Probabilistic Weakest Preconditions

Joost-Pieter Katoen

with Kevin Batz (RWTH), Benjamin Kaminski (Saarland),
and Christoph Matheja (Lyngby)

February 3, 2022, LI2022 Workshop

Overview

Probabilistic programs are hard to grasp

Does this program almost surely terminate?

```
x := 1;
while (x > 0) {
    x := x+2 [1/2] x := x-1
}
```

Probabilistic programs are hard to grasp

Does this program almost surely terminate?

```
x := 1;
while (x > 0) {
    x := x+2 [1/2] x := x-1
}
```

If not, what is its probability to diverge?

Probabilistic programs are hard to grasp

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y;
t := t+1 [5/9] skip;
r := 1;
for i in 1..3 {
   s := iid(bernouilli(1/2), 2t);
   if (s != t) { r := 0 } else skip
}
```

Probabilistic programs are hard to grasp

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y;
t := t+1 [5/9] skip;
r := 1;
for i in 1..3 {
   s := iid(bernouilli(1/2), 2t);
   if (s != t) { r := 0 } else skip
}
```

What is the probability that r equals one on termination?

[Flajolet *et al.*, 2009]

Our objective

A powerful, simple proof calculus for probabilistic programs.

At the source code level.

No "descend" into the underlying probabilistic model.

Our objective

A powerful, simple proof calculus for probabilistic programs.

At the source code level.

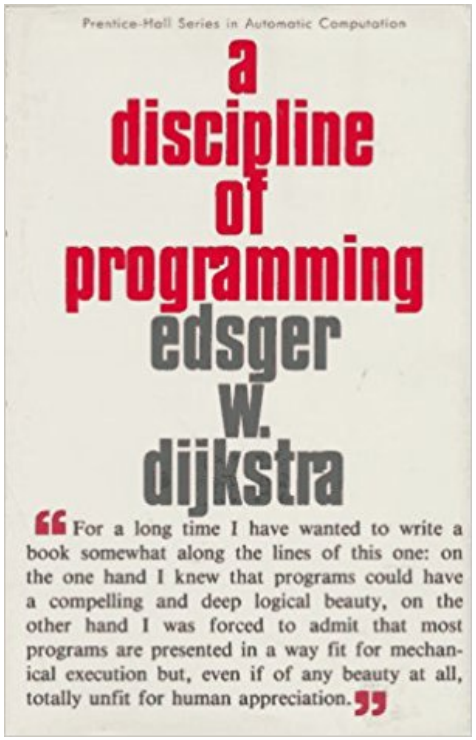
No "descend" into the underlying probabilistic model.

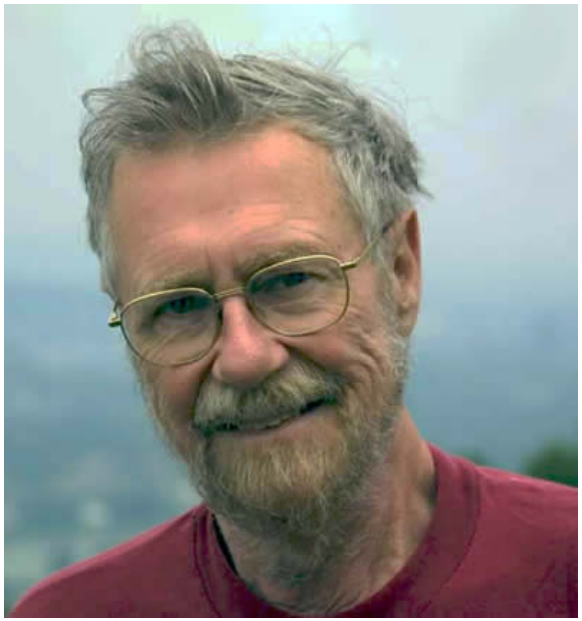
Push automation as much as we can.

Overview

A discipline of programming

WEAKEST PRECONDITIONS





Edsger Wybe Dijkstra
(1930–2002)

Predicate transformers

Let the set of states be:

$$\mathbb{S} \;=\; \{\, s \mid s : \mathit{Vars} \to \mathbb{Q}_{\geq 0} \,\}$$

Predicate transformers

Let the set of states be:

$$\mathbb{S} = \{ s \mid s : Vars \to \mathbb{Q}_{\geq 0} \}$$

Let the set of predicates:

$$\mathbb{P} = \left\{ F \mid F : \underbrace{\mathbb{S}}_{\text{states}} \to \{0, 1\} \right\}$$

Predicate transformers

Let the set of states be:

$$\mathbb{S} = \{ s \mid s : Vars \to \mathbb{Q}_{\geq 0} \}$$

Let the set of predicates:

$$\mathbb{P} = \left\{ F \mid F : \underbrace{\mathbb{S}}_{\text{states}} \to \{0, 1\} \right\}$$

$(\mathbb{P}, \sqsubseteq)$ is a complete lattice where $F \sqsubseteq G$ if and only if $F \Rightarrow G$

Predicate transformers

Let the set of states be:

$$\mathbb{S} = \{ s \mid s : Vars \to \mathbb{Q}_{\geq 0} \}$$

Let the set of predicates:

$$\mathbb{P} = \left\{ F \mid F : \underbrace{\mathbb{S}}_{\text{states}} \to \{0, 1\} \right\}$$

$(\mathbb{P}, \sqsubseteq)$ is a complete lattice where $F \sqsubseteq G$ if and only if $F \Rightarrow G$

Function $\Phi : \mathbb{P} \to \mathbb{P}$ is a predicate transformer

Weakest preconditions

For program P, let $wp[\![P]\!] : \mathbb{P} \to \mathbb{P}$ a predicate transformer.

Weakest preconditions

For program P, let $wp[\![P]\!] : \mathbb{P} \to \mathbb{P}$ a predicate transformer.

$G = wp[\![P]\!](F)$ is P's weakest precondition w.r.t. postcondition F iff

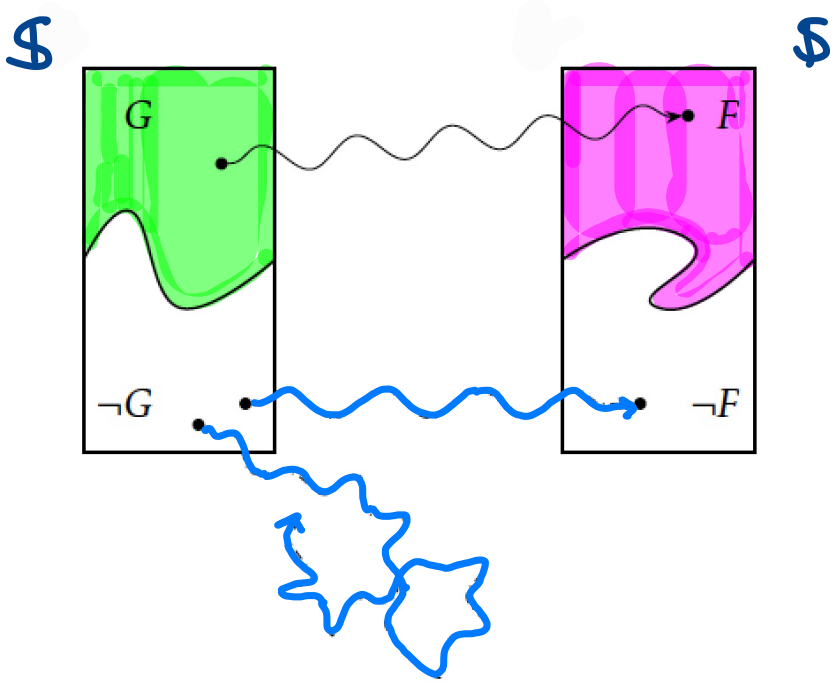
- If P starts in a state $s \vDash G$, it terminates in a state $t \vDash F$.
- Otherwise, P either terminates in a state $t \nvDash F$ or diverges

Weakest preconditions

For program P, let $wp[\![P]\!] : \mathbb{P} \to \mathbb{P}$ a predicate transformer.

$G = wp[\![P]\!](F)$ is P's weakest precondition w.r.t. postcondition F iff

- If P starts in a state $s \vDash G$, it terminates in a state $t \vDash F$.
- Otherwise, P either terminates in a state $t \nvDash F$ or diverges



Relation to Hoare triples

Hoare logic is relational

- For each $G \in \mathbb{P}$ there are many $F \in \mathbb{P}$ such that

$$\{\, G \,\}\ P\ \{\, F \,\} \qquad \text{is a valid statement}$$

- For each $F \in \mathbb{P}$ there are many $G \in \mathbb{P}$ such that

$$\{\, G \,\}\ P\ \{\, F \,\} \qquad \text{is a valid statement}$$

Relation to Hoare triples

Hoare logic is relational

- For each $G \in \mathbb{P}$ there are many $F \in \mathbb{P}$ such that

$$\{\, G \,\}\ P\ \{\, F \,\} \qquad \text{is a valid statement}$$

- For each $F \in \mathbb{P}$ there are many $G \in \mathbb{P}$ such that

$$\{\, G \,\}\ P\ \{\, F \,\} \qquad \text{is a valid statement}$$

Weakest preconditions are functional

- For each $F \in \mathbb{P}$ there is a unique $G \in \mathbb{P}$ such that

$$wp[\![P]\!](F) \;=\; G$$

Relation to Hoare triples

Hoare logic is relational

- For each $G \in \mathbb{P}$ there are many $F \in \mathbb{P}$ such that

$$\{\, G \,\}\ P\ \{\, F \,\} \qquad \text{is a valid statement}$$

- For each $F \in \mathbb{P}$ there are many $G \in \mathbb{P}$ such that

$$\{\, G \,\}\ P\ \{\, F \,\} \qquad \text{is a valid statement}$$

Weakest preconditions are functional

- For each $F \in \mathbb{P}$ there is a unique $G \in \mathbb{P}$ such that

$$wp[\![P]\!](F) \;=\; G$$

Weakest preconditions respect Hoare triples:

$$\{\, wp[\![P]\!](F) \,\}\ P\ \{\, F \,\} \qquad \text{is a valid statement}$$

Example

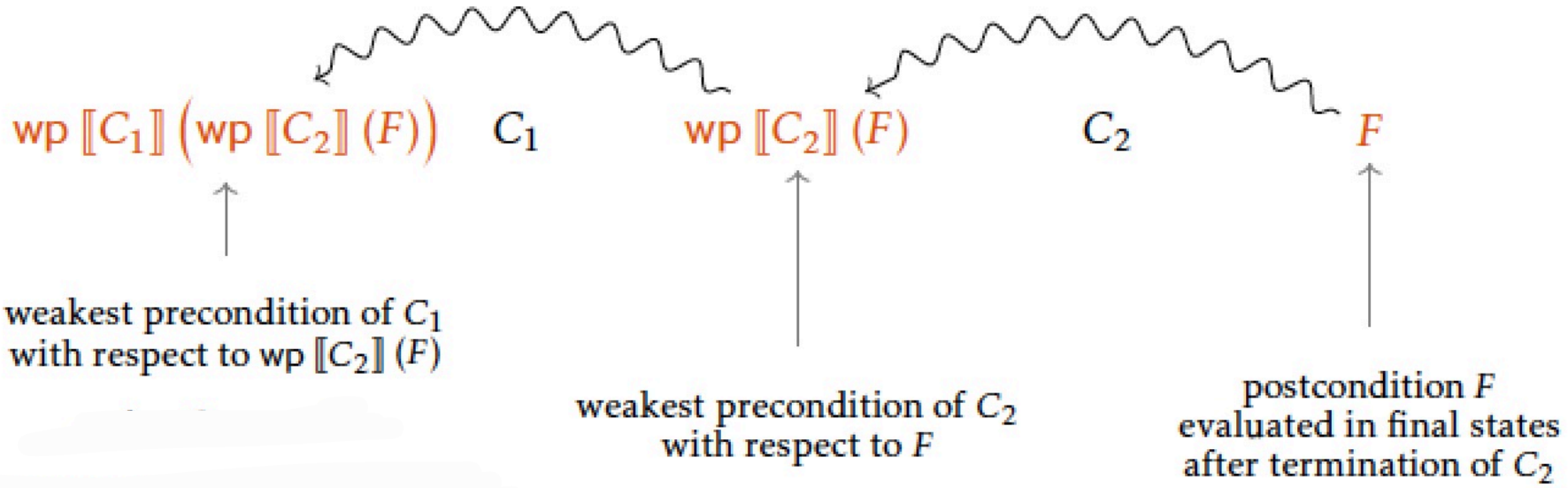
$a, b \in \mathbb{Z}$

$\{a=2 \wedge b=b\}\ a := a+2\ ;\ b := b-3\ \{a \cdot b = 12\}$

$\{a=2 \wedge b=b\}\ a := a+2\ ;\ b := b-3\ \{a=4 \vee b=a^2+1\}$

$$wp\,[\![\ a := a+2\ ;\ b := b-3\ ]\!]\,(a \cdot b = 12) = \bigvee_{a,b} (a+2)\cdot(b-3) = 12$$

Backward reasoning



Weakest preconditions reason in a backward manner about programs.

Predicate transformer semantics

Syntax program P	Semantics $wp[\![P]\!](F)$
`skip`	F
$x := E$	$F[x := E]$
$P; Q$	$wp[\![P]\!](wp[\![Q]\!](F))$

where lfp is the least fixed point wrt. the ordering $\sqsubseteq$ = $\Rightarrow$ on $\mathbb{P}$.

Predicate transformer semantics

Syntax program P	Semantics $wp[\![P]\!](F)$
`skip`	F
$x := E$	$F[x := E]$
$P; Q$	$wp[\![P]\!](wp[\![Q]\!](F))$
`if` (φ) P `else` Q	$(\varphi \wedge wp[\![P]\!](F)) \vee (\neg\varphi \wedge wp[\![Q]\!](F))$

where lfp is the least fixed point wrt. the ordering $\sqsubseteq$ = $\Rightarrow$ on $\mathbb{P}$.

Predicate transformer semantics

Syntax program P	Semantics $wp[\![P]\!](F)$
`skip`	F
$x := E$	$F[x := E]$
$P; Q$	$wp[\![P]\!](wp[\![Q]\!](F))$
`if` (φ) P `else` Q	$(\varphi \wedge wp[\![P]\!](F)) \vee (\neg\varphi \wedge wp[\![Q]\!](F))$
`while` (φ) $\{P\}$	$\mathrm{lfp}\, X.\ \underbrace{((\varphi \wedge wp[\![P]\!](X)) \vee (\neg\varphi \wedge F))}_{\text{loop characteristic function } \Phi_F(X)}$

where lfp is the least fixed point wrt. the ordering $\sqsubseteq\ = \Rightarrow$ on $\mathbb{P}$.

Unbounded loops

$$wp[\![\text{while } (\varphi)\,\{\,P\,\}\,,F]\!] \;=\; \text{lfp}\,X.\;\underbrace{((\varphi \wedge wp[\![P]\!](X)) \vee (\neg\varphi \wedge F))}_{\text{loop characteristic function } \Phi_F(X)}$$

Unbounded loops

$$wp[\![\text{while } (\varphi)\,\{\,P\,\}\,,F]\!] \;=\; \text{lfp}\, X.\; \underbrace{((\varphi \wedge wp[\![P]\!](X)) \vee (\neg\varphi \wedge F))}_{\text{loop characteristic function } \Phi_F(X)}$$

- The function $\Phi_F : \mathbb{P} \to \mathbb{P}$ is continuous on $(\mathbb{P}, \sqsubseteq)$.
- Kleene's fixpoint theorem yields: $\text{lfp}\ \Phi_F = \sup_{n \in \mathbb{N}} \Phi_F^n(\text{false})$.
- $\Phi^n(\text{false})$ denotes the wp of running the loop exactly n times starting from $\varnothing$.

wp [[P]] (true) = set of states on which P terminates

Loops in action

while $(y>0)$ $\{y--\}$ $\qquad F = y \geqslant 0$

$$wp[\![\text{loop}]\!](y=0) = \text{lfp}\, X.\ (y>0) \wedge wp[\![y--]\!](X) \vee \neg(y>0) \wedge y \geqslant 0$$

$$= \text{lfp}\, X.\ \underbrace{\left(y>0 \wedge X[y/y-1]\right) \vee y=0}_{\Phi}$$

$$\Phi^n(\text{false}) = y=0 \vee \ldots \vee y=n-1$$

$$\sup_n \Phi^n(\text{false}) = y \in \mathbb{N}_{\geqslant 0}$$

What if programs roll dice?

Overview

"Dijkstra's weakest preconditions go random"

WEAKEST PRE-EXPECTATIONS



Dexter Kozen, Annabelle McIver, and Carroll Morgan

From predicates to quantities

- Let program P be:

  `x := 5 [4/5] x := 10`

  The expected value of `x` on P's termination is: $\frac{4}{5} \cdot 5 + \frac{1}{5} \cdot 10 = 6$

From predicates to quantities

- Let program P be:

  `x := 5 [4/5] x := 10`

  The expected value of `x` on P's termination is: $\frac{4}{5}\cdot 5 + \frac{1}{5}\cdot 10 = 6$

- Let program P' be:

  `x := x+5 [4/5] x := 10`

  The expected value of `x` on P''s termination is: $\frac{4}{5}\cdot(x+5) + \frac{1}{5}\cdot 10 = \frac{4x}{5} + 6$

From predicates to quantities

- Let program P be:

  `x := 5 [4/5] x := 10`

  The expected value of `x` on P's termination is: $\frac{4}{5}\cdot 5 + \frac{1}{5}\cdot 10 = 6$

- Let program P' be:

  `x := x+5 [4/5] x := 10`

  The expected value of `x` on P''s termination is: $\frac{4}{5}\cdot(x+5) + \frac{1}{5}\cdot 10 = \frac{4x}{5} + 6$

- The probability that `x` = 10 on P''s termination is:

$$\frac{4}{5}\underbrace{\cdot[x+5 = 10]}_{\text{Iverson brackets}} + \frac{1}{5}\cdot 1 = \frac{4\cdot[x = 5] + 1}{5}$$

Expectation transformers

Let the set of expectations[1] (read: random variables):

$$\mathbb{E} = \left\{ f \mid f : \underbrace{\mathbb{S}}_{\text{states}} \to \mathbb{R}_{\geq 0} \cup \{\infty\} \right\}$$

Faggian: factors

[1] ≠ expectations in probability theory.

Expectation transformers

Let the set of expectations[1] (read: random variables):

$$\mathbb{E} = \left\{ f \mid f : \underbrace{\mathbb{S}}_{\text{states}} \to \mathbb{R}_{\geq 0} \cup \{\infty\} \right\}$$

$(\mathbb{E}, \sqsubseteq)$ is a complete lattice where $f \sqsubseteq g$ if and only if $\forall s \in \mathbb{S}.\, f(s) \leq g(s)$

[1] $\neq$ expectations in probability theory.

Expectation transformers

Let the set of expectations[1] (read: random variables):

$$\mathbb{E} = \left\{ f \mid f : \underbrace{\mathbb{S}}_{\text{states}} \to \mathbb{R}_{\geq 0} \cup \{\infty\} \right\}$$

$(\mathbb{E}, \sqsubseteq)$ is a complete lattice where $f \sqsubseteq g$ if and only if $\forall s \in \mathbb{S}.\, f(s) \leq g(s)$

Function $\Phi : \mathbb{E} \to \mathbb{E}$ is an expectation transformer

expectations are the quantitative analogue of predicates

[1] $\neq$ expectations in probability theory.

Weakest pre-expectations

For probabilistic program P, let $wp[\![P]\!] : \mathbb{E} \to \mathbb{E}$ an expectation transformer.

Weakest pre-expectations

For probabilistic program P, let $wp[\![P]\!] : \mathbb{E} \to \mathbb{E}$ an expectation transformer.

$g = wp[\![P]\!](f)$ is P's weakest pre-expectation w.r.t. post-expectation f iff

g maps initial state s to the expected value of f after executing P on s.

Weakest pre-expectations

For probabilistic program P, let $wp[\![P]\!] : \mathbb{E} \to \mathbb{E}$ an expectation transformer.

$g = wp[\![P]\!](f)$ is P's weakest pre-expectation w.r.t. post-expectation f iff

g maps initial state s to the expected value of f after executing P on s.

Examples:

- For P_1:: `x := 5 [4/5] x := 10`, we have $wp[\![P_1]\!](x) = 6$

Weakest pre-expectations

For probabilistic program P, let $wp[\![P]\!] : \mathbb{E} \to \mathbb{E}$ an expectation transformer.

$g = wp[\![P]\!](f)$ is P's weakest pre-expectation w.r.t. post-expectation f iff

g maps initial state s to the expected value of f after executing P on s.

Examples:

- For P_1:: `x := 5 [4/5] x := 10`, we have $wp[\![P_1]\!](x) = 6$
- For P_2:: `x := x+5 [4/5] x := 10`, we have:

$$wp[\![P_2]\!](x) = \frac{4x}{5} + 6 \quad \text{and} \quad wp[\![P_2]\!]([x = 10]) = \frac{4 \cdot [x = 5] + 1}{5}$$

Weakest pre-expectations

For probabilistic program P, let $wp[\![P]\!] : \mathbb{E} \to \mathbb{E}$ an expectation transformer.

$g = wp[\![P]\!](f)$ is P's weakest pre-expectation w.r.t. post-expectation f iff

g maps initial state s to the expected value of f after executing P on s.

Examples:

- For P_1:: `x := 5 [4/5] x := 10`, we have $wp[\![P_1]\!](x) = 6$
- For P_2:: `x := x+5 [4/5] x := 10`, we have:

$$wp[\![P_2]\!](x) = \frac{4x}{5} + 6 \quad \text{and} \quad wp[\![P_2]\!]([x = 10]) = \frac{4 \cdot [x = 5] + 1}{5}$$

If $f = [F]$ for predicate F, $wp[\![P]\!](f)$ is the probability of F on termination of P.

Kozen's duality theorem

If μ_P^s is the distribution over the final states obtained by running P on the initial state s, then for post-expectation f:

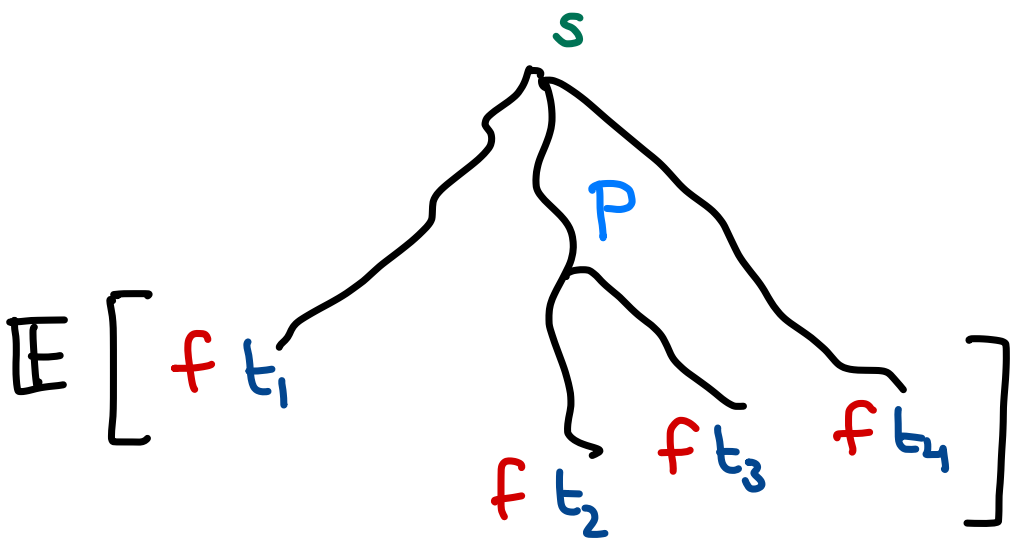
$$\underbrace{\sum_{t \in \mathbb{S}} \mu_P^s(t) \cdot f(t)}_{\text{forward}} = \underbrace{wp[\![P]\!](f)(s)}_{\text{backward}}$$

Kozen's duality theorem

If μ_P^s is the distribution over the final states obtained by running P on the initial state s, then for post-expectation f:

$$\underbrace{\sum_{t \in \mathbb{S}} \mu_P^s(t) \cdot f(t)}_{\text{forward}} \quad = \quad \underbrace{wp[\![P]\!](f)(s)}_{\text{backward}}$$

Pictorially:



Expectation transformer semantics

Syntax probabilistic program P	Semantics $wp[\![P]\!](f)$
`skip`	f
$x := E$	$f[x := E]$
$P; Q$	$wp[\![P]\!](wp[\![Q]\!](f))$

Expectation transformer semantics

Syntax probabilistic program P	Semantics $wp[\![P]\!](f)$
`skip`	f
$x := E$	$f[x := E]$
$P; Q$	$wp[\![P]\!](wp[\![Q]\!](f))$
`if` (φ) P `else` Q	$[\varphi] \cdot wp[\![P]\!](f) + [\neg\varphi] \cdot wp[\![Q]\!](f)$

Expectation transformer semantics

Syntax probabilistic program P	Semantics $wp[\![P]\!](f)$
`skip`	f
$x := E$	$f[x := E]$
$P; Q$	$wp[\![P]\!](wp[\![Q]\!](f))$
`if` (φ) P `else` Q	$[\varphi] \cdot wp[\![P]\!](f) + [\neg\varphi] \cdot wp[\![Q]\!](f)$
$P\,[p]\,Q$	$p \cdot wp[\![P]\!](f) + (1-p) \cdot wp[\![Q]\!](f)$

Expectation transformer semantics

Syntax probabilistic program P	Semantics $wp[\![P]\!](f)$
`skip`	f
$x := E$	$f[x := E]$
$P; Q$	$wp[\![P]\!](wp[\![Q]\!](f))$
`if` (φ) P `else` Q	$[\varphi] \cdot wp[\![P]\!](f) + [\neg\varphi] \cdot wp[\![Q]\!](f)$
$P\,[p]\,Q$	$p \cdot wp[\![P]\!](f) + (1-p) \cdot wp[\![Q]\!](f)$
`while` (φ) $\{P\}$	$\mathrm{lfp}\, X.\ \underbrace{(([\varphi] \cdot wp[\![P]\!](X)) + [\neg\varphi] \cdot f)}_{\text{loop characteristic function } \Psi_f(X)}$

where lfp is the least fixed point wrt. the ordering $\sqsubseteq$ on $\mathbb{E}$.

Loops in action

while $(y=1)$ $\{$ $y := 0$ $[\frac{1}{3}]$ $c := c+1$ $\}$ $\qquad f = 1$

$$\Psi_f(X) = [y=1] \cdot \left(\tfrac{1}{3} X(y \leftarrow 0) + \tfrac{2}{3} X(c \leftarrow c+1) \right) + [y \neq 1] \cdot 1$$

$$\Psi_f^n(0) = [y=1] \cdot \sum_{k=0}^{n} \tfrac{1}{3} \cdot \left(\tfrac{2}{3}\right)^k + [y \neq 1]$$

$$\sup_n \Psi_f^n(0) = [y=1] \frac{1/3}{1-2/3} + [y \neq 1] = 1$$

$$= wp[\![\text{loop}]\!](1)$$

Properties

For all programs P and expectations f, g it holds:

- Continuity: $wp[\![P]\!](\cdot)$ is continuous.
- Monotonicity: $f \sqsubseteq g$ implies $wp[\![P]\!](f) \sqsubseteq wp[\![P]\!](g)$
- Feasibility: $f \sqsubseteq \mathbf{k}$ implies $wp[\![P]\!](f) \sqsubseteq \mathbf{k}$
- Linearity: $wp[\![P]\!](r \cdot f + g) = r \cdot wp[\![P]\!](f) + wp[\![P]\!](g) \qquad \forall r \in \mathbb{R}_{\geq 0}$
- Strictness: $wp[\![P]\!](\mathbf{0}) = \mathbf{0}$

Properties

For all programs P and expectations f, g it holds:

- Continuity: $wp[\![P]\!](\cdot)$ is continuous.
- Monotonicity: $f \sqsubseteq g$ implies $wp[\![P]\!](f) \sqsubseteq wp[\![P]\!](g)$
- Feasibility: $f \sqsubseteq \mathbf{k}$ implies $wp[\![P]\!](f) \sqsubseteq \mathbf{k}$
- Linearity: $wp[\![P]\!](r \cdot f + g) = r \cdot wp[\![P]\!](f) + wp[\![P]\!](g) \quad \forall r \in \mathbb{R}_{\geq 0}$
- Strictness: $wp[\![P]\!](\mathbf{0}) = \mathbf{0}$

Good to know: $wp[\![P]\!](\mathbf{1})$ = termination probability of program P

Practical relevance

- Formal verification of randomised algorithms
- Exact inference for probabilistic programs
- Deciding program equivalence
- Proving program transformations
- Expected resource consumption
- Proving almost-sure termination

Overview

RELATIVE COMPLETENESS

SIAM J. COMPUT.
Vol. 7, No. 1, February 1978

SOUNDNESS AND COMPLETENESS OF AN AXIOM SYSTEM FOR PROGRAM VERIFICATION*

STEPHEN A. COOK†

Abstract. A simple ALGOL-like language is defined which includes conditional, while, and procedure call statements as well as blocks. A formal interpretive semantics and a Hoare style axiom system are given for the language. The axiom system is proved to be sound, and in a certain sense complete, relative to the interpretive semantics. The main new results are the completeness theorem, and a careful treatment of the procedure call rules for procedures with global variables in their declarations.

Key words. program verification, semantics, axiomatic semantics, interpretive semantics, consistency, completeness

1. Introduction. The axiomatic approach to program verification along the lines formulated by C. A. R. Hoare (see, for example, [6] and [7]) has received a great deal of attention in the last few years. My purpose here is to pick a simple programming language with a few basic features, give a Hoare style axiom system for the language, and then give a clean and careful justification for both the soundness and adequacy (i.e., completeness) of the axiom system. The justification is done by introducing an interpretive semantics for the language, rather like that in [10] and [8]. These two papers also have outlined soundness arguments for axiom systems, but for somewhat different language features, axioms, and interpretive models. The completeness claim and argument presented here is new (although completeness and incompleteness proofs inspired by an earlier version of this paper [2] appear in [3], [11], [12], [13], and [14]). I have tried to choose the axioms and rules of the formal system to be as simple as possible, subject to the constraints that they be sound, complete, and in the style and spirit of Hoare's rules.

SIAM J. on Computing, 1978

Stephen Cook

Two verification perspectives

- Extensional: use mathematical formulas as assertions
- Intensional: provide a syntax for assertions

Two verification perspectives

- Extensional: use mathematical formulas as assertions
- Intensional: provide a syntax for assertions
- Why bother about providing a syntax?
  - Enable automation (e.g., Dafny, Boogie, Viper, ...)
  - Often used (e.g., invariant templates, super martingales)
  - Enables guided search for specialised fragments
  - .....

Relative complete verification

Ordinary Programs

F

$G \Longrightarrow wp[\![P]\!](F)$

Relative complete verification

Ordinary Programs

$F \in$ FO-Arithmetic

$G \Longrightarrow wp[\![P]\!](F)$

Relative complete verification

Ordinary Programs

$F \in$ FO-Arithmetic

implies

$wp[\![P]\!](F) \in$ FO-Arithmetic

$$G \Longrightarrow wp[\![P]\!](F)$$

Relative complete verification

Ordinary Programs

$F \in$ FO-Arithmetic

implies

$wp[\![P]\!](F) \in$ FO-Arithmetic

$G \Longrightarrow wp[\![P]\!](F)$

is effectively decidable

modulo an oracle for deciding $\Rightarrow$

Relative complete verification

Ordinary Programs

$F \in$ FO-Arithmetic

implies

$wp[\![P]\!](F) \in$ FO-Arithmetic

$G \Longrightarrow wp[\![P]\!](F)$

is effectively decidable

modulo an oracle for deciding $\Rightarrow$

Probabilistic Programs

f

$g \sqsubseteq wp[\![P]\!](f)$

Relative complete verification

Ordinary Programs

$F \in$ FO-Arithmetic

implies

$wp[\![P]\!](F) \in$ FO-Arithmetic

$G \Longrightarrow wp[\![P]\!](F)$

is effectively decidable

modulo an oracle for deciding $\Rightarrow$

Probabilistic Programs

$f \in$ SomeSyntax

$g \sqsubseteq wp[\![P]\!](f)$

Relative complete verification

Ordinary Programs

$F \in$ FO-Arithmetic

implies

$wp[\![P]\!](F) \in$ FO-Arithmetic

$G \Longrightarrow wp[\![P]\!](F)$

is effectively decidable

modulo an oracle for deciding $\Rightarrow$

Probabilistic Programs

$f \in$ SomeSyntax

implies

$wp[\![P]\!](f) \in$ SomeSyntax

$g \sqsubseteq wp[\![P]\!](f)$

Relative complete verification

Ordinary Programs

$F \in$ FO-Arithmetic

implies

$wp[\![P]\!](F) \in$ FO-Arithmetic

$G \Longrightarrow wp[\![P]\!](F)$

is effectively decidable

modulo an oracle for deciding $\Rightarrow$

Probabilistic Programs

$f \in$ SomeSyntax

implies

$wp[\![P]\!](f) \in$ SomeSyntax

$g \sqsubseteq wp[\![P]\!](f)$

is effectively decidable

modulo an oracle for deciding $\sqsubseteq$

between two syntactic expectations.

Relative complete verification

Ordinary Programs

$F \in$ FO-Arithmetic

implies

$wp[\![P]\!](F) \in$ FO-Arithmetic

$G \Longrightarrow wp[\![P]\!](F)$

is effectively decidable

modulo an oracle for deciding $\Rightarrow$

Probabilistic Programs

$f \in$ SomeSyntax

implies

$wp[\![P]\!](f) \in$ SomeSyntax

$g \sqsubseteq wp[\![P]\!](f)$

is effectively decidable

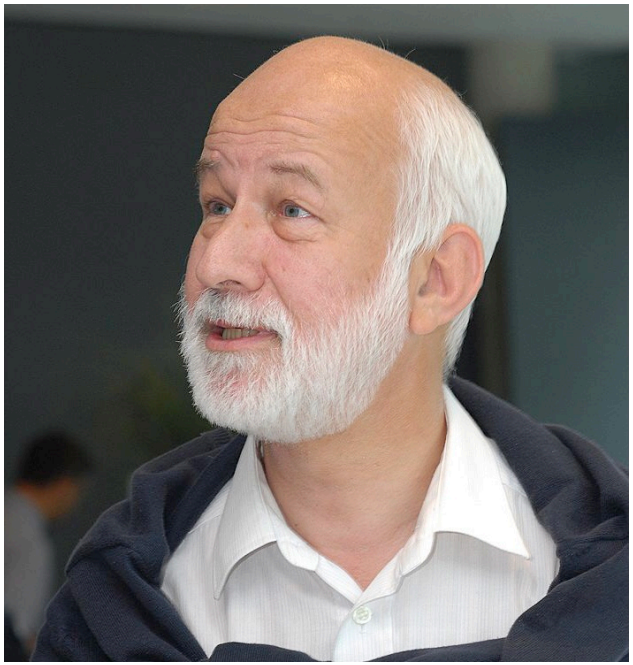
modulo an oracle for deciding $\sqsubseteq$

between two syntactic expectations.

Q: How does the SomeSyntax look like?

50 years of Hoare logic

"Completeness is a subtle manner and requires a careful analysis"



Krzysztof R. Apt

Ernst-Rüdiger Olderog

Requirements on a syntax

```
x := 1;
while (x > 0) {
 x +:= 2 [1/2] x -:= 1
}
```

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y;
t := t+1 [5/9] skip;
r := 1;
for i in 1..3 {
 s := iid(bernouilli(1/2),2t);
 if (s != t) { r := 0 }
}
```

Requirements on a syntax

```
x := 1;
while (x > 0) {
 x +:= 2 [1/2] x -:= 1
}
```

1

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y;
t := t+1 [5/9] skip;
r := 1;
for i in 1..3 {
 s := iid(bernouilli(1/2),2t);
 if (s != t) { r := 0 }
}
```

Requirements on a syntax

$\frac{\sqrt{5}-1}{2}$ (reciprocal of Golden ratio)

```
x := 1;
while (x > 0) {
 x +:= 2 [1/2] x -:= 1
}
```

1

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y;
t := t+1 [5/9] skip;
r := 1;
for i in 1..3 {
 s := iid(bernouilli(1/2),2t);
 if (s != t) { r := 0 }
}
```

Requirements on a syntax

$\frac{\sqrt{5}-1}{2}$ (reciprocal of Golden ratio)

```
x := 1;
while (x > 0) {
 x +:= 2 [1/2] x -:= 1
}
```

1

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y;
t := t+1 [5/9] skip;
r := 1;
for i in 1..3 {
 s := iid(bernouilli(1/2),2t);
 if (s != t) { r := 0 }
}
```

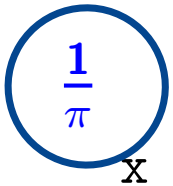
[r = 1]

Requirements on a syntax

$\frac{\sqrt{5}-1}{2}$ (reciprocal of Golden ratio)

```
x := 1;
while (x > 0) {
 x +:= 2 [1/2] x -:= 1
}
```

1




```
x := geometric(1/4);
y := geometric(1/4);
t := x+y;
t := t+1 [5/9] skip;
r := 1;
for i in 1..3 {
 s := iid(bernouilli(1/2),2t);
 if (s != t) { r := 0 }
}
```

[r = 1]

Requirements on a syntax

$\frac{\sqrt{5}-1}{2}$ (reciprocal of Golden ratio)

```
x := 1;
while (x > 0) {
 x +:= 2 [1/2] x -:= 1
}
```

$\mathbf{1}$

$\frac{\mathbf{1}}{\pi}$

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y;
t := t+1 [5/9] skip;
r := 1;
for i in 1..3 {
 s := iid(bernouilli(1/2),2t);
 if (s != t) { r := 0 }
}
```

$[\mathbf{r = 1}]$

rational numbers, algebraic numbers, transcedental numbers, etc.

Syntax: expressions

- Arithmetic expressions

$$
\begin{array}{lcll}
a & \longrightarrow & r \in \mathbb{Q}_{\geq 0} & \text{non-negative rational} \\
 & & \mid\ x \in \mathit{Vars} & \mathbb{Q}_{\geq 0}\text{-valued variable} \\
 & & \mid\ a + a & \text{addition} \\
 & & \mid\ a \cdot a & \text{multiplication} \\
 & & \mid\ a \mathbin{\dot{-}} a & \text{subtraction truncated at zero}
\end{array}
$$

Syntax: expressions

- Arithmetic expressions

$$
\begin{array}{lcll}
a & \longrightarrow & r \in \mathbb{Q}_{\geq 0} & \text{non-negative rational} \\
 & | & x \in \mathit{Vars} & \mathbb{Q}_{\geq 0}\text{-valued variable} \\
 & | & a + a & \text{addition} \\
 & | & a \cdot a & \text{multiplication} \\
 & | & a \dot{-} a & \text{subtraction truncated at zero}
\end{array}
$$

- Boolean expressions

$$
\begin{array}{lcll}
\varphi & \longrightarrow & a < a & \text{comparing arithmetic expressions} \\
 & | & \varphi \wedge \varphi & \text{conjunction} \\
 & | & \neg\varphi & \text{negation}
\end{array}
$$

Syntax: expectations

- Expectations

$$f \quad \longrightarrow \quad a \qquad \text{arithmetic expressions}$$

$$\mid \; [\varphi] \cdot f \qquad \text{guarding}$$

Syntax: expectations

- Expectations

$$
\begin{array}{lcll}
f & \longrightarrow & a & \text{arithmetic expressions} \\
 & & \mid [\varphi] \cdot f & \text{guarding} \\
 & & \mid f + f & \text{addition} \\
 & & \mid a \cdot f & \text{scaling by arithmetic expressions}
\end{array}
$$

Syntax: expectations

- Expectations

$$
\begin{array}{lcll}
f & \longrightarrow & a & \text{arithmetic expressions} \\
 & & \mid [\varphi] \cdot f & \text{guarding} \\
 & & \mid f + f & \text{addition} \\
 & & \mid a \cdot f & \text{scaling by arithmetic expressions} \\
 & & \mid \sup x \colon f & \text{supremum over variable } x \\
 & & \mid \inf x \colon f & \text{infimum over variable } x
\end{array}
$$

Syntax: expectations

- Expectations

$$
\begin{array}{lll}
f & \longrightarrow & a \qquad\qquad\qquad\qquad\qquad\qquad \text{arithmetic expressions} \\
& \mid & [\varphi] \cdot f \qquad\qquad\qquad\qquad\quad \text{guarding} \\
& \mid & f + f \qquad\qquad\qquad\qquad\quad \text{addition} \\
& \mid & a \cdot f \qquad\qquad\qquad \text{scaling by arithmetic expressions} \\
& \mid & \sqcup x{:}\, f \qquad\qquad\qquad \text{supremum over variable } x \\
& \mid & \sqcap x{:}\, f \qquad\qquad\qquad \text{infimum over variable } x
\end{array}
$$

- Examples:

$$\sqcup x{:}\, [x \cdot x < y] \cdot x$$

Syntax: expectations

- Expectations

$$
\begin{array}{lll}
f & \longrightarrow & a \qquad\qquad\qquad\qquad\qquad\qquad \text{arithmetic expressions} \\
& \mid & [\varphi] \cdot f \qquad\qquad\qquad\qquad\quad \text{guarding} \\
& \mid & f + f \qquad\qquad\qquad\qquad\quad \text{addition} \\
& \mid & a \cdot f \qquad\qquad\qquad \text{scaling by arithmetic expressions} \\
& \mid & \sqsupset x{:}\, f \qquad\qquad\quad \text{supremum over variable } x \\
& \mid & \sqsubset x{:}\, f \qquad\qquad\quad \text{infimum over variable } x
\end{array}
$$

- Examples:

$$\sqsupset x{:}[x \cdot x < y] \cdot x \;\equiv\; \sqrt{y}$$

Syntax: expectations

- Expectations

$$
\begin{array}{lll}
f \longrightarrow a & & \text{arithmetic expressions} \\
\quad \mid [\varphi] \cdot f & & \text{guarding} \\
\quad \mid f + f & & \text{addition} \\
\quad \mid a \cdot f & & \text{scaling by arithmetic expressions} \\
\quad \mid \sqsupset x{:}\, f & & \text{supremum over variable } x \\
\quad \mid \sqsubset x{:}\, f & & \text{infimum over variable } x
\end{array}
$$

- Examples:

$$\sqsupset x{:}\, [x \cdot x < y] \cdot x \equiv \sqrt{y} \qquad \sqsupset z{:}\, [z \cdot (x+1) = 1] \cdot z$$

Syntax: expectations

- Expectations

$$
\begin{array}{lll}
f & \longrightarrow & a \qquad\qquad\qquad\qquad\qquad\qquad \text{arithmetic expressions} \\
& \mid & [\varphi] \cdot f \qquad\qquad\qquad\qquad\quad \text{guarding} \\
& \mid & f + f \qquad\qquad\qquad\qquad\quad \text{addition} \\
& \mid & a \cdot f \qquad\qquad\qquad\quad \text{scaling by arithmetic expressions} \\
& \mid & \sqsupset x\colon f \qquad\qquad\qquad\quad \text{supremum over variable } x \\
& \mid & \sqsubset x\colon f \qquad\qquad\qquad\quad \text{infimum over variable } x
\end{array}
$$

- Examples:

$$\sqsupset x\colon [x \cdot x < y] \cdot x \;\equiv\; \sqrt{y} \qquad\qquad \sqsupset z\colon [z \cdot (x+1) = 1] \cdot z \;\equiv\; \frac{1}{x+1}$$

Syntax: expectations

- Expectations

$$
\begin{array}{lcll}
f & \longrightarrow & a & \text{arithmetic expressions} \\
 & & \mid [\varphi] \cdot f & \text{guarding} \\
 & & \mid f + f & \text{addition} \\
 & & \mid a \cdot f & \text{scaling by arithmetic expressions} \\
 & & \mid \sqsupset x{:}\, f & \text{supremum over variable } x \\
 & & \mid \sqsubset x{:}\, f & \text{infimum over variable } x
\end{array}
$$

- Examples:

$$\sqsupset x{:}[x \cdot x < y] \cdot x \;\equiv\; \sqrt{y} \qquad \sqsupset z{:}[z \cdot (x+1) = 1] \cdot z \;\equiv\; \frac{1}{x+1}$$

- $f \in \mathbb{E}$ is syntactic, if f is expressible in this syntax, i.e., if $f \in \mathrm{Exp}$

Semantics of syntactic expressions

Recall that state $s : \mathit{Vars} \to \mathbb{Q}_{\geq 0}$.

Semantics of syntactic expressions

Recall that state $s : \mathit{Vars} \to \mathbb{Q}_{\geq 0}$.

$$[\![a]\!]^s = [\![a]\!]^s$$

$$[\![[\varphi] \cdot f]\!]^s = \begin{cases} [\![f]\!]^s & \text{if } [\![\varphi]\!]^s = \mathit{true} \\ 0 & \text{otherwise} \end{cases}$$

$$[\![f + g]\!]^s = [\![f]\!]^s + [\![g]\!]^s$$

$$[\![a \cdot f]\!]^s = [\![a]\!]^s \cdot [\![f]\!]^s$$

Semantics of syntactic expressions

Recall that state $s : \mathit{Vars} \to \mathbb{Q}_{\geq 0}$.

$$[\![a]\!]^s = [\![a]\!]^s$$

$$[\![[\varphi] \cdot f]\!]^s = \begin{cases} [\![f]\!]^s & \text{if } [\![\varphi]\!]^s = \mathit{true} \\ 0 & \text{otherwise} \end{cases}$$

$$[\![f + g]\!]^s = [\![f]\!]^s + [\![g]\!]^s$$

$$[\![a \cdot f]\!]^s = [\![a]\!]^s \cdot [\![f]\!]^s$$

$$[\![\mathcal{Q} x \colon f]\!]^s = \sup \left\{ [\![f]\!]^{s[x \mapsto r]} \mid r \in \mathbb{Q}_{\geq 0} \right\}$$

$$[\![\mathcal{U} x \colon f]\!]^s = \inf \left\{ [\![f]\!]^{s[x \mapsto r]} \mid r \in \mathbb{Q}_{\geq 0} \right\}$$

Examples of expressible expectations

Starting from only rational-valued variable one can express:

Examples of expressible expectations

Starting from only rational-valued variable one can express:

- polynomials $y + x^3 + 2x^2 + x - 7$ widely used as templates
- rational functions $\frac{x^2 - 3x + 4}{y^2 \cdot x - 3y + 1}$
- square roots $\sqrt{x}$
- irrational, algebraic and transcendental numbers e, π, Ω

Examples of expressible expectations

Starting from only rational-valued variable one can express:

- polynomials $y + x^3 + 2x^2 + x - 7$ widely used as templates
- rational functions $\frac{x^2 - 3x + 4}{y^2 \cdot x - 3y + 1}$
- square roots $\sqrt{x}$
- irrational, algebraic and transcendental numbers e, π, Ω
- Harmonic numbers $H_k = \sum_{k=1}^{x} \frac{1}{k}$ used in run-time/termination analysis

Examples of expressible expectations

Starting from only rational-valued variable one can express:

- polynomials $y + x^3 + 2x^2 + x - 7$ widely used as templates
- rational functions $\frac{x^2 - 3x + 4}{y^2 \cdot x - 3y + 1}$
- square roots $\sqrt{x}$
- irrational, algebraic and transcendental numbers e, π, Ω
- Harmonic numbers $H_k = \sum_{k=1}^{x} \frac{1}{k}$ used in run-time/termination analysis

$$H_x = \text{Sum}\left[v_{sum}, \frac{1}{v_{sum}}, x\right] \text{ with } [\![\text{Sum}[v_{sum}, f, x]]\!]^s = \sum_{j=0}^{s(x)} [\![f[v_{sum}/j]]\!]^s$$

Expressiveness

The set Exp of syntactic expectations is expressive

if and only if

for all programs P and $f \in \text{Exp}$ it holds

$$wp[\![P]\!]([\![f]\!]) \quad = \quad [\![g]\!]$$

for some syntactic expectation $g \in \text{Exp}$.

Expressiveness

The set Exp of syntactic expectations is expressive

if and only if

for all programs P and $f \in \text{Exp}$ it holds

$$wp[\![P]\!]([\![f]\!]) \quad = \quad [\![g]\!]$$

for some syntactic expectation $g \in \text{Exp}$.

Q: is the proposed syntax expressive for loops?

A useful lemma

Any syntactic expectation $g \in \mathrm{Exp}$ can be transformed into an equivalent formula in prenex normal form:

$$Qx_1 : Qx_2 : \ldots Qx_n : f \quad \equiv \quad g$$

where $Q \in \{ \forall, \exists \}$ and f is quantifier free.

A useful lemma

Any syntactic expectation $g \in \mathrm{Exp}$ can be transformed into an equivalent formula in prenex normal form:

$$Qx_1 : Qx_2 : \ldots Qx_n : f \quad \equiv \quad g$$

where $Q \in \{ \mathcal{Q}, \mathcal{L} \}$ and f is quantifier free.

Remark: this does not hold if $f \cdot g$ is a first-class citizen in our syntax.
Note that $f \cdot g$ is expressible indirectly in our syntax

Proof strategy

By structural induction on the programs. Involved case: loops what else?

Proof strategy

By structural induction on the programs. Involved case: loops what else?

Let P = while $(\varphi)\{\, P' \,\}$

Proof strategy

By structural induction on the programs. Involved case: loops what else?

Let $P = \text{while } (\varphi)\{\, P' \,\}$

Proof obligation: $\forall f \in Exp.\ \exists g \in \text{Exp}.\quad wp[\![P]\!]([\![f]\!]) = [\![g]\!]$

given that $\forall f' \in Exp.\ \exists g' \in \text{Exp}.\quad wp[\![P']\!]([\![f']\!]) = [\![g']\!]$

Proof strategy

By structural induction on the programs. Involved case: loops what else?

Let $P = \text{while } (\varphi)\{\, P' \,\}$

Proof obligation: $\forall f \in \mathit{Exp}.\ \exists g \in \text{Exp}.\quad wp[\![P]\!]([\![f]\!]) = [\![g]\!]$

given that $\forall f' \in \mathit{Exp}.\ \exists g' \in \text{Exp}.\quad wp[\![P']\!]([\![f']\!]) = [\![g']\!]$

Step 1: exploit Kozen's duality result to ease a syntactic characterisation

$$wp[\![P]\!]([\![f]\!]) = \lambda s_0.\ \sum_{t \in \mathbb{S}} [\![\ \underbrace{[\neg\varphi] \cdot f}_{\text{strengthen } f}\ ]\!](t) \cdot \mu_P^{s_0}(t)$$

Proof strategy

$$wp[\![P]\!]([\![f]\!]) \;=\; \lambda s_0.\ \sum_{t \in \mathbb{S}} [\![[\neg\varphi] \cdot f]\!](t) \cdot \mu_P^{s_0}(t)$$

Proof strategy

$$wp[\![P]\!]([\![f]\!]) \;=\; \lambda s_0.\, \sum_{t \in \mathbb{S}} [\![[\neg\varphi] \cdot f]\!](t) \cdot \mu_P^{s_0}(t)$$

Step 2: sum over all terminating paths $s_0 \ldots s_{k-1}$ of intermediate states

$$wp[\![P]\!]([\![f]\!]) \;=\; \lambda s_0.\, \sup_{k \in \mathbb{N}} \sum_{s_0,\ldots,s_{k-1} \in \mathbb{S}} [\![[\neg\varphi] \cdot f]\!](s_{k-1}) \cdot \prod_{k=0}^{k-2} \mu_{P_{iter}}^{s_i}(s_{i+1})$$

Proof strategy

$$wp[\![P]\!]([\![f]\!]) \;=\; \lambda s_0.\, \sum_{t \in \mathbb{S}} [\![[\neg\varphi] \cdot f]\!](t) \cdot \mu_P^{s_0}(t)$$

Step 2: sum over all terminating paths $s_0 \ldots s_{k-1}$ of intermediate states

$$wp[\![P]\!]([\![f]\!]) \;=\; \lambda s_0.\, \sup_{k \in \mathbb{N}} \sum_{s_0,\ldots,s_{k-1} \in \mathbb{S}} [\![[\neg\varphi] \cdot f]\!](s_{k-1}) \cdot \prod_{k=0}^{k-2} \mu_{P_{iter}}^{s_i}(s_{i+1})$$

where program P_{iter} = if $(\varphi)\{\, P' \,\}$ else $\{$ skip $\}$ is a single loop execution

↑ loop body

Proof strategy

$$wp[\![P]\!]([\![f]\!]) \;=\; \lambda s_0.\, \sum_{t \in \mathbb{S}} [\![[\neg\varphi] \cdot f]\!](t) \cdot \mu_P^{s_0}(t)$$

Step 2: sum over all terminating paths $s_0 \ldots s_{k-1}$ of intermediate states

$$wp[\![P]\!]([\![f]\!]) \;=\; \lambda s_0.\, \sup_{k \in \mathbb{N}} \sum_{s_0,\ldots,s_{k-1} \in \mathbb{S}} [\![[\neg\varphi] \cdot f]\!](s_{k-1}) \cdot \prod_{k=0}^{k-2} \mu_{P_{iter}}^{s_i}(s_{i+1})$$

where program P_{iter} = if $(\varphi)\{\, P' \,\}$ else $\{$ skip $\}$ is a single loop execution

Step 3: characterise $\mu_{P_{iter}}^{s_i}(s_{i+1})$ with wp. Use the characteristic assertion

$$[s] \;=\; \left[\bigwedge_{x \in Vars} x = s(x) \right]$$

Proof strategy

$$wp[\![P]\!]([\![f]\!]) \;=\; \lambda s_0.\, \sum_{t \in \mathbb{S}} [\![[\neg\varphi] \cdot f]\!](t) \cdot \mu_P^{s_0}(t)$$

Step 2: sum over all terminating paths $s_0 \ldots s_{k-1}$ of intermediate states

$$wp[\![P]\!]([\![f]\!]) \;=\; \lambda s_0.\, \sup_{k \in \mathbb{N}} \sum_{s_0,\ldots,s_{k-1} \in \mathbb{S}} [\![[\neg\varphi] \cdot f]\!](s_{k-1}) \cdot \prod_{k=0}^{k-2} \mu^{s_i}_{P_{iter}}(\sigma_{i+1})$$

where program P_{iter} = if $(\varphi)\{\, P' \,\}$ else $\{$ skip $\}$ is a single loop execution

Step 3: characterise $\mu^{s_i}_{P_{iter}}(\, s_{i+1} \,)$ with wp. Use the characteristic assertion

$$[s] \;=\; \left[\bigwedge_{x \in Vars} x = s(x) \right]$$

By Kozen's duality theorem

$$\mu^{s_i}_{P_{iter}}(\, s_{i+1} \,) \quad = \quad wp[\![P_{iter}]\!]([\![[s_{i+1}]]\!])(s_i)$$

Proof strategy

$$wp[\![P]\!]([\![f]\!]) \;=\; \lambda s_0.\, \sum_{t \in \mathbb{S}} [\![[\neg\varphi] \cdot f]\!](t) \cdot \mu_P^{s_0}(t)$$

Step 2: sum over all terminating paths $s_0 \ldots s_{k-1}$ of intermediate states

$$wp[\![P]\!]([\![f]\!]) \;=\; \lambda s_0.\, \sup_{k \in \mathbb{N}} \sum_{s_0,\ldots,s_{k-1} \in \mathbb{S}} [\![[\neg\varphi] \cdot f]\!](s_{k-1}) \cdot \prod_{k=0}^{k-2} \mu_{P_{iter}}^{s_i}(\sigma_{i+1})$$

where program $P_{iter} \;=\;$ if $(\varphi)\{\, P' \,\}$ else $\{$ skip $\}$ is a single loop execution

Step 3: characterise $\mu_{P_{iter}}^{s_i}(s_{i+1})$ with wp. Use the characteristic assertion

$$[s] \;=\; \left[\bigwedge_{x \in Vars} x = s(x) \right]$$

By Kozen's duality theorem and the induction hypothesis:

$$\mu_{P_{iter}}^{s_i}(s_{i+1}) \;=\; wp[\![P_{iter}]\!]([\![[s_{i+1}]]\!])(s_i) \overset{\text{I.H.}}{=} [\![\overbrace{g_{P_{iter}}^{s_{i+1}}}^{\text{in Exp}}]\!](s_i)$$

Wrapping it up

It follows that:

$$wp[\![\text{while } (\varphi)\{P'\}]\!](f) =$$

$$\lambda s_0.\ \mathfrak{Z}\, k\colon \sum_{s_0,\ldots,s_{k-1}} ([\neg\varphi]\cdot f)(s_{k-1}) \cdot \prod_{i=0}^{k-2} wp[\![\underbrace{\text{if } (\varphi)\{P'\} \text{ else } \{\text{skip}\}}_{\text{program } P_{iter}}]\!]([s_{i+1}])(s_i)$$

Wrapping it up

It follows that:

$$wp[\![\text{while } (\varphi)\{P'\}]\!](f) =$$

$$\lambda s_0.\ \mathfrak{D}\, k\colon \sum_{s_0,\ldots,s_{k-1}} ([\neg\varphi]\cdot f)(s_{k-1}) \cdot \prod_{i=0}^{k-2} wp[\![\underbrace{\text{if } (\varphi)\{P'\} \text{ else } \{\text{skip}\}}_{\text{program } P_{iter}}]\!]([s_{i+1}])(s_i)$$

Technical challenges in the remaining proof:

- Encode sequences of states via Gödelisation as natural numbers
- Encode sequences of rationals via Gödelisation as natural numbers

Wrapping it up

It follows that:

$$wp[\![\text{while } (\varphi)\{P'\}]\!](f) =$$

$$\lambda s_0.\ \mathfrak{D}\, k\colon \sum_{s_0,\ldots,s_{k-1}} ([\neg\varphi]\cdot f)(s_{k-1}) \cdot \prod_{i=0}^{k-2} wp[\![\underbrace{\text{if } (\varphi)\,\{P'\}\ \text{else}\ \{\text{skip}\}}_{\text{program } P_{iter}}]\!]([s_{i+1}])(s_i)$$

Technical challenges in the remaining proof:

- Encode sequences of states via Gödelisation as natural numbers
- Encode sequences of rationals via Gödelisation as natural numbers
- Express sums with variable # summands in Exp (using Sum, see H_x)
- Express products with variable # factors in Exp (similarly)
- General products of two syntactic expectations

Wrapping it up

It follows that:

$$wp[\![\text{while } (\varphi)\{P'\}]\!](f) =$$

$$\lambda s_0.\ \mathcal{Q}\, k\colon \sum_{s_0,\ldots,s_{k-1}} ([\neg\varphi]\cdot f)(s_{k-1}) \cdot \prod_{i=0}^{k-2} wp[\![\underbrace{\text{if } (\varphi)\{P'\} \text{ else } \{\text{skip}\}}_{\text{program } P_{iter}}]\!]([s_{i+1}])(s_i)$$

Technical challenges in the remaining proof:

- Encode sequences of states via Gödelisation as natural numbers
- Encode sequences of rationals via Gödelisation as natural numbers
- Express sums with variable # summands in Exp (using Sum, see H_x)
- Express products with variable # factors in Exp (similarly)
- General products of two syntactic expectations

For details, see our POPL 2021 paper.

The expressive syntax in action

$/\!/\!/ \; \partial\, len\, \partial\, nums : \mathsf{Sum}\,[\ldots, [\mathsf{StateSeq}_x(\ldots, len)] \odot \mathsf{Path}[x](len, \ldots), nums]$ (constructive)

$= \; \boldsymbol{x + [c > 0] \cdot 2}$ (simpler syntax, equivalent expectation)

$$
\begin{array}{l}
\texttt{while}\,(c > 0)\,\{ \\
\quad \{\, c := 0 \,\}\, [1/2]\, \{\, c := 1 \,\}\,; \\
\quad x := x + 1 \\
\}
\end{array}
$$

$/\!/\!/ \; \boldsymbol{x}$

Relevance

- Relative completeness à la Cook:

bounds like $g \sqsubseteq wp[\![P]\!](f)$ are effectively decidable

[2] Deciding whether P terminates on s with *at most* probability α is Π^0_2-complete.

Relevance

- Relative completeness à la Cook:

  bounds like $g \sqsubseteq wp[\![P]\!](f)$ are effectively decidable

  algebraic, transcedental numbers via Dedekind cuts
  for $\alpha \in \mathbb{R}^{\infty}_{\geqslant 0}$: $Cut(\alpha) = \{ r \in \mathbb{Q}_{>0} \mid r < \alpha \}$

- Termination probabilities $wp[\![P]\!](\mathbf{1})$ on any input are expressible[2]

- Probability to terminate in postcondition φ as $wp[\![P]\!]([\varphi])$

- Distribution over final states where $t(x_i) = v_i$:

$$\mu_P^s(t) \quad = \quad wp[\![P]\!]\left([x_1 = v_1 \wedge \cdots \wedge x_k = v_k]\right)$$

Thus Kozen's measure transformers can be syntactically expressed

[2] Deciding whether P terminates on s with *at most* probability α is Π^0_2-complete.

Overview

Epilogue

Summary

- Weakest precondition reasoning about probabilistic programs
- $g \sqsubseteq wp[\![P]\!](f)$ is effectively decidable (modulo an oracle for $\sqsubseteq$)
- A suitable syntax for specifying quantitative "assertions"

Epilogue

Summary

- Weakest precondition reasoning about probabilistic programs
- $g \sqsubseteq wp[\![P]\!](f)$ is effectively decidable (modulo an oracle for $\sqsubseteq$)
- A suitable syntax for specifying quantitative "assertions"

Future work

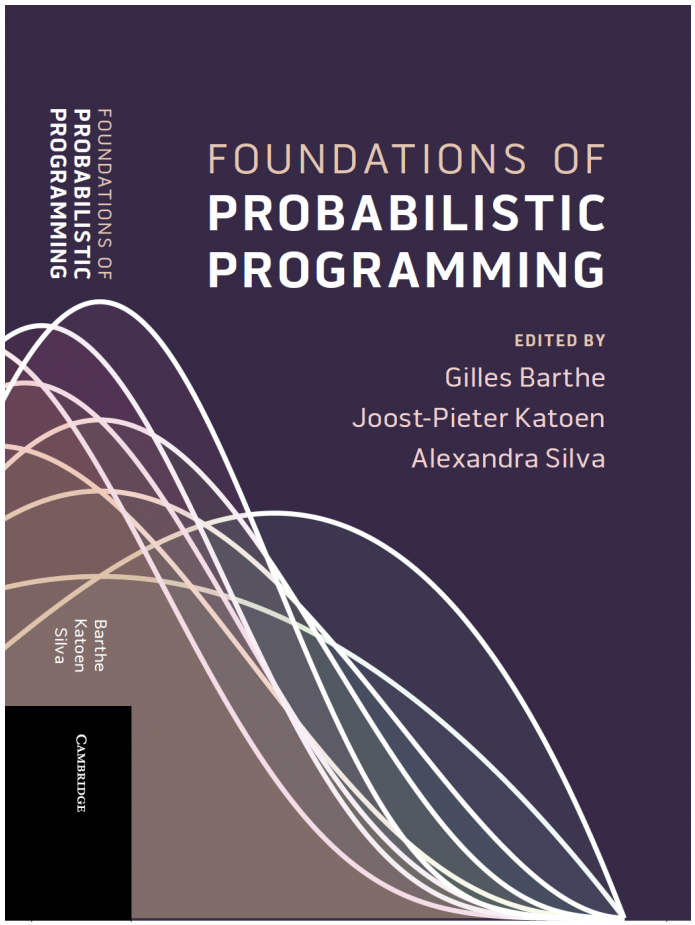
- Expressiveness of extensions (cf. next slide)
- Automation

Extensions of wp-reasoning

- . . . . . for inference (treats divergence) [TOPLAS 2018]
- . . . . . for continuous distributions w. scoring + weighting [SETTS 2019]
- . . . . . for recursion [LICS 2016]
- . . . . . for expected run-time analysis [J.ACM 2018]
- . . . . . for probabilistic separation logic [POPL 2019]
- . . . . . for weighted programs [OOPSLA 2022]

Q: which syntax suffices for these extensions?

Foundations book: open access



- semantics
- termination analysis
- concentration bounds
- algebraic reasoning
- abstract interpretation
- expected run-time analysis
- Kuifje
- Bayesian inference
- probabilistic couplings
- .....