Weakest Precondition Reasoning for Expected Run–Times of Probabilistic Programs

Benjamin Kaminski Joost-Pieter Katoen

Christoph Matheja Federico Olmedo

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

25th European Symposium on Programming

19th edition of the European Joint Conferences on Theory & Practice of Software

April 4, 2016, Eindhoven, Netherlands

Probabilistic Programs

Probabilistic Programs

- Introduce randomization into computation

Probabilistic Programs

- Introduce randomization into computation
- Significant speed–up in solving difficult problems at cost of tolerating incorrect results with low probability

Probabilistic Programs

- Introduce randomization into computation
- Significant speed–up in solving difficult problems at cost of tolerating incorrect results with low probability
- Solution to problems where deterministic techniques fail:

  E.g. symmetry breaking in Dining Philosophers, Leader Election, Ethernet's randomized exponential backoff

Probabilistic Programs

- Introduce randomization into computation
- Significant speed–up in solving difficult problems at cost of tolerating incorrect results with low probability
- Solution to problems where deterministic techniques fail:

  E.g. symmetry breaking in Dining Philosophers, Leader Election, Ethernet's randomized exponential backoff

- Randomization of some sort occurs almost in any technique related used in cryptography and security

Probabilistic Programs

- Introduce randomization into computation
- Significant speed–up in solving difficult problems at cost of tolerating incorrect results with low probability
- Solution to problems where deterministic techniques fail:

  E.g. symmetry breaking in Dining Philosophers, Leader Election, Ethernet's randomized exponential backoff
- Randomization of some sort occurs almost in any technique related used in cryptography and security
- Model probability distributions in machine learning

Syntax of Probabilistic Programs

$$\begin{array}{rcl} C & \longrightarrow & \texttt{skip} \;\big|\; x \mathrel{\texttt{:=}} E \;\big|\; C;\; C \;\big|\; \{C\} \,\Box\, \{C\} \\ & & \big|\; \texttt{if}\,(\xi)\,\{C\}\,\texttt{else}\,\{C\} \;\big|\; \texttt{while}\,(\xi)\,\{C\} \end{array}$$

Syntax of Probabilistic Programs

$$\begin{aligned} C \quad \longrightarrow \quad & \texttt{skip} \;\big|\; x \;\texttt{:=}\; E \;\big|\; C;\ C \;\big|\; \{C\} \,\Box\, \{C\} \\ & \big|\; \texttt{if}\,(\xi)\,\{C\}\,\texttt{else}\,\{C\} \;\big|\; \texttt{while}\,(\xi)\,\{C\} \end{aligned}$$

Syntax of Probabilistic Programs

$$
\begin{aligned}
C \quad \longrightarrow \quad & \texttt{skip} \;\Big|\; x \texttt{ := } E \;\Big|\; C;\ C \;\Big|\; \{C\} \,\Box\, \{C\} \\
& \Big|\; \texttt{if}\,(\xi)\,\{C\}\,\texttt{else}\,\{C\} \;\Big|\; \texttt{while}\,(\xi)\,\{C\}
\end{aligned}
$$

Syntax of Probabilistic Programs

$$C \quad \longrightarrow \quad \texttt{skip} \;\Big|\; x \mathrel{\texttt{:=}} E \;\Big|\; C;\; C \;\Big|\; \{C\} \,\square\, \{C\}$$
$$\Big|\; \texttt{if}\,(\xi)\,\{C\}\,\texttt{else}\,\{C\} \;\Big|\; \texttt{while}\,(\xi)\,\{C\}$$

Syntax of Probabilistic Programs

$$
\begin{aligned}
C \quad \longrightarrow \quad & \texttt{skip} \;\Big|\; x \mathrel{\texttt{:=}} E \;\Big|\; C;\ C \;\Big|\; \{C\} \,\square\, \{C\} \\
& \Big|\; \texttt{if}\,(\xi)\,\{C\}\,\texttt{else}\,\{C\} \;\Big|\; \texttt{while}\,(\xi)\,\{C\}
\end{aligned}
$$

Syntax of Probabilistic Programs

$$
\begin{aligned}
C \quad \longrightarrow \quad & \texttt{skip} \;\big|\; x \mathrel{\texttt{:=}} E \;\big|\; C;\ C \;\big|\; \{C\} \,\square\, \{C\} \\
& \big|\; \texttt{if}\,(\xi)\,\{C\}\,\texttt{else}\,\{C\} \;\big|\; \texttt{while}\,(\xi)\,\{C\}
\end{aligned}
$$

Syntax of Probabilistic Programs

$$
\begin{aligned}
C \quad \longrightarrow \quad & \texttt{skip} \;\big|\; x \texttt{ := } E \;\big|\; C;\ C \;\big|\; \{C\} \,\Box\, \{C\} \\
& \big|\; \texttt{if}\,(\xi)\,\{C\}\,\texttt{else}\,\{C\} \;\big|\; \texttt{while}\,(\xi)\,\{C\}
\end{aligned}
$$

Syntax of Probabilistic Programs

$$
\begin{aligned}
C \quad \longrightarrow \quad & \texttt{skip} \;\Big|\; x \texttt{ := } E \;\Big|\; C;\ C \;\Big|\; \{C\} \,\square\, \{C\} \\
& \Big|\; \texttt{if}\,(\xi)\,\{C\}\,\texttt{else}\,\{C\} \;\Big|\; \texttt{while}\,(\xi)\,\{C\}
\end{aligned}
$$

What is probabilistic about that language?

Syntax of Probabilistic Programs

$$\begin{array}{rcl} C & \longrightarrow & \texttt{skip} \;\big|\; x \mathrel{\texttt{:=}} E \;\big|\; C;\ C \;\big|\; \{C\} \mathbin{\square} \{C\} \\ & & \big|\; \texttt{if}\,(\xi)\,\{C\}\,\texttt{else}\,\{C\} \;\big|\; \texttt{while}\,(\xi)\,\{C\} \end{array}$$

What is probabilistic about that language?

Probabilistic guards $\xi\colon \Sigma \to \mathcal{D}(\{\text{true}, \text{false}\})$:

Syntax of Probabilistic Programs

$$C \quad \longrightarrow \quad \texttt{skip} \;\Big|\; x \texttt{ := } E \;\Big|\; C;\ C \;\Big|\; \{C\} \,\Box\, \{C\}$$
$$\Big|\; \texttt{if}\,(\xi)\,\{C\}\,\texttt{else}\,\{C\} \;\Big|\; \texttt{while}\,(\xi)\,\{C\}$$

What is probabilistic about that language?

Probabilistic guards $\xi \colon \Sigma \to \mathcal{D}(\{\text{true}, \text{false}\})$:

- $[\![\xi\colon \text{true}]\!](\sigma) = 1 - [\![\xi\colon \text{false}]\!](\sigma)$ is the probability of ξ evaluating to true

Syntax of Probabilistic Programs

$$C \quad \longrightarrow \quad \texttt{skip} \;\Big|\; x \texttt{ := } E \;\Big|\; C;\ C \;\Big|\; \{C\} \,\Box\, \{C\}$$
$$\Big|\; \texttt{if}\,(\xi)\,\{C\}\,\texttt{else}\,\{C\} \;\Big|\; \texttt{while}\,(\xi)\,\{C\}$$

What is probabilistic about that language?

Probabilistic guards $\xi \colon \Sigma \to \mathcal{D}(\{\mathsf{true}, \mathsf{false}\})$:

- $[\![\xi \colon \mathsf{true}]\!](\sigma) = 1 - [\![\xi \colon \mathsf{false}]\!](\sigma)$ is the probability of ξ evaluating to true
- E.g. $\frac{2}{3}\langle\mathsf{true}\rangle + \frac{1}{3}\langle\mathsf{false}\rangle$

Syntax of Probabilistic Programs

$$C \quad \longrightarrow \quad \texttt{skip} \;\Big|\; x \mathrel{\texttt{:=}} E \;\Big|\; C;\ C \;\Big|\; \{C\} \,\square\, \{C\}$$
$$\Big|\; \texttt{if}\,(\xi)\,\{C\}\,\texttt{else}\,\{C\} \;\Big|\; \texttt{while}\,(\xi)\,\{C\}$$

What is probabilistic about that language?

Probabilistic guards $\xi \colon \Sigma \to \mathcal{D}(\{\mathsf{true}, \mathsf{false}\})$:

- $[\![\xi\colon \mathsf{true}]\!](\sigma) = 1 - [\![\xi\colon \mathsf{false}]\!](\sigma)$ is the probability of ξ evaluating to true
- E.g. $\frac{2}{3}\langle\mathsf{true}\rangle + \frac{1}{3}\langle\mathsf{false}\rangle$, $\quad \frac{1}{2}\langle x > y\rangle + \frac{1}{2}\langle x \geq y\rangle$

Probabilistic Programs

What does a probabilistic program C do?

Probabilistic Programs

What does a probabilistic program C do?

- Run program C on initial state σ

Probabilistic Programs

What does a probabilistic program C do?

- Run program C on initial state σ
- Obtain final set of distributions μ over terminal states

Probabilistic Programs

What does a probabilistic program C do?

- Run program C on initial state σ
- Obtain final set of (sub–)distributions μ over terminal states

Probabilistic Programs

What does a probabilistic program C do?

- Run program C on initial state σ
- Obtain final set of (sub–)distributions μ over terminal states

What is the run–time of C on input σ?

Probabilistic Programs

What does a probabilistic program C do?

- Run program C on initial state σ
- Obtain final set of (sub–)distributions μ over terminal states

What is the run–time of C on input σ?

- Behavior of C not entirely determined by σ

Probabilistic Programs

What does a probabilistic program C do?

- Run program C on initial state σ
- Obtain final set of (sub–)distributions μ over terminal states

What is the run–time of C on input σ?

- Behavior of C not entirely determined by σ
- Probabilistic nature of C influences its run–time

Probabilistic Programs

What does a probabilistic program C do?

- Run program C on initial state σ
- Obtain final set of (sub–)distributions μ over terminal states

What is the run–time of C on input σ?

- Behavior of C not entirely determined by σ
- Probabilistic nature of C influences its run–time

Better Question:

What is the expected run–time (ERT) of C on input σ?

Expected Run–Time Phenomena

Expected Run–Time Phenomena

- ERT of C can be finite even if C admits infinite computations

Expected Run–Time Phenomena

- ERT of C can be finite even if C admits infinite computations

---

$$x \texttt{ := } 1;\ \texttt{while}\,(1/2)\,\{x \texttt{ := } 2 \cdot x\}$$

Expected Run–Time Phenomena

- ERT of C can be finite even if C admits infinite computations

$$x \texttt{ := } 1;\ \texttt{while}\,(1/2)\,\{x \texttt{ := } 2 \cdot x\}$$

Expected Run–Time Phenomena

- ERT of C can be finite even if C admits infinite computations
- Positive almost–sure termination:

$$x \texttt{ := } 1;\ \texttt{while}\,(1/2)\,\{x \texttt{ := } 2 \cdot x\}$$

Expected Run–Time Phenomena

- ERT of C can be finite even if C admits infinite computations
- Positive almost–sure termination:
  - ERT of C is finite

$$x \texttt{ := } 1;\ \texttt{while}\,(1/2)\,\{x \texttt{ := } 2 \cdot x\}$$

Expected Run–Time Phenomena

- ERT of C can be finite even if C admits infinite computations
- Positive almost–sure termination:
  - ERT of C is finite

$$x \texttt{ := } 1;\ \texttt{while}\,(1/2)\,\{x \texttt{ := } 2 \cdot x\};$$
$$\texttt{while}\,(x > 0)\,\{x \texttt{ := } x - 1\}$$

Expected Run–Time Phenomena

- ERT of C can be finite even if C admits infinite computations
- Positive almost–sure termination:
  - ERT of C is finite
  - Positively almost–surely terminating programs are not closed under sequential composition

---

$$x \texttt{ := } 1;\ \texttt{while}\,(1/2)\,\{x \texttt{ := } 2 \cdot x\};$$
$$\texttt{while}\,(x > 0)\,\{x \texttt{ := } x - 1\}$$

Expected Run–Time Phenomena

- ERT of C can be finite even if C admits infinite computations
- Positive almost–sure termination:
  - ERT of C is finite
  - Positively almost–surely terminating programs are not closed under sequential composition
  - Reasoning about positive almost–sure termination is computationally very difficult:

$$x \texttt{ := } 1;\ \texttt{while}\,(1/2)\,\{x \texttt{ := } 2 \cdot x\};$$
$$\texttt{while}\,(x > 0)\,\{x \texttt{ := } x - 1\}$$

Expected Run–Time Phenomena

- ERT of C can be finite even if C admits infinite computations
- Positive almost–sure termination:
  - ERT of C is finite
  - Positively almost–surely terminating programs are not closed under sequential composition
  - Reasoning about positive almost–sure termination is computationally very difficult:

    Strictly more difficult than the termination problem for non–probabilistic programs [MFCS 2015]

$$x \mathrel{\texttt{:=}} 1;\ \texttt{while}\,(1/2)\ \{x \mathrel{\texttt{:=}} 2 \cdot x\};$$
$$\texttt{while}\,(x > 0)\ \{x \mathrel{\texttt{:=}} x - 1\}$$

Expected Run–Time Phenomena

- ERT of C can be finite even if C admits infinite computations
- Positive almost–sure termination:
  - ERT of C is finite
  - Positively almost–surely terminating programs are not closed under sequential composition
  - Reasoning about positive almost–sure termination is computationally very difficult:

    Strictly more difficult than the termination problem for non–probabilistic programs [MFCS 2015]
- ERT of C can be infinite, even if C terminates almost–surely[1]

$$x \texttt{ := } 1;\ \texttt{while}\,(1/2)\,\{x \texttt{ := } 2 \cdot x\};$$
$$\texttt{while}\,(x > 0)\,\{x \texttt{ := } x - 1\}$$

[1] i.e. with probability 1

Expected Run–Times

Expected Run–Times

- ERT if C terminates almost–surely on σ:

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

Expected Run–Times

- ERT if C terminates almost–surely on σ:

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

- ERT if C does not terminate almost–surely on σ:

$$\infty$$

Expected Run–Times

- ERT if C terminates almost–surely on σ:

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

- ERT if C does not terminate almost–surely on σ:

$$\infty$$

- In general: ERT of C is a function

$$t \colon \Sigma \to \mathbb{R}_{\geq 0}^{\infty}$$

Expected Run–Times

- ERT if C terminates almost–surely on σ:

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

- ERT if C does not terminate almost–surely on σ:

$$\infty$$

- In general: ERT of C is a function

$$t \colon \Sigma \to \mathbb{R}_{\geq 0}^{\infty}$$

- Call such a t a run–time.

Expected Run–Times

- ERT if C terminates almost–surely on σ:

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

- ERT if C does not terminate almost–surely on σ:

$$\infty$$

- In general: ERT of C is a function

$$t\colon \Sigma \to \mathbb{R}_{\geq 0}^{\infty}$$

- Call such a t a run–time. Denote set of run–times by $\mathbb{T}$.

Expected Run–Times

- ERT if C terminates almost–surely on σ:

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

- ERT if C does not terminate almost–surely on σ:

$$\infty$$

- In general: ERT of C is a function

$$t\colon \Sigma \to \mathbb{R}_{\geq 0}^{\infty}$$

- Call such a t a run–time. Denote set of run–times by $\mathbb{T}$.
- Complete partial order on $\mathbb{T}$:

$$t_1 \preceq t_2 \quad \text{iff} \quad \forall\, \sigma \in \Sigma\colon t_1(\sigma) \leq t_2(\sigma)$$

Weakest Precondition Reasoning for Expected Run–Times

The ert Transformer

Weakest Precondition Reasoning for Expected Run–Times

The ert Transformer

Use a continuation passing style ERT transformer $\mathsf{ert}[C]\colon\ \mathbb{T} \rightarrow \mathbb{T}$.

Weakest Precondition Reasoning for Expected Run–Times

The ert Transformer

Use a continuation passing style ERT transformer $\mathrm{ert}[C]\colon\ \mathbb{T} \rightarrow \mathbb{T}$.

$$C$$

Weakest Precondition Reasoning for Expected Run–Times

The ert Transformer

Use a continuation passing style ERT transformer $\text{ert}[C]\colon\ \mathbb{T} \to \mathbb{T}$.

$$C \qquad \underset{\substack{\uparrow \\ \text{time needed} \\ \text{after executing } C}}{t}$$

Weakest Precondition Reasoning for Expected Run–Times

The ert Transformer

Use a continuation passing style ERT transformer $\mathsf{ert}[C]\colon\ \mathbb{T} \rightarrow \mathbb{T}$.

$C \qquad t$

time needed
after executing C

Weakest Precondition Reasoning for Expected Run–Times

The ert Transformer

Use a continuation passing style ERT transformer $\text{ert}[C]\colon\ \mathbb{T} \rightarrow \mathbb{T}$.

$$\text{ert}\,[C]\,(t) \qquad C \qquad t$$

expected time needed before executing C

time needed after executing C

Weakest Precondition Reasoning for Expected Run–Times

The ert Transformer

Use a continuation passing style ERT transformer $\mathsf{ert}[C] \colon \ \mathbb{T} \to \mathbb{T}$.

$$\mathsf{ert}\,[C]\,(t) \qquad\qquad C \qquad\qquad t$$

expected time needed before executing C

time needed after executing C

ERT in Terms of ert

$$\mathsf{ert}\,[C]\,(\mathbf{0})\,(\sigma) \;=\; \text{“ERT of } C \text{ on input } \sigma\text{”}$$

Rules for the ert Transformer

C	$\textbf{ert}\,[C]\,(t)$
`skip`	$\mathbf{1} + t$

Rules for the ert Transformer

C	$\textbf{ert}\,[C]\,(t)$
`skip`	$\mathbf{1} + t$
x `:=` E	$\mathbf{1} + t\,[x/E]$

Rules for the ert Transformer

C	$\textbf{ert}\,[C]\,(t)$
`skip`	$\mathbf{1} + t$
x `:=` E	$\mathbf{1} + t\,[x/E]$

Rules for the ert Transformer

C	$\textbf{ert}\,[C]\,(t)$
`skip`	$\mathbf{1} + t$
x `:=` E	$\mathbf{1} + t\,[x/E]$
$C_1\,;\,C_2$	$\mathsf{ert}\,[C_1]\,(\mathsf{ert}\,[C_2]\,(t))$

Rules for the ert Transformer

$\boldsymbol{C}$	$\textbf{ert}\,[C]\,(t)$
`skip`	$\mathbf{1} + t$
x `:=` E	$\mathbf{1} + t\,[x/E]$
$C_1\,;\,C_2$	$\mathsf{ert}\,[C_1]\,(\mathsf{ert}\,[C_2]\,(t))$
$\{C_1\}\,\square\,\{C_2\}$	$\max\{\mathsf{ert}\,[C_1]\,(t)\,,\,\mathsf{ert}\,[C_2]\,(t)\}$

Rules for the ert Transformer

C	$\mathsf{ert}\,[C]\,(t)$
`skip`	$\mathbf{1} + t$
x `:=` E	$\mathbf{1} + t\,[x/E]$
$C_1\,;\,C_2$	$\mathsf{ert}\,[C_1]\,(\mathsf{ert}\,[C_2]\,(t))$
$\{C_1\} \,\Box\, \{C_2\}$	$\max\{\mathsf{ert}\,[C_1]\,(t)\,,\ \mathsf{ert}\,[C_2]\,(t)\}$
`if` $(\xi)\ \{C_1\}$ `else` $\{C_2\}$	$\mathbf{1} + [\![\xi\colon \mathsf{true}]\!] \cdot \mathsf{ert}\,[C_1]\,(t) + [\![\xi\colon \mathsf{false}]\!] \cdot \mathsf{ert}\,[C_2]\,(t)$

Rules for the ert Transformer

C	$\mathsf{ert}\,[C]\,(t)$
`skip`	$\mathbf{1} + t$
x `:=` E	$\mathbf{1} + t\,[x/E]$
$C_1\,;\,C_2$	$\mathsf{ert}\,[C_1]\,(\mathsf{ert}\,[C_2]\,(t))$
$\{C_1\} \,\square\, \{C_2\}$	$\max\{\mathsf{ert}\,[C_1]\,(t)\,,\ \mathsf{ert}\,[C_2]\,(t)\}$
`if` $(\xi)\,\{C_1\}$ `else` $\{C_2\}$	$\mathbf{1} + [\![\xi\colon \mathsf{true}]\!] \cdot \mathsf{ert}\,[C_1]\,(t) + [\![\xi\colon \mathsf{false}]\!] \cdot \mathsf{ert}\,[C_2]\,(t)$
`while` $(\xi)\,\{C'\}$	$\mathsf{lfp}\, X\,\bullet\, \mathbf{1} + [\![\xi\colon \mathsf{false}]\!] \cdot t + [\![\xi\colon \mathsf{true}]\!] \cdot \mathsf{ert}\,[C']\,(X)$

Upper Bounds for ert of Loops

Upper Bounds for ert of Loops

Recall the definition of $\mathsf{ert}\left[\mathtt{while}\,(\xi)\,\{C\}\right](t)$:

$$\mathsf{lfp}\, X \centerdot\, \mathbf{1} + [\![\xi\colon \mathsf{false}]\!] \cdot t + [\![\xi\colon \mathsf{true}]\!] \cdot \mathsf{ert}\,[C]\,(X)$$

Upper Bounds for ert of Loops

Recall the definition of ert $[\texttt{while}\,(\xi)\,\{C\}]\,(t)$:

$$\mathsf{lfp}\, X\bullet\, \underbrace{\mathbf{1} + [\![\xi\colon \mathsf{false}]\!]\cdot t + [\![\xi\colon \mathsf{true}]\!]\cdot \mathsf{ert}\,[C]\,(X)}_{=:\; F(X)}$$

Upper Bounds for ert of Loops

Recall the definition of $\mathsf{ert}\left[\texttt{while}\ (\xi)\ \{C\}\right](t)$:

$$\mathsf{lfp}\, X\,\bullet\, \underbrace{\mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}\left[C\right](X)}_{=:\ F(X)}$$

Theorem: Upper Bounds from Upper Invariants

Upper Bounds for ert of Loops

Recall the definition of $\mathsf{ert}\left[\mathtt{while}\,(\xi)\,\{C\}\right](t)$:

$$\mathsf{lfp}\, X \bullet\ \underbrace{\mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}\,[C]\,(X)}_{=:\ F(X)}$$

Theorem: Upper Bounds from Upper Invariants

If $I \in \mathbb{T}$ is an upper invariant of $\mathtt{while}\,(\xi)\,\{C\}$, i.e. if

$$F(I) \preceq I$$

Upper Bounds for ert of Loops

Recall the definition of $\mathsf{ert}\left[\mathtt{while}\,(\xi)\,\{C\}\right](t)$:

$$\mathsf{lfp}\, X\centerdot\ \underbrace{\mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}\left[C\right](X)}_{=:\ F(X)}$$

Theorem: Upper Bounds from Upper Invariants

If $I \in \mathbb{T}$ is an upper invariant of $\mathtt{while}\,(\xi)\,\{C\}$, i.e. if

$$F(I) \preceq I$$

then

$$\mathsf{ert}\left[\mathtt{while}\,(\xi)\,\{C\}\right](t) \ \preceq I\ .$$

Lower Bounds for ert of Loops

Lower Bounds for ert of Loops

Reasoning on lower bounds is more involved:

Find an argument for being below a least fixed point

Lower Bounds for ert of Loops

Reasoning on lower bounds is more involved:

Find an argument for being below a least fixed point

Theorem: Lower Bounds from Lower ω–Invariants

Lower Bounds for ert of Loops

Reasoning on lower bounds is more involved:

Find an argument for being below a least fixed point

Theorem: Lower Bounds from Lower ω–Invariants

If $\{I_n\}_{n \in \mathbb{N}} \subseteq \mathbb{T}$ is a lower ω–invariant, i.e. if

$$\begin{aligned} I_0 &\preceq F(\mathbf{0}), \quad \text{and} \\ I_{n+1} &\preceq F(I_n) \end{aligned}$$

Lower Bounds for ert of Loops

Reasoning on lower bounds is more involved:

Find an argument for being below a least fixed point

Theorem: Lower Bounds from Lower ω–Invariants

If $\{I_n\}_{n\in\mathbb{N}} \subseteq \mathbb{T}$ is a lower ω–invariant, i.e. if

$$I_0 \preceq F(\mathbf{0}), \quad \text{and}$$

$$I_{n+1} \preceq F(I_n)$$

then

$$\sup_{n\in\mathbb{N}} I_n \preceq \mathsf{ert}\left[\mathtt{while}\,(\xi)\,\{C\}\right](t) \;.$$

Theorem: Completeness of Proof Rules

The presented proof rules are complete

Theorem: Completeness of Proof Rules

The presented proof rules are complete, since $I = \mathsf{lfp}\, F$ is an upper invariant

Theorem: Completeness of Proof Rules

The presented proof rules are complete, since $I = \mathsf{lfp}\, F$ is an upper invariant and a lower ω–invariant is given by

$$I_n \ = \ \underbrace{F \circ \cdots \circ F}_{n \text{ times}}(\mathbf{0}) \ .$$

Theorem: Completeness of Proof Rules

The presented proof rules are complete, since $I = \mathsf{lfp}\, F$ is an upper invariant and a lower ω–invariant is given by

$$I_n \;=\; \underbrace{F \circ \cdots \circ F}_{n \text{ times}}(\mathbf{0}) \;.$$

Theorem: Bound Refinement

If I is an upper bound and $F(I) \preceq I$, then $F(I)$ is also an upper bound.

Theorem: Completeness of Proof Rules

The presented proof rules are complete, since $I = \mathsf{lfp}\, F$ is an upper invariant and a lower ω–invariant is given by

$$I_n \;=\; \underbrace{F \circ \cdots \circ F}_{n \text{ times}}(\mathbf{0})\;.$$

Theorem: Bound Refinement

If I is an upper bound and $F(I) \preceq I$, then $F(I)$ is also an upper bound. Dually for lower bounds.

Is the ert Calculus a Reasonable Run–Time Model?

Is the ert Calculus a Reasonable Run–Time Model?

- Correspondence to an operational semantics:

Is the ert Calculus a Reasonable Run–Time Model?

- Correspondence to an operational semantics:
  - Operational model defined in terms of a reward MDP à la [QEST 2012] and [MFPS 2015]

Is the ert Calculus a Reasonable Run–Time Model?

- Correspondence to an operational semantics:
  - Operational model defined in terms of a reward MDP à la [QEST 2012] and [MFPS 2015]
  - ert coincides with expected reward in the operational MDP

Is the ert Calculus a Reasonable Run–Time Model?

- Correspondence to an operational semantics:
  - Operational model defined in terms of a reward MDP à la [QEST 2012] and [MFPS 2015]
  - ert coincides with expected reward in the operational MDP
  - Enables bounded model checking of expected run–times

Is the ert Calculus a Reasonable Run–Time Model?

- Correspondence to an operational semantics:
  - Operational model defined in terms of a reward MDP à la [QEST 2012] and [MFPS 2015]
  - ert coincides with expected reward in the operational MDP
  - Enables bounded model checking of expected run–times
- Nielson's Hoare–style logic for reasoning about run–time orders of magnitude of *deterministic programs*:

Is the ert Calculus a Reasonable Run–Time Model?

- Correspondence to an operational semantics:
  - Operational model defined in terms of a reward MDP à la [QEST 2012] and [MFPS 2015]
  - ert coincides with expected reward in the operational MDP
  - Enables bounded model checking of expected run–times
- Nielson's Hoare–style logic for reasoning about run–time orders of magnitude of *deterministic programs*:
  - Nielson's logic relies on introducing additional *logical* variables

Is the ert Calculus a Reasonable Run–Time Model?

- Correspondence to an operational semantics:
  - Operational model defined in terms of a reward MDP à la [QEST 2012] and [MFPS 2015]
  - ert coincides with expected reward in the operational MDP
  - Enables bounded model checking of expected run–times
- Nielson's Hoare–style logic for reasoning about run–time orders of magnitude of *deterministic programs*:
  - Nielson's logic relies on introducing additional *logical* variables
  - ert is sound and complete with respect to Nielson's logic

Is the ert Calculus a Reasonable Run–Time Model?

- Correspondence to an operational semantics:
  - Operational model defined in terms of a reward MDP à la [QEST 2012] and [MFPS 2015]
  - ert coincides with expected reward in the operational MDP
  - Enables bounded model checking of expected run–times
- Nielson's Hoare–style logic for reasoning about run–time orders of magnitude of *deterministic programs*:
  - Nielson's logic relies on introducing additional *logical* variables
  - ert is sound and complete with respect to Nielson's logic
  - ert calculus is arguably easier to apply — no additional variables!

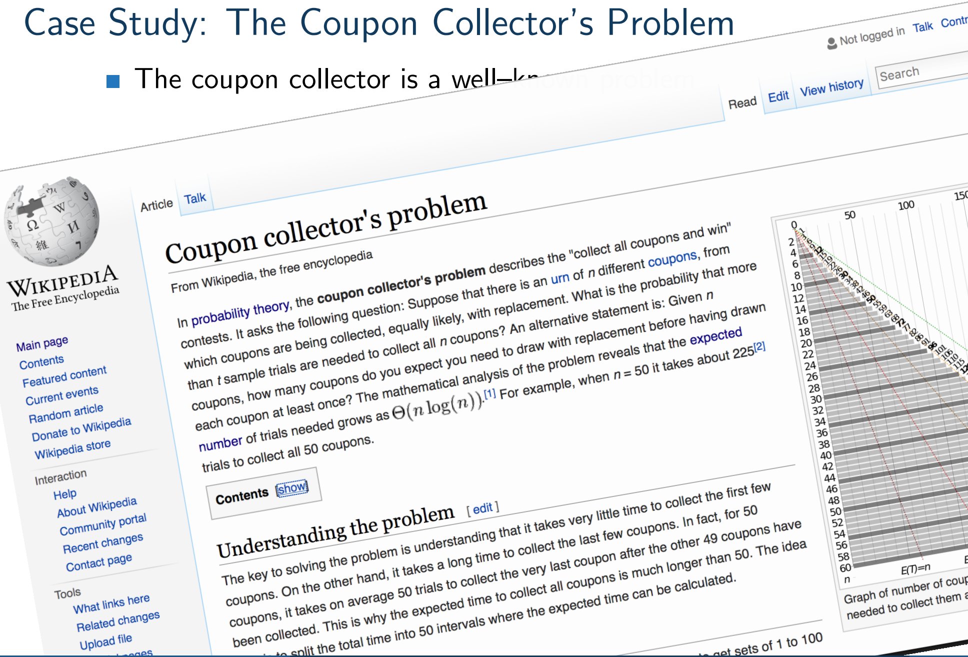
Case Study: The Coupon Collector's Problem

Case Study: The Coupon Collector's Problem

- The coupon collector is a well–known problem

Case Study: The Coupon Collector's Problem

- The coupon collector is a well–kn



Not logged in Talk Contr

Read Edit View history Search

Article Talk

Coupon collector's problem

From Wikipedia, the free encyclopedia

In probability theory, the **coupon collector's problem** describes the "collect all coupons and win" contests. It asks the following question: Suppose that there is an urn of *n* different coupons, from which coupons are being collected, equally likely, with replacement. What is the probability that more than *t* sample trials are needed to collect all *n* coupons? An alternative statement is: Given *n* coupons, how many coupons do you expect you need to draw with replacement before having drawn each coupon at least once? The mathematical analysis of the problem reveals that the expected number of trials needed grows as $\Theta(n \log(n))$.[1] For example, when $n = 50$ it takes about 225[2] trials to collect all 50 coupons.

Contents [show]

Understanding the problem [edit]

The key to solving the problem is understanding that it takes very little time to collect the first few coupons. On the other hand, it takes a long time to collect the last few coupons. In fact, for 50 coupons, it takes on average 50 trials to collect the very last coupon after the other 49 coupons have been collected. This is why the expected time to collect all coupons is much longer than 50. The idea ... to split the total time into 50 intervals where the expected time can be calculated.

... get sets of 1 to 100

WIKIPEDIA The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

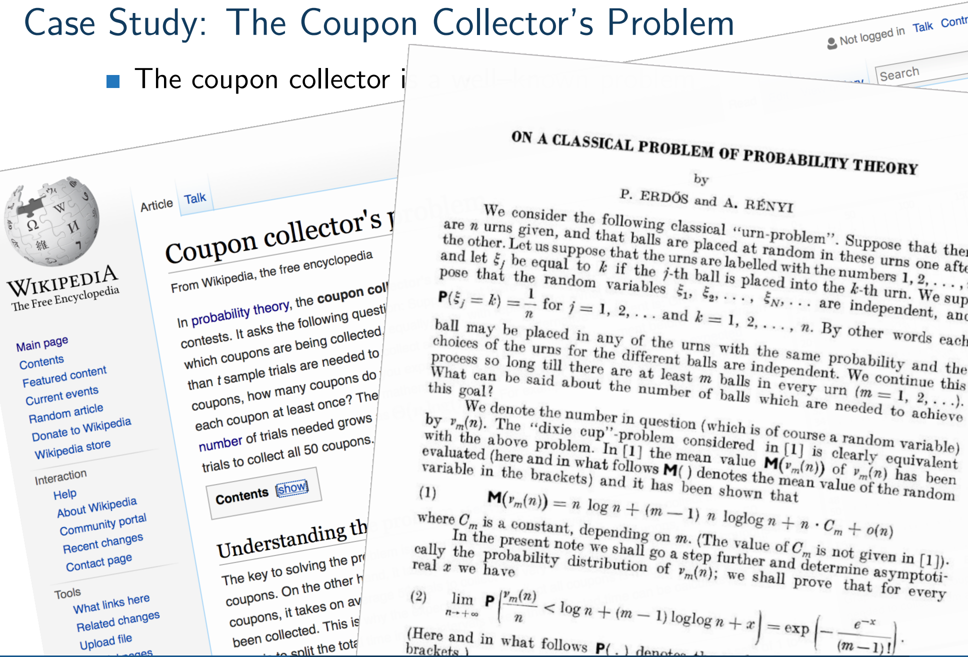
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file

Graph of number of coup... needed to collect them a...

Case Study: The Coupon Collector's Problem

- The coupon collector i[s a well-known problem]



Not logged in Talk Contr

Search

Article Talk

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file

Coupon collector's p

From Wikipedia, the free encyclopedia

In probability theory, the **coupon coll** contests. It asks the following questi which coupons are being collected. than *t* sample trials are needed to coupons, how many coupons do each coupon at least once? The number of trials needed grows trials to collect all 50 coupons.

Contents [show]

Understanding th

The key to solving the pr coupons. On the other h coupons, it takes on av been collected. This is

ON A CLASSICAL PROBLEM OF PROBABILITY THEORY

by

P. ERDŐS and A. RÉNYI

We consider the following classical "urn-problem". Suppose that ther are n urns given, and that balls are placed at random in these urns one afte the other. Let us suppose that the urns are labelled with the numbers $1, 2, \ldots,$ and let ξ_j be equal to k if the j-th ball is placed into the k-th urn. We sup pose that the random variables $\xi_1, \xi_2, \ldots, \xi_N, \ldots$ are independent, and $\mathbf{P}(\xi_j = k) = \frac{1}{n}$ for $j = 1, 2, \ldots$ and $k = 1, 2, \ldots, n$. By other words each ball may be placed in any of the urns with the same probability and the choices of the urns for the different balls are independent. We continue this process so long till there are at least m balls in every urn ($m = 1, 2, \ldots$). What can be said about the number of balls which are needed to achieve this goal?

We denote the number in question (which is of course a random variable) by $\nu_m(n)$. The "dixie cup"-problem considered in [1] is clearly equivalent with the above problem. In [1] the mean value $\mathbf{M}(\nu_m(n))$ of $\nu_m(n)$ has been evaluated (here and in what follows $\mathbf{M}(\ )$ denotes the mean value of the random variable in the brackets) and it has been shown that

$$\mathbf{M}(\nu_m(n)) = n \log n + (m-1)\, n \log\log n + n \cdot C_m + o(n) \tag{1}$$

where C_m is a constant, depending on m. (The value of C_m is not given in [1]).

In the present note we shall go a step further and determine asymptotically the probability distribution of $\nu_m(n)$; we shall prove that for every real x we have

$$\lim_{n\to+\infty} \mathbf{P}\left(\frac{\nu_m(n)}{n} < \log n + (m-1)\log\log n + x\right) = \exp\left(-\frac{e^{-x}}{(m-1)!}\right). \tag{2}$$

(Here and in what follows $\mathbf{P}(\ldots)$ denotes the ... brackets.)

Case Study: The Coupon Collector's Problem

- The coupon collector is a well–known problem

Case Study: The Coupon Collector's Problem

- The coupon collector is a well–known problem
- We model it by the following algorithm:

$$
\begin{array}{l}
cp \texttt{ := } [0, \ldots, 0];\ i \texttt{ := } 1;\ x \texttt{ := } N;\\
\texttt{while}\,(x > 0)\,\{\\
\qquad \texttt{while}\,(cp[i] \neq 0)\,\{\ i \texttt{ :}\!\approx \texttt{Unif}[1 \ldots N]\ \};\\
\qquad cp[i] \texttt{ := } 1;\ x \texttt{ := } x - 1\ \}
\end{array}
$$

Case Study: The Coupon Collector's Problem

- The coupon collector is a well–known problem
- We model it by the following algorithm:

$$
\begin{array}{l}
cp \;\texttt{:=}\; [0, \ldots, 0];\, i \;\texttt{:=}\; 1;\, x \;\texttt{:=}\; N; \\
\texttt{while}\,(x > 0)\,\{ \\
\qquad \texttt{while}\,(cp[i] \neq 0)\,\{\; i \;\texttt{:}\!\approx \texttt{Unif}[1 \ldots N] \;\}; \\
\qquad cp[i] \;\texttt{:=}\; 1;\, x \;\texttt{:=}\; x - 1 \;\}
\end{array}
$$

- Using ert, we can analyze the ERT of the above algorithm directly on the source code given above:

Case Study: The Coupon Collector's Problem

- The coupon collector is a well–known problem
- We model it by the following algorithm:

$$
\begin{array}{l}
cp \;\texttt{:=}\; [0, \ldots, 0];\; i \;\texttt{:=}\; 1;\; x \;\texttt{:=}\; N;\\
\texttt{while}\,(x > 0)\,\{\\
\qquad \texttt{while}\,(cp[i] \neq 0)\,\{\; i \;\texttt{:}\!\approx\; \texttt{Unif}[1 \ldots N]\;\};\\
\qquad cp[i] \;\texttt{:=}\; 1;\; x \;\texttt{:=}\; x - 1\;\}
\end{array}
$$

- Using ert, we can analyze the ERT of the above algorithm directly on the source code given above:

$$\mathsf{ert}\,[\mathit{coup.\ coll.}]\,(\mathbf{0}) \;=\; \mathbf{4} + [N > 0] \cdot 2N \cdot (\mathbf{2} + \mathcal{H}_{N-1})$$

Case Study: The Coupon Collector's Problem

- The coupon collector is a well–known problem
- We model it by the following algorithm:

$$
\begin{array}{l}
cp \;\texttt{:=}\; [0, \ldots, 0];\; i \;\texttt{:=}\; 1;\; x \;\texttt{:=}\; N; \\
\texttt{while}\,(x > 0)\,\{ \\
\qquad \texttt{while}\,(cp[i] \neq 0)\,\{\; i \;\texttt{:}\!\approx\; \texttt{Unif}[1 \ldots N] \;\}; \\
\qquad cp[i] \;\texttt{:=}\; 1;\; x \;\texttt{:=}\; x - 1 \;\}
\end{array}
$$

- Using ert, we can analyze the ERT of the above algorithm directly on the source code given above:

$$\mathsf{ert}\,[coup.\ coll.]\,(\mathbf{0}) \;=\; \mathbf{4} + [N > 0] \cdot 2N \cdot (\mathbf{2} + \mathcal{H}_{N-1})$$

- Harmonic number $\mathcal{H}_{N-1}$ is in $\Theta(\log N)$

Case Study: The Coupon Collector's Problem

- The coupon collector is a well–known problem
- We model it by the following algorithm:

$$
\begin{array}{l}
cp \mathrel{:=} [0, \ldots, 0];\, i \mathrel{:=} 1;\, x \mathrel{:=} N; \\
\texttt{while}\,(x > 0)\,\{ \\
\qquad \texttt{while}\,(cp[i] \neq 0)\,\{\, i \mathrel{:\approx} \texttt{Unif}[1 \ldots N] \,\}; \\
\qquad cp[i] \mathrel{:=} 1;\, x \mathrel{:=} x - 1 \;\}
\end{array}
$$

- Using ert, we can analyze the ERT of the above algorithm directly on the source code given above:

$$\mathsf{ert}\,[coup.\ coll.]\,(\mathbf{0}) \;=\; \mathbf{4} + [N > 0] \cdot 2N \cdot (\mathbf{2} + \mathcal{H}_{N-1})$$

- Harmonic number $\mathcal{H}_{N-1}$ is in $\Theta(\log N)$
- Coupon collector program runs in $\Theta(N \cdot \log N)$ for $N > 0$

Summary

Summary

- ert is an easy to understand weakest–precondition–style calculus for reasoning about ERT of probabilistic programs

Summary

- ert is an easy to understand weakest–precondition–style calculus for reasoning about ERT of probabilistic programs
- ert is sound and complete for reasoning about expected run–times and positive almost–sure termination

Summary

- ert is an easy to understand weakest–precondition–style calculus for reasoning about ERT of probabilistic programs
- ert is sound and complete for reasoning about expected run–times and positive almost–sure termination
- ert comes with proof rules for reasoning about loops

Summary

- ert is an easy to understand weakest–precondition–style calculus for reasoning about ERT of probabilistic programs
- ert is sound and complete for reasoning about expected run–times and positive almost–sure termination
- ert comes with proof rules for reasoning about loops
- ert is a powerful alternative to ranking super–martingales

Summary

- ert is an easy to understand weakest–precondition–style calculus for reasoning about ERT of probabilistic programs
- ert is sound and complete for reasoning about expected run–times and positive almost–sure termination
- ert comes with proof rules for reasoning about loops
- ert is a powerful alternative to ranking super–martingales
- ert is applicable to tricky real–world examples which are difficult to reason about by formal verification techniques

Summary

- ert is an easy to understand weakest–precondition–style calculus for reasoning about ERT of probabilistic programs
- ert is sound and complete for reasoning about expected run–times and positive almost–sure termination
- ert comes with proof rules for reasoning about loops
- ert is a powerful alternative to ranking super–martingales
- ert is applicable to tricky real–world examples which are difficult to reason about by formal verification techniques

ert is Isabelle/HOL certified (courtesy of Johannes Hölzl, TUM)

Summary

- ert is an easy to understand weakest–precondition–style calculus for reasoning about ERT of probabilistic programs
- ert is sound and complete for reasoning about expected run–times and positive almost–sure termination
- ert comes with proof rules for reasoning about loops
- ert is a powerful alternative to ranking super–martingales
- ert is applicable to tricky real–world examples which are difficult to reason about by formal verification techniques
- ert is Isabelle/HOL certified (courtesy of Johannes Hölzl, TUM)
- Future work: recursion, conditioning, run–time variance

Summary

- ert is an easy to understand weakest–precondition–style calculus for reasoning about ERT of probabilistic programs
- ert is sound and complete for reasoning about expected run–times and positive almost–sure termination
- ert comes with proof rules for reasoning about loops
- ert is a powerful alternative to ranking super–martingales
- ert is applicable to tricky real–world examples which are difficult to reason about by formal verification techniques
- ert is Isabelle/HOL certified (courtesy of Johannes Hölzl, TUM)
- Future work: recursion, conditioning, run–time variance

Thank you for your kind attention!

Backup Slides: The Actual Rule for Assignments

C	$\mathsf{ert}\,[C]\,(t)$
$x :\approx \mu$	$\mathbf{1} + \lambda\sigma_\bullet\ \mathsf{E}_{[\![\mu]\!](\sigma)}\,(\lambda v.\, t\,[x/v]\,(\sigma))$

Backup Slides: ert Calculations and Proof Rule Application

Example 4 (Geometric distribution). Consider loop

$$C_{\mathtt{geo}}\colon\quad \mathtt{while}\,(c=1)\,\{c :\approx {}^1\!/_2\cdot\langle 0\rangle + {}^1\!/_2\cdot\langle 1\rangle\}\ .$$

From the calculations below we conclude that $I = \mathbf{1} + [\![c=1]\!]\cdot\mathbf{4}$ is an upper invariant with respect to $\mathbf{0}$:

$$\begin{aligned}
&\mathbf{1} + [\![c\neq 1]\!]\cdot\mathbf{0} + [\![c=1]\!]\cdot \mathsf{ert}\,[c :\approx {}^1\!/_2\cdot\langle 0\rangle + {}^1\!/_2\cdot\langle 1\rangle]\,(I)\\
&\quad = \ \mathbf{1} + [\![c=1]\!]\cdot\big(\mathbf{1} + \tfrac{1}{2}\cdot I\,[c/0] + \tfrac{1}{2}\cdot I\,[c/1]\big)\\
&\quad = \ \mathbf{1} + [\![c=1]\!]\cdot\big(\mathbf{1} + \tfrac{1}{2}\cdot \underbrace{(\mathbf{1} + [\![0=1]\!]\cdot\mathbf{4})}_{=\mathbf{1}} + \tfrac{1}{2}\cdot \underbrace{(\mathbf{1} + [\![1=1]\!]\cdot\mathbf{4})}_{=\mathbf{5}}\big)\\
&\quad = \ \mathbf{1} + [\![c=1]\!]\cdot\mathbf{4} \ = \ I \ \preceq \ I
\end{aligned}$$

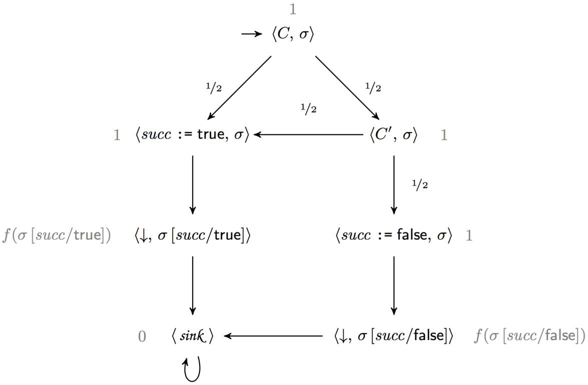
Then applying Theorem 3 we obtain

$$\mathsf{ert}\,[C_{\mathtt{geo}}]\,(\mathbf{0}) \ \preceq \ \mathbf{1} + [\![c=1]\!]\cdot\mathbf{4}\ .$$

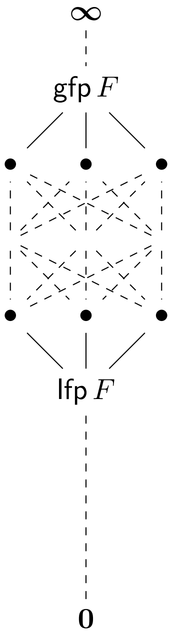
In words, the expected run–time of $C_{\mathtt{geo}}$ is at most 5 from any initial state where $c=1$ and at most 1 from the remaining states. △

Backup Slides: Operational RMDP

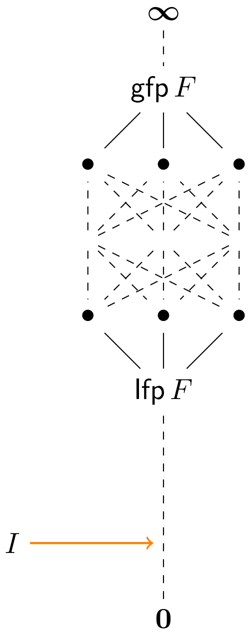
$$C_{trunc}\colon\quad \texttt{if}\ \left({}^1\!/_2\cdot\langle\mathsf{true}\rangle + {}^1\!/_2\cdot\langle\mathsf{false}\rangle\right)\ \{\mathit{succc} := \mathsf{true}\}\ \texttt{else}\ \{$$
$$\texttt{if}\ \left({}^1\!/_2\cdot\langle\mathsf{true}\rangle + {}^1\!/_2\cdot\langle\mathsf{false}\rangle\right)\ \{\mathit{succ} := \mathsf{true}\}$$
$$\texttt{else}\ \{\mathit{succ} := \mathsf{false}\}$$
$$\}$$



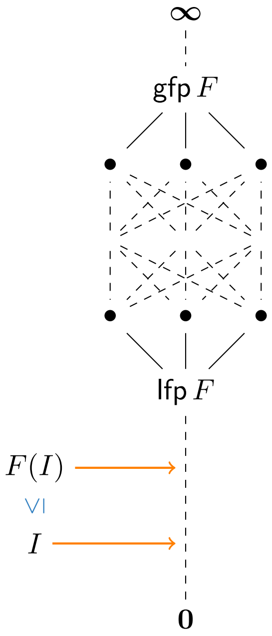
Backup Slides: Park's Lemma



Backup Slides: Park's Lemma

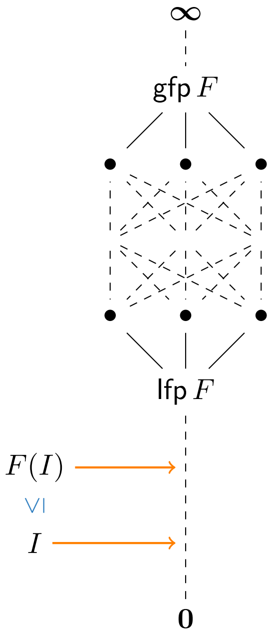


Backup Slides: Park's Lemma



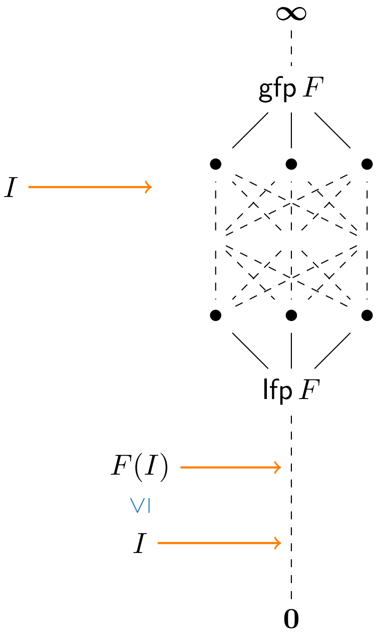
Backup Slides: Park's Lemma

$F(I) \leq I$ implies $\mathsf{lfp}\, F \leq I$



Backup Slides: Park's Lemma

$F(I) \leq I$ implies $\mathsf{lfp}\, F \leq I$



Backup Slides: Park's Lemma

$F(I) \leq I$ implies $\mathsf{lfp}\, F \leq I$