
- Otherwise, \(T_{prob}^*\) returns \(\top\)
Some Helper Functions
Some Helper Functions

Extracting Probabilities and Variable Values of Terminal States

\[ \alpha(\sigma) = \begin{cases} 
  a, & \text{if } \sigma = \langle \downarrow, \eta, a, \theta \rangle \\
  0, & \text{otherwise}
\end{cases} \]
Some Helper Functions

Extracting Probabilities and Variable Values of Terminal States

\[ \alpha(\sigma) = \begin{cases} 
  a, & \text{if } \sigma = \langle \downarrow, \eta, a, \theta \rangle \\
  0, & \text{otherwise}
\end{cases} \]

\[ \varphi(\sigma, v) = \begin{cases} 
  \eta(v) \cdot a, & \text{if } \sigma = \langle \downarrow, \eta, a, \theta \rangle \\
  0, & \text{otherwise}
\end{cases} \]
Some Helper Functions

Extracting Probabilities and Variable Values of Terminal States

\[
\alpha(\sigma) = \begin{cases} 
a, & \text{if } \sigma = \langle \downarrow, \eta, a, \theta \rangle \\
0, & \text{otherwise}
\end{cases}
\]

\[
\phi(\sigma, v) = \begin{cases} 
\eta(v) \cdot a, & \text{if } \sigma = \langle \downarrow, \eta, a, \theta \rangle \\
0, & \text{otherwise}
\end{cases}
\]

Computable Enumeration of all \( w \in \{L, R\}^* \)

There exists a computable bijection \( h : \mathbb{N} \rightarrow \{L, R\}^* \).
Expected Outcomes

The Expected Outcome $E_P(v)$

The expected outcome of variable $v$ after executing program $P$:

$$E_P(v) := \sum_{i \in \mathbb{N}} \sum_{k \in \mathbb{N}} \varphi(T^*_{prob}((P, \eta_0, 1, \varepsilon), k, h(i)), v)$$
Expected Outcomes

The Expected Outcome $E_P(v)$

The expected outcome of variable $v$ after executing program $P$:

$$E_P(v) := \sum_{i \in \mathbb{N}} \sum_{k \in \mathbb{N}} \phi \left( T^*_\text{prob} \left( \langle P, \eta_0, 1, \varepsilon \rangle, k, h(i) \right), v \right)$$

- Start in initial state $\langle P, \eta_0, 1, \varepsilon \rangle$
Expected Outcomes

The Expected Outcome $E_P(v)$

The expected outcome of variable $v$ after executing program $P$:

$$E_P(v) := \sum_{i \in \mathbb{N}} \sum_{k \in \mathbb{N}} \varnothing(T^*_\text{prob}(\langle P, \eta_0, 1, \varepsilon \rangle, k, h(i)), v)$$

- Start in initial state $\langle P, \eta_0, 1, \varepsilon \rangle$
- Allow $P$ any finite number of $k \in \mathbb{N}$ steps to terminate
Expected Outcomes

The Expected Outcome $E_P(v)$

The expected outcome of variable $v$ after executing program $P$:

$$E_P(v) := \sum_{i \in \mathbb{N}} \sum_{k \in \mathbb{N}} \phi \left( T^*_{prob} \left( \langle P, \eta_0, 1, \varepsilon \rangle, k, h(i) \right), v \right)$$

- Start in initial state $\langle P, \eta_0, 1, \varepsilon \rangle$
- Allow $P$ any finite number of $k \in \mathbb{N}$ steps to terminate
- Sum over all possible resolutions of the probabilistic choice
The Expected Outcome $E_P(v)$

The expected outcome of variable $v$ after executing program $P$:

$$E_P(v) := \sum_{i \in \mathbb{N}} \sum_{k \in \mathbb{N}} \varphi(T_{prob}^*(\langle P, \eta_0, 1, \varepsilon \rangle, k, h(i)), v)$$

- Start in initial state $\langle P, \eta_0, 1, \varepsilon \rangle$
- Allow $P$ any finite number of $k \in \mathbb{N}$ steps to terminate
- Sum over all possible resolutions of the probabilistic choice
- For each terminal state, take the probability times the value of $v$ as the summand
The Probability $\Pr_P(\downarrow)$ that $P$ terminates

The probability that $P$ terminates:

\[
\Pr_P(\downarrow) := \sum_{i \in \mathbb{N}} \sum_{k \in \mathbb{N}} \alpha \left( T^*_{\text{prob}} \left( \langle P, \eta_0, 1, \varepsilon \rangle, k, h(i) \right) \right)
\]
Termination Probabilities

The Probability $\Pr_P(\downarrow)$ that $P$ terminates

The probability that $P$ terminates:

$$\Pr_P(\downarrow) := \sum_{i \in \mathbb{N}} \sum_{k \in \mathbb{N}} \alpha \left( T^*_{\text{prob}}(\langle P, \eta_0, 1, \varepsilon \rangle, k, h(i)) \right)$$

- Start in initial state $\langle P, \eta_0, 1, \varepsilon \rangle$
Termination Probabilities

The Probability $\Pr_P(\downarrow)$ that $P$ terminates

The probability that $P$ terminates:

$$\Pr_P(\downarrow) := \sum_{i \in \mathbb{N}} \sum_{k \in \mathbb{N}} \alpha \left( T_{\text{prob}}^* (\langle P, \eta_0, 1, \varepsilon \rangle, k, h(i)) \right)$$

- Start in initial state $\langle P, \eta_0, 1, \varepsilon \rangle$
- Allow $P$ any finite number of $k \in \mathbb{N}$ steps to terminate
Termination Probabilities

The Probability $\Pr_P(\downarrow)$ that $P$ terminates:

$$\Pr_P(\downarrow) := \sum_{i \in \mathbb{N}} \sum_{k \in \mathbb{N}} \alpha \left( T_{prob}^* \left( \langle P, \eta_0, 1, \varepsilon \rangle, k, h(i) \right) \right)$$

- Start in initial state $\langle P, \eta_0, 1, \varepsilon \rangle$
- Allow $P$ any finite number of $k \in \mathbb{N}$ steps to terminate
- Sum over all possible resolutions of the probabilistic choice
The Probability $\Pr_P(\downarrow)$ that $P$ terminates

The probability that $P$ terminates:

$$\Pr_P(\downarrow) := \sum_{i \in \mathbb{N}} \sum_{k \in \mathbb{N}} \alpha \left( T_{\text{prob}}^* \left( \langle P, \eta_0, 1, \varepsilon \rangle, k, h(i) \right) \right)$$

- Start in initial state $\langle P, \eta_0, 1, \varepsilon \rangle$
- Allow $P$ any finite number of $k \in \mathbb{N}$ steps to terminate
- Sum over all possible resolutions of the probabilistic choice
- For each terminal state, take the probability as the summand
Decision Problems We Analyzed
## Decision Problems We Analyzed

**Almost–Sure Termination $\text{AST}$**

\[
P \in \text{AST} \iff \Pr_P(\downarrow) = 1
\]
Decision Problems We Analyzed

Almost–Sure Termination $\textit{AST}$

\[
P \in \textit{AST} : \iff \Pr_P(\downarrow) = 1
\]

Lower and Upper Bounds, and Exact Expected Outcomes

\[
(P, v, q) \in \textit{LEXP} : \iff q < E_P(v)
\]
\[
(P, v, q) \in \textit{UEXP} : \iff q > E_P(v)
\]
\[
(P, v, q) \in \textit{EXP} : \iff q = E_P(v)
\]
Hardness of Deciding $\mathbf{LEXP}$
Recursive Enumerability of $\text{LEXP}$
Recursive Enumerability of $\mathcal{LEXP}$

Lemma

$\mathcal{LEXP} \in \Sigma^0_1$, thus $\mathcal{LEXP}$ is recursively enumerable
Recursive Enumerability of $\mathcal{LEXP}$

**Lemma**

$\mathcal{LEXP} \in \Sigma_1^0$, thus $\mathcal{LEXP}$ is recursively enumerable

**Proof of $\mathcal{LEXP} \in \Sigma_1^0$**

...
Recursive Enumerability of $\mathcal{LEXP}$

**Lemma**

$\mathcal{LEXP} \in \Sigma^0_1$, thus $\mathcal{LEXP}$ is recursively enumerable

**Proof of $\mathcal{LEXP} \in \Sigma^0_1$**

$\mathcal{LEXP}$ is defined by the following formula:

$$\exists y_1 \exists y_2 : q < \sum_{i=1}^{y_1} \sum_{k=1}^{y_2} \psi \left( T_{\text{prob}}^* \left( \langle P, \eta_0, 1, \epsilon \rangle, k, h(i) \right), v \right)$$
Proof of $\mathcal{LEXP} \in \Sigma^0_1$

$\mathcal{LEXP}$ is defined by the following formula:

$$\exists y_1 \exists y_2 : q < \sum_{i=1}^{y_1} \sum_{k=1}^{y_2} \varphi\left(T^*_{prob}(\langle P, \eta_0, 1, \varepsilon \rangle, k, h(i)), v \right)$$
Proof of $\mathcal{LEXP} \in \sum_1^0$

$\mathcal{LEXP}$ is defined by the following formula:

$$\exists y_1 \exists y_2 : q < \sum_{i=1}^{y_1} \sum_{k=1}^{y_2} \phi \left( T_{prob}^* \left( \langle P, \eta_0, 1, \varepsilon \rangle, k, h(i) \right), v \right)$$

\[\mathbb{E}_P(v) \]
Upper Bound for Hardness of Deciding $\text{UEXP}$
Upper Bound for Hardness of Deciding $\mathcal{UEX}P$

Lemma

$\mathcal{UEX}P^2 \supseteq \mathcal{L}_{0,1}$

Proof of $\mathcal{UEX}P^2 \supseteq \mathcal{L}_{0,1}$

$\mathcal{UEX}P$ is defined by the following formula:

$q > 0$

$y_1, y_2 : q_i \geq 1$

$x_i = 1$

$x_k \leq 1$

$\{ \downarrow \}

\left\{ \begin{array}{c}
\mathbb{P}_{v} (d_i, v, t_i) \end{array} \right.$

Benjamin Kaminski

Analyzing Probabilistic Programs is Hard 28.5.2014
Upper Bound for Hardness of Deciding $\text{UEXP}$

Lemma

$\text{UEXP} \in \Sigma^0_2$
Upper Bound for Hardness of Deciding $\text{UEXP}$

Lemma

$\text{UEXP} \in \Sigma^0_2$

Proof of $\text{UEXP} \in \Sigma^0_2$
Upper Bound for Hardness of Deciding $\mathcal{UEXP}$

Lemma

$\mathcal{UEXP} \in \Sigma^0_2$

Proof of $\mathcal{UEXP} \in \Sigma^0_2$

$\mathcal{UEXP}$ is defined by the following formula:

$$\exists \delta > 0 \ \forall y_1 \ \forall y_2 : \ q - \delta > \sum_{i=1}^{y_1} \sum_{k=1}^{y_2} \phi \left( T^*_{\text{prob}} \left( \langle P, \eta_0, 1, \varepsilon \rangle, k, h(i) \right), v \right)$$
Proof of $\mathcal{UEXP} \in \Sigma^0_2$

$\mathcal{UEXP}$ is defined by the following formula:

$$\exists \delta > 0 \ \forall y_1 \ \forall y_2 : \ q - \delta > \sum_{i=1}^{y_1} \sum_{k=1}^{y_2} \phi\left( T^*_{prob}(\langle P, \eta_0, 1, \varepsilon \rangle, k, h(i)), v \right)$$
Proof of $\mathcal{UEXP} \in \Sigma^0_2$

$\mathcal{UEXP}$ is defined by the following formula:

$$\exists \delta > 0 \ \forall y_1 \ \forall y_2 : q - \delta > \sum_{i=1}^{y_1} \sum_{k=1}^{y_2} \delta \left( T_{\text{prob}}^* \left( \langle P, \eta_0, 1, \varepsilon \rangle, k, h(i) \right), v \right)$$
Actual Hardness of Deciding $\text{UEXP}$ and $\text{EXP}$
The Universal Halting Problem $UH$
The Universal Halting Problem $\mathcal{UH}$

The universal halting problem $\mathcal{UH} \subset \text{ordProg}$:

$$P \in \mathcal{UH} \iff \forall \eta \exists k \exists \eta' : \langle P, \eta, 1, \varepsilon \rangle \vdash^k \langle \bot, \eta', 1, \varepsilon \rangle$$
The Universal Halting Problem $\mathcal{UH}$

The universal halting problem $\mathcal{UH} \subset \text{ordProg}$:

$$P \in \mathcal{UH} \iff \forall \eta \exists k \exists \eta' : \langle P, \eta, 1, \varepsilon \rangle \vdash^k \langle \downarrow, \eta', 1, \varepsilon \rangle$$

The complement of the universal halting problem $\overline{\mathcal{UH}}$:

$$\overline{\mathcal{UH}} := \text{ordProg} \setminus \mathcal{UH}.$$
The Universal Halting Problem $\mathcal{UH}$

The universal halting problem $\mathcal{UH} \subset \text{ordProg}$:

$$P \in \mathcal{UH} \iff \forall \eta \exists k \exists \eta': \langle P, \eta, 1, \varepsilon \rangle \Downarrow^k \langle \downarrow, \eta', 1, \varepsilon \rangle$$

The complement of the universal halting problem $\overline{\mathcal{UH}}$:

$$\overline{\mathcal{UH}} := \text{ordProg} \setminus \mathcal{UH}.$$  

Completenesses of $\mathcal{UH}$ and $\overline{\mathcal{UH}}$

$\mathcal{UH}$ is $\Pi^0_2$–complete and $\overline{\mathcal{UH}}$ is $\Sigma^0_2$–complete.
Hardness of Deciding $\text{UEXP}$ and $\text{EXP}$

Lemma $\text{UEXP}$ is $\mathbb{O}_2$-complete.

Proof: Recall $\text{UEXP}^2 \subseteq \mathbb{O}_2$ and prove $\mathbb{U} \subseteq \text{UEXP}$.

Omitted.

Theorem $\text{EXP}$ is $\mathbb{O}_2$-complete.

Proof: Prove $\text{EXP}^2 \subseteq \mathbb{O}_2$ and prove $\mathbb{U} \subseteq \text{EXP}$.

Omitted.
Hardness of Deciding \( \textbf{UEXP} \) and \( \textbf{EXP} \)

**Lemma**

\( \textbf{UEXP} \) is \( \Sigma^0_2 \)-complete.
Hardness of Deciding $\text{UEXP}$ and $\text{EXP}$

Lemma

$\text{UEXP}$ is $\Sigma^0_2$–complete.

Proof: Recall $\text{UEXP} \in \Sigma^0_2$ and prove $\overline{\text{UH}} \leq \text{UEXP}$.
Hardness of Deciding $\text{UEXP}$ and $\text{EXP}$

Lemma

$\text{UEXP}$ is $\Sigma_2^0$–complete.

Proof: Recall $\text{UEXP} \in \Sigma_2^0$ and prove $\overline{\text{UH}} \leq \text{UEXP}$. Omitted.
Hardness of Deciding $\text{UEXP}$ and $\text{EXP}$

Lemma

$\text{UEXP}$ is $\Sigma^0_2$–complete.

Proof: Recall $\text{UEXP} \in \Sigma^0_2$ and prove $\overline{\text{UH}} \leq \text{UEXP}$. Omitted.

Theorem

$\text{EXP}$ is $\Pi^0_2$–complete.
Hardness of Deciding $\text{UEXP}$ and $\text{EXP}$

**Lemma**

$\text{UEXP}$ is $\Sigma^0_2$–complete.

Proof: Recall $\text{UEXP} \in \Sigma^0_2$ and prove $\overline{\text{UH}} \leq \text{UEXP}$. Omitted.

**Theorem**

$\text{EXP}$ is $\Pi^0_2$–complete.

Proof: Prove $\text{EXP} \in \Pi^0_2$ and prove $\overline{\text{UH}} \leq \text{EXP}$. Omitted.
Hardness of Deciding $\text{UEXP}$ and $\text{EXP}$

**Lemma**

$\text{UEXP}$ is $\Sigma_2^0$–complete.

Proof: Recall $\text{UEXP} \in \Sigma_2^0$ and prove $\overline{\text{un}} \leq \text{UEXP}$. Omitted.

**Theorem**

$\text{EXP}$ is $\Pi_2^0$–complete.

Proof: Prove $\text{EXP} \in \Pi_2^0$ and prove $\text{un} \leq \text{EXP}$. Omitted.
Hardness of Deciding $AST$
Hardness of Deciding Almost–Sure Termination
Hardness of Deciding Almost–Sure Termination

Lemma

\[ \text{AST} \in \Pi_2^0 \]
Hardness of Deciding Almost–Sure Termination

Lemma

\( \mathcal{AST} \in \Pi^0_2 \)

Proof: Prove \( \mathcal{AST} \leq \text{EXP} \).
Hardness of Deciding Almost-Sure Termination

Lemma

\[ \text{AST} \in \Pi_2^0 \]

Proof: Prove \( \text{AST} \leq \text{EXP} \). Omitted.
Hardness of Deciding Almost–Sure Termination

Lemma

\[ \text{AST} \in \Pi^0_2 \]

Proof: Prove \( \text{AST} \leq \text{EXP} \). Omitted.

Theorem

\( \text{AST} \) is \( \Pi^0_2 \)–complete.
Hardness of Deciding Almost–Sure Termination

Lemma

$$\mathcal{AST} \in \Pi^0_2$$

Proof: Prove $$\mathcal{AST} \leq \mathcal{EXP}$$. Omitted.

Theorem

$$\mathcal{AST}$$ is $$\Pi^0_2$$–complete.

Proof: Recall $$\mathcal{AST} \in \Pi^0_2$$ and prove $$\mathcal{UH} \leq \mathcal{AST}$$.
Hardness of Deciding Almost–Sure Termination

Lemma

\( \text{AST} \in \Pi_2^0 \)

Proof: Prove \( \text{AST} \leq \text{EXP} \). Omitted.

Theorem

\( \text{AST} \) is \( \Pi_2^0 \)–complete.

Proof: Recall \( \text{AST} \in \Pi_2^0 \) and prove \( \text{UH} \leq \text{AST} \). Not omitted!
Proof of $\mathcal{UH} \leq \mathcal{AST}$

Proof obligation for $\mathcal{UH} \leq \mathcal{AST}$
Proof of $\mathcal{UH} \leq \mathcal{AST}$

Proof obligation for $\mathcal{UH} \leq \mathcal{AST}$

Find a computable function $f$, such that

\[ Q \in \mathcal{UH} \iff f(Q) \in \mathcal{AST} \]
Proof of $UH \leq AST$

Proof obligation for $UH \leq AST$

Find a computable function $f$, such that

$$Q \in UH \iff f(Q) \in AST$$

First observation

There is a computable enumeration $g_Q: \mathbb{N} \to \{\text{Var} \to \mathbb{Q}\}$ enumerating all possible variable valuations subject to

$$\forall i \in \mathbb{N} \ \forall v \in \text{Var}: \left[ [g_Q(i)](v) \neq 0 \right] \implies \left[ v \text{ occurs in } Q \right]$$
Proof of $\mathcal{UH} \leq \mathsf{AST}$

Candidate for $f$
Proof of $\mathcal{UH} \leq \mathcal{AST}$

**Candidate for $f$**

$f(Q)$ returns the following probabilistic program $P$:
Proof of $\mathcal{UH} \leq \mathcal{AST}$

Candidate for $f$

$f(Q)$ returns the following probabilistic program $P$:

\[
i := 0; \{\text{continue} := 0\} [0.5] \{\text{continue} := 1\}; \\
\text{while} (\text{continue} \neq 0)\{ \\
\hspace{1em} i := i + 1; \\
\hspace{2em} \{\text{continue} := 0\} [0.5] \{\text{continue} := 1\} \\
\}; \\
TQ
\]
Proof of $\mathcal{U}H \leq AST$

Candidate for $f$

$f(Q)$ returns the following probabilistic program $P$:

\[
i := 0; \{\text{continue := 0}\} \ [0.5] \ \{\text{continue := 1}\}; \\
\text{while} \ (\text{continue} \neq 0)\{ \\
\quad i := i + 1; \\
\quad \{\text{continue := 0}\} \ [0.5] \ \{\text{continue := 1}\} \\
\}; \\
TQ
\]

where $TQ$ is a program that simulates $Q$ on input $g_Q(i)$. 
Partial Correctness
Partial Correctness

Probabilistic Program $P$ Returned by $f(Q)$

\[
i := 0; \{ \text{continue} := 0 \} \ [0.5] \ { \text{continue} := 1} \};
\]
\[
\text{while} \ (\text{continue} \neq 0) \{
\quad i := i + 1;
\quad \{ \text{continue} := 0 \} \ [0.5] \ { \text{continue} := 1} \};
\]
\[
TQ
\]
Partial Correctness

Probabilistic Program $P$ Returned by $f(Q)$

\[ i := 0; \{\text{continue} := 0\} [0.5] \{\text{continue} := 1\}; \]
\[ \text{while} \ (\text{continue} \neq 0)\{ \]
\[ i := i + 1; \]
\[ \{\text{continue} := 0\} [0.5] \{\text{continue} := 1\} \]
\[ \} ; \]
\[ TQ \]

- While–loop establishes geometric distribution on $i$
Partial Correctness

Probabilistic Program $P$ Returned by $f(Q)$

```
i := 0; {continue := 0} [0.5] {continue := 1};
while (continue $\neq$ 0){
    i := i + 1;
    {continue := 0} [0.5] {continue := 1}
};
TQ
```

- While-loop establishes geometric distribution on $i$ — hence, by $g_Q(i)$, a geometric distribution on all possible inputs for $Q$
Partial Correctness

Probabilistic Program $P$ Returned by $f(Q)$

\[
i := 0; \{\text{continue := 0}\} \ [0.5] \ {\text{continue := 1}}; \\
\text{while} \ (\text{continue } \neq 0)\{ \\
\quad i := i + 1; \\
\quad \{\text{continue := 0}\} \ [0.5] \ {\text{continue := 1}} \\
\}; \\
TQ
\]

- While–loop establishes geometric distribution on $i$ — hence, by $g_Q(i)$, a geometric distribution on all possible inputs for $Q$
- Then program $P$ terminates with probability $\sum_{k \in \mathbb{N}} \frac{1}{2^k} = 1$ iff the simulation of $Q$ terminates on every possible input $g_Q(i)$
Total Correctness
Total Correctness

- Code for $g_Q$ is computable
- Code for simulation of $Q$ on a given input is computable
Total Correctness

- Code for $g_Q$ is computable
- Code for simulation of $Q$ on a given input is computable
- So in total, program code for $P$ is computable
Total Correctness

- Code for $g_Q$ is computable
- Code for simulation of $Q$ on a given input is computable
- So in total, program code for $P$ is computable
- All of the above computations terminate
Total Correctness

- Code for $g_Q$ is computable
- Code for simulation of $Q$ on a given input is computable
- So in total, program code for $P$ is computable
- All of the above computations terminate
Summary
Summary

\[ \Sigma_2^0 \quad \Delta_2^0 \quad \Pi_2^0 \]

\[ \Sigma_1^0 \quad \Delta_1^0 \quad \Pi_1^0 \]

\[ \mathcal{H} \quad \overline{\mathcal{H}} \quad \overline{\mathcal{H}} \quad \mathcal{H} \]

\[ \mathcal{U} \mathcal{H} \quad \mathcal{U} \mathcal{H} \]

Thank you for your kind attention :-)

Benjamin Kaminski
Summary
Summary

Summary of the hardnesses of deciding ASTs:

- \( \Sigma^0_2 \) and \( \Pi^0_2 \) are semi-decidable with access to \( H \)-oracle.
- \( \Delta^0_2 \) is not semi-decidable, even with access to \( H \)-oracle.

THANK YOU FOR YOUR KIND ATTENTION :-)

Benjamin Kaminski
Summary

\[
\begin{align*}
\Sigma_2^0 & \quad \mathcal{UEXP} \\
\overline{\mathcal{UH}} & \\
\Sigma_1^0 & \quad \mathcal{LEXP} \\
\mathcal{H} & \\
\Delta_2^0 & \\
\Pi_2^0 & \\
\overline{\mathcal{UH}} & \\
\Delta_1^0 & \\
\Pi_1^0 & \\
\overline{\mathcal{H}} &
\end{align*}
\]

semi-decidable

Thank you for your kind attention :-)

Benjamin Kaminski

Analyzing Probabilistic Programs is Hard 28.5.2014
Summary

with access to $\mathcal{H}$–oracle: semi–decidable

semi–decidable
Summary

with access to $\mathcal{H}$–oracle: semi–decidable

semi–decidable
Summary

with access to $\mathcal{H}$–oracle: semi–decidable

semi–decidable
Summary

with access to $\mathcal{H}$–oracle: semi–decidable

semi–decidable

not semi–decidable; even with access to $\mathcal{H}$–oracle
Summary

Thank you for your kind attention :-)