universität innsbruck

Abstraction, Refinement and Proof in a Probabilistic Setting

Master Seminar WS20/21

Jonas Schöpf

November 25, 2020 & December 1, 2020

Motivation

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

Motivation

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

Motivation

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

Motivation

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

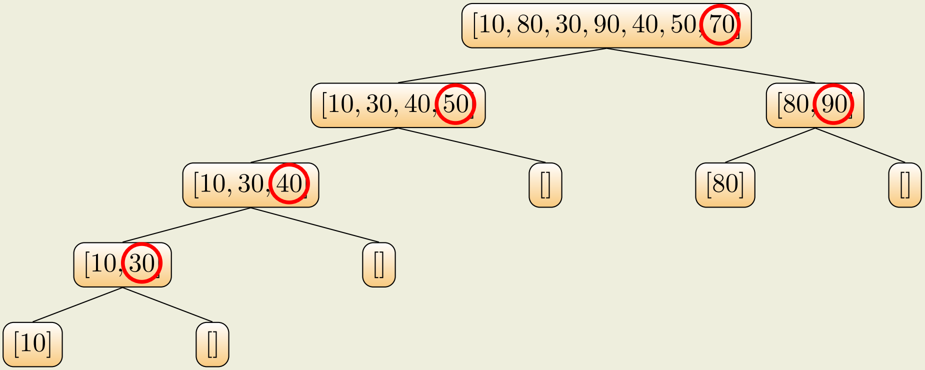
Motivation - Quicksort

- "standard" vs. randomized quicksort

Motivation - Quicksort

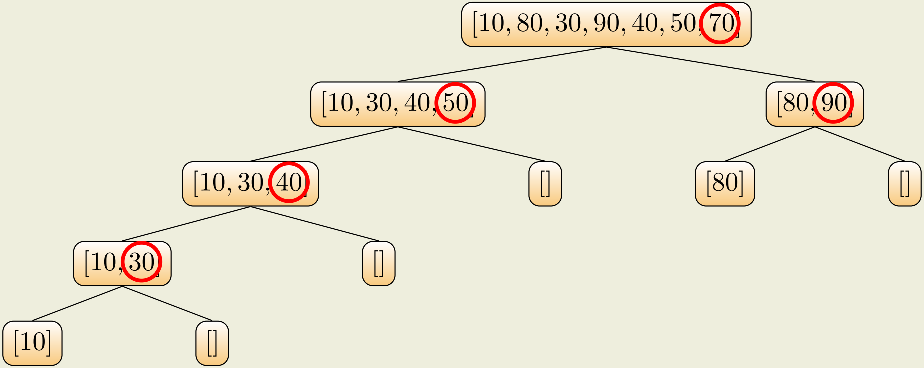
- "standard" vs. randomized quicksort

Example Quicksort



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

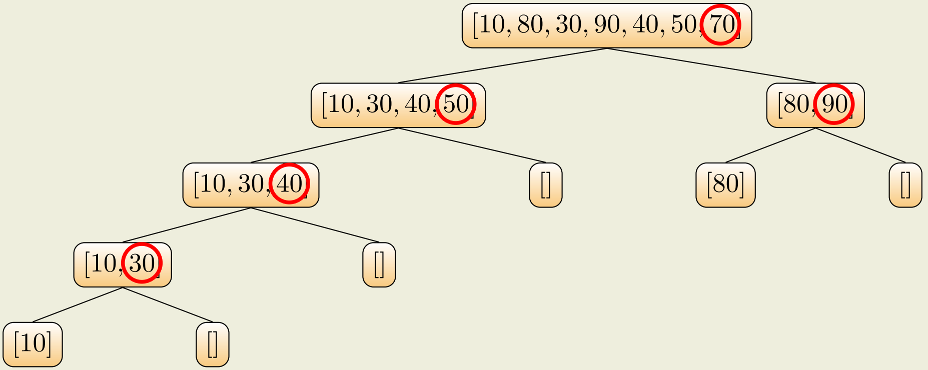
Example Quicksort



Motivation - Quicksort

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

Example Quicksort



Overview

- Guarded Command Language (GCL)
- Probabilistic Guarded Command Language (pGCL)
- Abstraction and Refinement
- Probably Hoare? Hoare Probably!
- Abstraction Refinement and Proof for Probabilistic Systems

Overview

- Guarded Command Language (GCL)
- Probabilistic Guarded Command Language (pGCL)
- Abstraction and Refinement
- Probably Hoare? Hoare Probably!
- Abstraction Refinement and Proof for Probabilistic Systems

Guarded Command Language (GCL)

- simple ($\Rightarrow$ simplicity in reasoning helps)
- "statement list prefixed by a boolean expression"

Guarded Command Language (GCL)

- simple ($\Rightarrow$ simplicity in reasoning helps)
- "statement list prefixed by a boolean expression"
- alternative construct & repetitive construct

Guarded Command Language (GCL)

- simple ($\Rightarrow$ simplicity in reasoning helps)
- "statement list prefixed by a boolean expression"
- alternative construct & repetitive construct
- used for weakest-pre-condition semantics

Guarded Command Language (GCL)

- simple ($\Rightarrow$ simplicity in reasoning helps)
- "statement list prefixed by a boolean expression"
- alternative construct & repetitive construct
- used for weakest-pre-condition semantics

Syntax of GCL

$$
\begin{aligned}
\langle \text{guarded command} \rangle &::= \langle \text{guard} \rangle \rightarrow \langle \text{guarded list} \rangle \\
\langle \text{guard} \rangle &::= \langle \text{boolean expression} \rangle \\
\langle \text{guarded list} \rangle &::= \langle \text{statement} \rangle \{ \mathbf{;} \langle \text{statement} \rangle \} \\
\langle \text{guarded command set} \rangle &::= \langle \text{guarded command} \rangle \{ \Box \langle \text{guarded command} \rangle \}
\end{aligned}
$$

Guarded Command Language (GCL)

- simple ($\Rightarrow$ simplicity in reasoning helps)
- "statement list prefixed by a boolean expression"
- alternative construct & repetitive construct
- used for weakest-pre-condition semantics

Syntax of GCL

$$
\begin{aligned}
\langle\text{guarded command}\rangle &::= \langle\text{guard}\rangle \rightarrow \langle\text{guarded list}\rangle \\
\langle\text{guard}\rangle &::= \langle\text{boolean expression}\rangle \\
\langle\text{guarded list}\rangle &::= \langle\text{statement}\rangle\{\mathbf{;}\langle\text{statement}\rangle\} \\
\langle\text{guarded command set}\rangle &::= \langle\text{guarded command}\rangle\{\square\langle\text{guarded command}\rangle\} \\
\langle\text{statement}\rangle &::= \langle\text{alternative construct}\rangle \mid \langle\text{repetitive construct}\rangle \mid \text{``other statements''}
\end{aligned}
$$

Guarded Command Language (GCL)

- simple ($\Rightarrow$ simplicity in reasoning helps)
- "statement list prefixed by a boolean expression"
- alternative construct & repetitive construct
- used for weakest-pre-condition semantics

Syntax of GCL

$$
\begin{aligned}
\langle\text{guarded command}\rangle &::= \langle\text{guard}\rangle \rightarrow \langle\text{guarded list}\rangle \\
\langle\text{guard}\rangle &::= \langle\text{boolean expression}\rangle \\
\langle\text{guarded list}\rangle &::= \langle\text{statement}\rangle\{\mathbf{;}\langle\text{statement}\rangle\} \\
\langle\text{guarded command set}\rangle &::= \langle\text{guarded command}\rangle\{\square\langle\text{guarded command}\rangle\} \\
\langle\text{alternative construct}\rangle &::= \mathbf{if}\langle\text{guarded command set}\rangle\mathbf{fi} \\
\langle\text{repetitive construct}\rangle &::= \mathbf{do}\langle\text{guarded command set}\rangle\mathbf{od} \\
\langle\text{statement}\rangle &::= \langle\text{alternative construct}\rangle \mid \langle\text{repetitive construct}\rangle \mid \text{“other statements”}
\end{aligned}
$$

Alternative Construct

$$\begin{aligned}&\textbf{if } x \geq y \rightarrow m := x\\&\square\ y \geq x \rightarrow m := y\\&\textbf{fi}\end{aligned}$$

Alternative Construct (Nondeterminism)

$$\begin{array}{l} \textbf{if } x \geq y \rightarrow m := x \\ \Box \ y \geq x \rightarrow m := y \\ \textbf{fi} \end{array}$$

Alternative Construct (Nondeterminism)

$$
\begin{array}{l}
\textbf{if } x \geq y \to m := x \\
\Box \; y \geq x \to m := y \\
\textbf{fi}
\end{array}
$$

Repetitive Construct

$$
\begin{array}{ll}
k := 0; j := 1; & \\
\textbf{do } j \neq n \to & \textbf{if } \mathsf{f}(j) \leq \mathsf{f}(k) \to j := j + 1 \\
 & \Box \; \mathsf{f}(j) \geq \mathsf{f}(k) \to k := j; j := j + 1 \\
 & \textbf{fi} \\
\textbf{od} &
\end{array}
$$

Overview

- Guarded Command Language (GCL)
- Probabilistic Guarded Command Language (pGCL)
- Abstraction and Refinement
- Probably Hoare? Hoare Probably!
- Abstraction Refinement and Proof for Probabilistic Systems

Primer: Nondeterminism vs. Determinism

"the simplicity and elegance of the above would have been destroyed by requiring the derivation of deterministic programs only" – E.W.Dijkstra in [1]

Primer: Nondeterminism vs. Determinism

"the simplicity and elegance of the above would have been destroyed by requiring the derivation of deterministic programs only" – E.W.Dijkstra in [1]

Nondeterminism Example NE

$$
\begin{array}{l}
\textbf{if } x \geq y \rightarrow m := x \\
\Box\ y \geq x \rightarrow m := y \\
\textbf{fi}
\end{array}
$$

Primer: Nondeterminism vs. Determinism

"the simplicity and elegance of the above would have been destroyed by requiring the derivation of deterministic programs only" – E.W.Dijkstra in [1]

Nondeterminism Example NE

$$
\begin{aligned}
&\textbf{if } x \geq y \rightarrow m := x \\
&\square\ y \geq x \rightarrow m := y \\
&\textbf{fi}
\end{aligned}
$$

Determinism Example DE

$$
\begin{aligned}
&\textbf{if } x > y \rightarrow m := x \\
&\square\ y < x \rightarrow m := y \\
&\square\ y = x \rightarrow m := y \\
&\textbf{fi}
\end{aligned}
$$

Primer cont'd

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

Primer cont'd

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

Formalized — into logic — it looks as:

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

Primer cont'd

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

Formalized — into logic — it looks as:

	$\{pre\}\ prog\ \{post\}$	Hoare-style
or	$pre \Rrightarrow \textit{wp}.prog.post$	Dijkstra-style

Example

$$\{x = y\}\ \textsf{NE}\ \{m = y\}$$

or $(x = y) \Rrightarrow \textit{wp}.\textsf{NE}.(m = y)$ later $\Rrightarrow$ ..."is no more than"

Primer cont'd

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

Formalized — into logic — it looks as:

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

Example

	$\{x = y\}$ NE $\{m = y\}$	
or	$(x = y) \Rrightarrow \mathit{wp}.\mathsf{NE}.(m = y)$	later $\Rrightarrow$ ..."is no more than"

- reasoning about weakest pre-conditions of programs ⇒ weakest pre-condition semantics

Primer cont'd

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

Formalized — into logic — it looks as:

	$\{pre\}\ prog\ \{post\}$	Hoare-style
or	$pre \Rrightarrow \textit{wp}.prog.post$	Dijkstra-style

Example

$$\{x = y\}\ \text{NE}\ \{m = y\}$$

or $(x = y) \Rrightarrow \textit{wp}.\text{NE}.(m = y)$ later $\Rrightarrow$ ..."is no more than"

- reasoning about weakest pre-conditions of programs ⇒ weakest pre-condition semantics
- Hoare logic = formal system (set of logical rules) for reasoning about the correctness of programs

How to Use GCL in a Probabilistic Setting?

- *deterministic* vs. *nondeterministic* vs. *probabilistic* choice
- 'demonic' choice in GCL by Dijkstra (first overlapping guards)

How to Use GCL in a Probabilistic Setting?

- *deterministic* vs. *nondeterministic* vs. *probabilistic* choice
- 'demonic' choice in GCL by Dijkstra (first overlapping guards)

Demonic Choice

- first not fundamental $\Rightarrow$ abandoned

How to Use GCL in a Probabilistic Setting?

- *deterministic* vs. *nondeterministic* vs. *probabilistic* choice
- 'demonic' choice in GCL by Dijkstra (first overlapping guards)

Demonic Choice

- first not fundamental $\Rightarrow$ abandoned
- replaced by probabilistic choice

How to Use GCL in a Probabilistic Setting?

- *deterministic* vs. *nondeterministic* vs. *probabilistic* choice
- 'demonic' choice in GCL by Dijkstra (first overlapping guards)

Demonic Choice

- first not fundamental $\Rightarrow$ abandoned
- replaced by probabilistic choice
- probabilistic semantics divorced

How to Use GCL in a Probabilistic Setting?

- *deterministic* vs. *nondeterministic* vs. *probabilistic* choice
- 'demonic' choice in GCL by Dijkstra (first overlapping guards)

Demonic Choice

- first not fundamental $\Rightarrow$ abandoned
- replaced by probabilistic choice
- probabilistic semantics divorced
- deterministic **refines** probabilistic choice, which **refines** demonic choice

How to Use GCL in a Probabilistic Setting?

- *deterministic* vs. *nondeterministic* vs. *probabilistic* choice
- 'demonic' choice in GCL by Dijkstra (first overlapping guards)

Demonic Choice

- first not fundamental $\Rightarrow$ abandoned
- replaced by probabilistic choice
- probabilistic semantics divorced
- deterministic **refines** probabilistic choice, which **refines** demonic choice

Demonic Choice Operator

$$\mathit{this} \sqcap \mathit{that}$$

Basically means, that it does not matter if we choose *this* or *that*.

Probabilistic Guarded Command Language (pGCL)

⇒ extension of GCL to incorporate probabilities **&** demonic choice

Probabilistic Guarded Command Language (pGCL)

- $\Rightarrow$ extension of GCL to incorporate probabilities **&** demonic choice
- $\Rightarrow$ acts over *expectations* rather than *predicates*; an expectation is real
  special case: $[P]$ is probability that predicate P holds, so if false, then $[P] = 0$, if true $[P] = 1$

Probabilistic Guarded Command Language (pGCL)

- ⇒ extension of GCL to incorporate probabilities **&** demonic choice
- ⇒ acts over *expectations* rather than *predicates*; an expectation is real
  special case: $[P]$ is probability that predicate P holds, so if false, then $[P] = 0$, if true $[P] = 1$

(Part of the) Syntax of pGCL

$$\langle \text{prog} \rangle := \quad \textbf{abort} \mid \textbf{skip} \mid x := E \mid \langle \text{prog} \rangle ; \langle \text{prog} \rangle$$

Probabilistic Guarded Command Language (pGCL)

- ⇒ extension of GCL to incorporate probabilities **&** demonic choice
- ⇒ acts over *expectations* rather than *predicates*; an expectation is real
  special case: $[P]$ is probability that predicate P holds, so if false, then $[P] = 0$, if true $[P] = 1$

(Part of the) Syntax of pGCL

$$\begin{aligned}\langle\text{prog}\rangle := \quad & \textbf{abort} \mid \textbf{skip} \mid x := E \mid \langle\text{prog}\rangle ; \langle\text{prog}\rangle \\ & \langle\text{prog}\rangle \;{}_p\!\oplus \langle\text{prog}\rangle \mid \langle\text{prog}\rangle \sqcap \langle\text{prog}\rangle \mid\end{aligned}$$

Probabilistic Guarded Command Language (pGCL)

- $\Rightarrow$ extension of GCL to incorporate probabilities **&** demonic choice
- $\Rightarrow$ acts over *expectations* rather than *predicates*; an expectation is real
  special case: $[P]$ is probability that predicate P holds, so if false, then $[P] = 0$, if true $[P] = 1$

(Part of the) Syntax of pGCL

$$\begin{aligned} \langle \text{prog} \rangle := \quad & \textbf{abort} \mid \textbf{skip} \mid x := E \mid \langle \text{prog} \rangle ; \langle \text{prog} \rangle \\ & \langle \text{prog} \rangle \ {}_p\!\oplus \langle \text{prog} \rangle \mid \langle \text{prog} \rangle \sqcap \langle \text{prog} \rangle \mid \\ & (\textbf{mu} \ xxx \cdot \mathcal{C}) \end{aligned}$$

Probabilistic Guarded Command Language (pGCL)

- ⇒ extension of GCL to incorporate probabilities **&** demonic choice
- ⇒ acts over *expectations* rather than *predicates*; an expectation is real special case: $[P]$ is probability that predicate P holds, so if false, then $[P] = 0$, if true $[P] = 1$

(Part of the) Syntax of pGCL

$$\langle\text{prog}\rangle := \quad \textbf{abort} \mid \textbf{skip} \mid x := E \mid \langle\text{prog}\rangle; \langle\text{prog}\rangle$$
$$\langle\text{prog}\rangle \;{}_p\!\oplus \langle\text{prog}\rangle \mid \langle\text{prog}\rangle \sqcap \langle\text{prog}\rangle \mid$$
$$(\textbf{mu}\ xxx \cdot \mathcal{C})$$

Probabilistic Choice Operator: Coin Flip

Tail ${}_{\frac{1}{2}}\!\oplus$ *Head* . . . fair coin

Probabilistic Guarded Command Language (pGCL)

- ⇒ extension of GCL to incorporate probabilities **&** demonic choice
- ⇒ acts over *expectations* rather than *predicates*; an expectation is real
  special case: $[P]$ is probability that predicate P holds, so if false, then $[P] = 0$, if true $[P] = 1$

(Part of the) Syntax of pGCL

$$\begin{aligned}\langle\text{prog}\rangle := \quad & \textbf{abort} \mid \textbf{skip} \mid x := E \mid \langle\text{prog}\rangle ; \langle\text{prog}\rangle \\ & \langle\text{prog}\rangle \;{}_p\!\oplus \langle\text{prog}\rangle \mid \langle\text{prog}\rangle \sqcap \langle\text{prog}\rangle \mid \\ & (\textbf{mu}\ xxx \cdot \mathcal{C})\end{aligned}$$

Probabilistic Choice Operator: Coin Flip

$$\textit{Tail} \;{}_{\frac{1}{2}}\!\oplus \textit{Head} \qquad \ldots \text{fair coin}$$

no perfect coins in nature:

Probabilistic Guarded Command Language (pGCL)

- ⇒ extension of GCL to incorporate probabilities **&** demonic choice
- ⇒ acts over *expectations* rather than *predicates*; an expectation is real
  special case: $[P]$ is probability that predicate P holds, so if false, then $[P] = 0$, if true $[P] = 1$

(Part of the) Syntax of pGCL

$$\begin{aligned} \langle \text{prog} \rangle := \quad & \textbf{abort} \mid \textbf{skip} \mid x := E \mid \langle \text{prog} \rangle ; \langle \text{prog} \rangle \\ & \langle \text{prog} \rangle \ {}_p\!\oplus \langle \text{prog} \rangle \mid \langle \text{prog} \rangle \sqcap \langle \text{prog} \rangle \mid \\ & (\textbf{mu}\ xxx \cdot \mathcal{C}) \end{aligned}$$

Probabilistic Choice Operator: Coin Flip

$$\textit{Tail}\ {}_{\frac{1}{2}}\!\oplus \textit{Head} \qquad \ldots \text{fair coin}$$

no perfect coins in nature:

$$\textit{Tail}\ {}_{0.49}\!\oplus \textit{Head} \sqcap \textit{Tail}\ {}_{0.51}\!\oplus \textit{Head} \qquad \ldots \text{nearly fair coin}$$

pGCL cont'd

There exist more constructs such as:

- Boolean embedding of predicate *pred* as expectation:

$$[\mathit{pred}] := \text{“}\textbf{if } \mathit{pred} \textbf{ then } 1 \textbf{ else } 0\text{”}$$

pGCL cont'd

There exist more constructs such as:

- Boolean embedding of predicate *pred* as expectation:

$$[\mathit{pred}] := \text{“}\mathbf{if}\ \mathit{pred}\ \mathbf{then}\ 1\ \mathbf{else}\ 0\text{”}$$

- Conditional:

$$\mathbf{if}\ \mathit{pred}\ \mathbf{then}\ \mathit{prog}\ \mathbf{else}\ \mathit{prog'}\ \mathbf{fi} := \mathit{prog}\ {}_{[\mathit{pred}]}\!\oplus\ \mathit{prog'}$$

pGCL cont'd

There exist more constructs such as:

- Boolean embedding of predicate *pred* as expectation:

$$[\textit{pred}] := \text{“}\textbf{if } \textit{pred} \textbf{ then } 1 \textbf{ else } 0\text{”}$$

- Conditional:

$$\textbf{if } \textit{pred} \textbf{ then } \textit{prog} \textbf{ else } \textit{prog'} \textbf{ fi} := \textit{prog} \; {}_{[\textit{pred}]}\!\oplus \; \textit{prog'}$$

- Multi-way probabilistic choices
- Variations on ${}_p\oplus$
- Demonic choice in variable assignments

pGCL cont'd

There exist more constructs such as:

- Boolean embedding of predicate *pred* as expectation:

$$[\textit{pred}] := \text{``}\textbf{if}\ \textit{pred}\ \textbf{then}\ 1\ \textbf{else}\ 0\text{''}$$

- Conditional:

$$\textbf{if}\ \textit{pred}\ \textbf{then}\ \textit{prog}\ \textbf{else}\ \textit{prog'}\ \textbf{fi} := \textit{prog}\ {}_{[\textit{pred}]}\!\oplus\ \textit{prog'}$$

- Multi-way probabilistic choices
- Variations on ${}_p\oplus$
- Demonic choice in variable assignments
- Iteration

$$\textbf{do}\ \textit{pred} \rightarrow \textit{body}\ \textbf{od} := (\textbf{mu}\ xxx \cdot (\textit{body}; xxx)\ \textbf{if}\ \textit{pred}\ \textbf{else skip})$$

- Implication-like relations for expectations *exp*, *exp'*:

$exp \Rrightarrow exp'$	means	*exp* is everywhere less than or equal to *exp'*
$exp \equiv exp'$	means	*exp* and *exp'* are everywhere equal
$exp \Lleftarrow exp'$	means	*exp* is everywhere greater than or equal to *exp'*

wp-Semantics of pGCL

wp-Semantics of pGCL

$$\begin{aligned} \mathit{wp}.\textbf{abort}.\mathit{postE} &:= 0 \\ \mathit{wp}.\textbf{skip}.\mathit{postE} &:= \mathit{postE} \end{aligned}$$

wp-Semantics of pGCL

$$\begin{aligned}
\mathit{wp}.\mathbf{abort}.\mathit{postE} &:= 0 \\
\mathit{wp}.\mathbf{skip}.\mathit{postE} &:= \mathit{postE} \\
\mathit{wp}.(x := \mathit{expr}).\mathit{postE} &:= \mathit{postE}\langle x \mapsto \mathit{expr}\rangle
\end{aligned}$$

wp-Semantics of pGCL

$$
\begin{aligned}
\textit{wp}.\textbf{abort}.\textit{postE} &:= 0\\
\textit{wp}.\textbf{skip}.\textit{postE} &:= \textit{postE}\\
\textit{wp}.(x := \textit{expr}).\textit{postE} &:= \textit{postE}\langle x \mapsto \textit{expr}\rangle\\
\textit{wp}.(\textit{prog}; \textit{prog'}).\textit{postE} &:= \textit{wp}.\textit{prog}.(\textit{wp}.\textit{prog'}.\textit{postE})
\end{aligned}
$$

wp-Semantics of pGCL

$$
\begin{aligned}
\mathit{wp}.\mathbf{abort}.\mathit{postE} &:= 0 \\
\mathit{wp}.\mathbf{skip}.\mathit{postE} &:= \mathit{postE} \\
\mathit{wp}.(x := \mathit{expr}).\mathit{postE} &:= \mathit{postE}\langle x \mapsto \mathit{expr}\rangle \\
\mathit{wp}.(\mathit{prog}; \mathit{prog}').\mathit{postE} &:= \mathit{wp}.\mathit{prog}.(\mathit{wp}.\mathit{prog}'.\mathit{postE}) \\
\mathit{wp}.(\mathit{prog} \sqcap \mathit{prog}').\mathit{postE} &:= \mathit{wp}.\mathit{prog}.\mathit{postE} \min \mathit{wp}.\mathit{prog}'.\mathit{postE}
\end{aligned}
$$

wp-Semantics of pGCL

$$\begin{aligned}
\textit{wp}.\textbf{abort}.\textit{postE} &:= 0 \\
\textit{wp}.\textbf{skip}.\textit{postE} &:= \textit{postE} \\
\textit{wp}.(x := \textit{expr}).\textit{postE} &:= \textit{postE}\langle x \mapsto \textit{expr}\rangle \\
\textit{wp}.(\textit{prog}; \textit{prog'}).\textit{postE} &:= \textit{wp}.\textit{prog}.(\textit{wp}.\textit{prog'}.\textit{postE}) \\
\textit{wp}.(\textit{prog} \sqcap \textit{prog'}).\textit{postE} &:= \textit{wp}.\textit{prog}.\textit{postE} \min \textit{wp}.\textit{prog'}.\textit{postE} \\
\textit{wp}.(\textit{prog} \,{}_p\!\oplus\, \textit{prog'}).\textit{postE} &:= p * \textit{wp}.\textit{prog}.\textit{postE} + (1-p) * \textit{wp}.\textit{prog'}.\textit{postE}
\end{aligned}$$

Overview

- Guarded Command Language (GCL)
- Probabilistic Guarded Command Language (pGCL)
- Abstraction and Refinement
- Probably Hoare? Hoare Probably!
- Abstraction Refinement and Proof for Probabilistic Systems

What is Abstraction?

Abstraction is the process of extracting the underlying structures, patterns or properties of a mathematical concept or object, and generalizing it so that it has wider applications or matching among other abstract descriptions of equivalent phenomena. — Wikipedia

What is Refinement? (Specialization)

Refinement is the process of refining the underlying structures, patterns or properties of mathematical concepts or objects to a more specialized version.

What is Abstraction?

Abstraction is the process of extracting the underlying structures, patterns or properties of a mathematical concept or object, and generalizing it so that it has wider applications or matching among other abstract descriptions of equivalent phenomena. — Wikipedia

What is Refinement? (Specialization)

Refinement is the process of refining the underlying structures, patterns or properties of mathematical concepts or objects to a more specialized version.

Consider the input set $\mathcal{I}$ for functions/programs f, g, then g is a refinement of f if

$$\{g(i) \mid i \in \mathcal{I}\} \subset^* \{f(i) \mid i \in \mathcal{I}\}$$

*: N.B.: This is not true for all types of abstraction or how abstraction is used.

Example

$$x := -y \; {}_{\frac{1}{3}}\!\oplus \; x := +y$$

Example

$$x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y$$

We want to calculate:

$$\mathit{wp}.(x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y).[x \geq 0]$$

Example

$$x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y$$

We want to calculate:

$$\textit{wp}.(x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y).[x \geq 0]$$

Which means, “what is the probability that the predicate ’the final state, will satisfy $x \geq 0$’ holds in some given initial state of the program?”

Example

$$x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y$$

We want to calculate:

$$\textit{wp}.(x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y).[x \geq 0]$$

Which means, "what is the probability that the predicate 'the final state, will satisfy $x \geq 0$' holds in some given initial state of the program?"

$$\textit{wp}.(x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y).[x \geq 0]$$

Example

$$x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y$$

We want to calculate:

$$\textit{wp}.(x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y).[x \geq 0]$$

Which means, "what is the probability that the predicate 'the final state, will satisfy $x \geq 0$' holds in some given initial state of the program?"

$$\begin{aligned} & \textit{wp}.(x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y).[x \geq 0] \\ \equiv\ & \frac{1}{3} * \textit{wp}.(x := -y).[x \geq 0] + \frac{2}{3} * \textit{wp}.(x := +y).[x \geq 0] \end{aligned}$$

Example

$$x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y$$

We want to calculate:

$$\textit{wp}.(x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y).[x \geq 0]$$

Which means, "what is the probability that the predicate 'the final state, will satisfy $x \geq 0$' holds in some given initial state of the program?"

$$\begin{aligned}
& \textit{wp}.(x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y).[x \geq 0] \\
\equiv\; & \frac{1}{3} * \textit{wp}.(x := -y).[x \geq 0] + \frac{2}{3} * \textit{wp}.(x := +y).[x \geq 0] \\
\equiv\; & \frac{1}{3} * [-y \geq 0] + \frac{2}{3} * [+y \geq 0]
\end{aligned}$$

Example

$$x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y$$

We want to calculate:

$$\mathit{wp}.(x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y).[x \geq 0]$$

Which means, "what is the probability that the predicate 'the final state, will satisfy $x \geq 0$' holds in some given initial state of the program?"

$$\begin{aligned}
& \mathit{wp}.(x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y).[x \geq 0] \\
\equiv\; & \frac{1}{3} * \mathit{wp}.(x := -y).[x \geq 0] + \frac{2}{3} * \mathit{wp}.(x := +y).[x \geq 0] \\
\equiv\; & \frac{1}{3} * [-y \geq 0] + \frac{2}{3} * [+y \geq 0] \\
\equiv\; & \frac{[y < 0]}{3} + [y = 0] + \frac{2[+y \geq 0]}{3}
\end{aligned}$$

Example cont'd

$$\frac{[y < 0]}{3} + [y = 0] + \frac{2[+y \geq 0]}{3}$$

This is our calculated pre-expectation.

Example cont'd

$$\frac{[y < 0]}{3} + [y = 0] + \frac{2[+y \geq 0]}{3}$$

This is our calculated pre-expectation.

The probabilities can be read off from it:

when $y < 0$	$\frac{1}{3} + 0 + \frac{2 * 0}{3} = \frac{1}{3}$
when $y = 0$	$\frac{0}{3} + 1 + \frac{2 * 0}{3} = 1$
when $y > 0$	$\frac{0}{3} + 0 + \frac{2 * 1}{3} = \frac{2}{3}$

Example cont'd

$$\frac{[y < 0]}{3} + [y = 0] + \frac{2[+y \geq 0]}{3}$$

This is our calculated pre-expectation.

The probabilities can be read off from it:

when $y < 0$	$\frac{1}{3} + 0 + \frac{2 * 0}{3} = \frac{1}{3}$
when $y = 0$	$\frac{0}{3} + 1 + \frac{2 * 0}{3} = 1$
when $y > 0$	$\frac{0}{3} + 0 + \frac{2 * 1}{3} = \frac{2}{3}$

How can we build a more abstract program of this Example?

$$x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y$$

Example Abstraction

- $x := -y$ is to be executed with probability at least $\frac{1}{3}$
- $x := +y$ is to be executed with probability at least $\frac{1}{4}$
- it is certain that one or the other will be executed

Example Abstraction

- $x := -y$ is to be executed with probability at least $\frac{1}{3}$
- $x := +y$ is to be executed with probability at least $\frac{1}{4}$
- it is certain that one or the other will be executed

What else can we say from this specification?

Example Abstraction

- $x := -y$ is to be executed with probability at least $\frac{1}{3}$
- $x := +y$ is to be executed with probability at least $\frac{1}{4}$
- it is certain that one or the other will be executed

What else can we say from this specification?

$$x := -y \;{}_{\frac{1}{3}}\!\oplus x := +y \sqcap x := -y \;{}_{\frac{3}{4}}\!\oplus x := +y$$

We can also specify that a program part is executed given some range of probability.

Example Abstraction cont'd

$$(x := -y \ {}_{\frac{1}{3}}\!\oplus x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus x := +y)$$

Example Abstraction cont'd

$$(x := -y \ {}_{\frac{1}{3}}\!\oplus x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus x := +y)$$

Using again the *wp*-semantics, we compute the following

$$wp.((x := -y \ {}_{\frac{1}{3}}\!\oplus x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus x := +y)).[x \geq 0]$$

Example Abstraction cont'd

$$(x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus\ x := +y)$$

Using again the *wp*-semantics, we compute the following

$$\begin{aligned} & \mathit{wp}.((x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus\ x := +y)).[x \geq 0] \\ \equiv\ & \mathit{wp}.(x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y).[x \geq 0] \min \mathit{wp}.(x := -y \ {}_{\frac{3}{4}}\!\oplus\ x := +y).[x \geq 0] \end{aligned}$$

Example Abstraction cont'd

$$(x := -y \ {}_{\frac{1}{3}}\!\oplus x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus x := +y)$$

Using again the *wp*-semantics, we compute the following

$$\begin{aligned}
& \mathit{wp}.((x := -y \ {}_{\frac{1}{3}}\!\oplus x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus x := +y)).[x \geq 0] \\
\equiv\ & \mathit{wp}.(x := -y \ {}_{\frac{1}{3}}\!\oplus x := +y).[x \geq 0] \min \mathit{wp}.(x := -y \ {}_{\frac{3}{4}}\!\oplus x := +y).[x \geq 0] \\
\equiv\ & \frac{[y \leq 0]}{3} + \frac{2 * [y \geq 0]}{3} \min \frac{3 * [y \leq 0]}{4} + \frac{[y \geq 0]}{4}
\end{aligned}$$

Example Abstraction cont'd

$$(x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus\ x := +y)$$

Using again the *wp*-semantics, we compute the following

$$\begin{aligned}
& \mathit{wp}.((x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus\ x := +y)).[x \geq 0] \\
\equiv\ & \mathit{wp}.(x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y).[x \geq 0] \min \mathit{wp}.(x := -y \ {}_{\frac{3}{4}}\!\oplus\ x := +y).[x \geq 0] \\
\equiv\ & \frac{[y \leq 0]}{3} + \frac{2 * [y \geq 0]}{3} \min \frac{3 * [y \leq 0]}{4} + \frac{[y \geq 0]}{4} \\
\equiv\ & \frac{[y < 0]}{3} + [y = 0] + \frac{[y > 0]}{4}
\end{aligned}$$

Example Refinement

Refinement is the converse of abstraction:

$$S \sqsubseteq T := \textit{wp}.S.R \Rrightarrow \textit{wp}.T.R \qquad \text{for all } R$$

Example Refinement

Refinement is the converse of abstraction:

$$S \sqsubseteq T := \mathit{wp}.S.R \Rrightarrow \mathit{wp}.T.R \qquad \text{for all } R$$

Consider the program of before:

$$(x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus\ x := +y)$$

Example Refinement

Refinement is the converse of abstraction:

$$S \sqsubseteq T := \textit{wp}.S.R \Rrightarrow \textit{wp}.T.R \qquad \text{for all } R$$

Consider the program of before:

$$(x := -y \;{}_{\frac{1}{3}}\!\oplus x := +y) \sqcap (x := -y \;{}_{\frac{3}{4}}\!\oplus x := +y)$$

This programs is a refinement according to the specification:

$$x := -y \;{}_{\frac{1}{2}}\!\oplus x := +y$$

Example Refinement cont'd

Prove the following refinement relation:

$$(x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus\ x := +y)$$
$$\Rrightarrow x := -y \ {}_{\frac{1}{2}}\!\oplus\ x := +y$$

Example Refinement cont'd

Prove the following refinement relation:

$$
\begin{aligned}
&(x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y) \sqcap (x := -y \;{}_{\frac{3}{4}}\!\oplus\; x := +y) \\
\Rrightarrow\; & x := -y \;{}_{\frac{1}{2}}\!\oplus\; x := +y
\end{aligned}
$$

Semantic Level

$$
\mathit{wp}.(x := -y \;{}_{\frac{1}{2}}\!\oplus\; x := +y).P
$$

Example Refinement cont'd

Prove the following refinement relation:

$$(x := -y \ {}_{\frac{1}{3}}\!\oplus x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus x := +y)$$
$$\Rrightarrow x := -y \ {}_{\frac{1}{2}}\!\oplus x := +y$$

Semantic Level

$$\mathit{wp}.(x := -y \ {}_{\frac{1}{2}}\!\oplus x := +y).P$$
$$\equiv \frac{\mathit{wp}.(x := -y).P}{2} + \frac{\mathit{wp}.(x := +y).P}{2}$$

Example Refinement cont'd

Prove the following refinement relation:

$$(x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus\ x := +y)$$
$$\Rrightarrow x := -y \ {}_{\frac{1}{2}}\!\oplus\ x := +y$$

Semantic Level

$$\begin{aligned} & \mathit{wp}.(x := -y \ {}_{\frac{1}{2}}\!\oplus\ x := +y).P \\ \equiv & \frac{\mathit{wp}.(x := -y).P}{2} + \frac{\mathit{wp}.(x := +y).P}{2} \\ \equiv & \frac{P^-}{2} + \frac{P^+}{2} \end{aligned}$$

Example Refinement cont'd

Prove the following refinement relation:

$$(x := -y \;{}_{\frac{1}{3}}\!\oplus x := +y) \sqcap (x := -y \;{}_{\frac{3}{4}}\!\oplus x := +y)$$
$$\Rrightarrow x := -y \;{}_{\frac{1}{2}}\!\oplus x := +y$$

Semantic Level

$$\begin{aligned}
& \mathit{wp}.(x := -y \;{}_{\frac{1}{2}}\!\oplus x := +y).P \\
\equiv & \frac{\mathit{wp}.(x := -y).P}{2} + \frac{\mathit{wp}.(x := +y).P}{2} \\
\equiv & \frac{P^-}{2} + \frac{P^+}{2} \\
\equiv & \frac{3}{5} * (\frac{P^-}{3} + \frac{2 * P^+}{3}) + \frac{2}{5} * (\frac{3 * P^-}{4} + \frac{P^+}{4})
\end{aligned}$$

$$
\begin{aligned}
&\textit{wp}.(x := -y \ {}_{\frac{1}{2}}\!\oplus\ x := +y).P \\
\equiv\ & \frac{\textit{wp}.(x := -y).P}{2} + \frac{\textit{wp}.(x := +y).P}{2} \\
\equiv\ & \frac{P^-}{2} + \frac{P^+}{2} \\
\equiv\ & \frac{3}{5} * (\frac{P^-}{3} + \frac{2 * P^+}{3}) + \frac{2}{5} * (\frac{3 * P^-}{4} + \frac{P^+}{4})
\end{aligned}
$$

$$
\begin{aligned}
& \textit{wp}.(x := -y \ {}_{\frac{1}{2}}\oplus \ x := +y).P \\
\equiv \ & \frac{\textit{wp}.(x := -y).P}{2} + \frac{\textit{wp}.(x := +y).P}{2} \\
\equiv \ & \frac{P^-}{2} + \frac{P^+}{2} \\
\equiv \ & \frac{3}{5} * (\frac{P^-}{3} + \frac{2 * P^+}{3}) + \frac{2}{5} * (\frac{3 * P^-}{4} + \frac{P^+}{4}) \\
\Leftarrow \ & \frac{P^-}{3} + \frac{2 * P^+}{3} \min \frac{3 * P^-}{4} + \frac{P^+}{4}
\end{aligned}
$$

because $\frac{3}{5} * \frac{1}{3} + \frac{2}{5} * \frac{3}{4} = \frac{1}{2}$
and $\frac{3}{5} * \frac{2}{3} + \frac{2}{5} * \frac{1}{4} = \frac{1}{2}$

$$
\begin{aligned}
& \textit{wp}.(x := -y \;{}_{\frac{1}{2}}\!\oplus x := +y).P \\
\equiv\; & \frac{\textit{wp}.(x := -y).P}{2} + \frac{\textit{wp}.(x := +y).P}{2} \\
\equiv\; & \frac{P^-}{2} + \frac{P^+}{2} \\
\equiv\; & \frac{3}{5} * (\frac{P^-}{3} + \frac{2 * P^+}{3}) + \frac{2}{5} * (\frac{3 * P^-}{4} + \frac{P^+}{4}) \\
\Leftarrow\; & \frac{P^-}{3} + \frac{2 * P^+}{3} \min \frac{3 * P^-}{4} + \frac{P^+}{4}
\end{aligned}
$$

because $\frac{3}{5} * \frac{1}{3} + \frac{2}{5} * \frac{3}{4} = \frac{1}{2}$
and $\frac{3}{5} * \frac{2}{3} + \frac{2}{5} * \frac{1}{4} = \frac{1}{2}$

$$
\equiv\; \textit{wp}.(x := -y \;{}_{\frac{1}{3}}\!\oplus x := +y \sqcap x := -y \;{}_{\frac{3}{4}}\!\oplus x := +y).P
$$

Program Level

$$x := -y \;{}_{\frac{1}{2}}\!\oplus\; x := +y$$

Program Level

$$
\begin{aligned}
& x := -y \ {}_{\frac{1}{2}}\!\oplus\ x := +y \\
= {} & (x := -y \ {}_{\frac{1}{3}}\!\oplus\ x := +y) \ {}_{\frac{3}{5}}\!\oplus\ (x := -y \ {}_{\frac{3}{4}}\!\oplus\ x := +y)
\end{aligned}
$$

Program Level

$$
\begin{aligned}
& x := -y \ {}_{\frac{1}{2}}\!\oplus x := +y \\
= & \ (x := -y \ {}_{\frac{1}{3}}\!\oplus x := +y) \ {}_{\frac{3}{5}}\!\oplus (x := -y \ {}_{\frac{3}{4}}\!\oplus x := +y) \\
\sqsupseteq & \ (x := -y \ {}_{\frac{1}{3}}\!\oplus x := +y) \sqcap (x := -y \ {}_{\frac{3}{4}}\!\oplus x := +y)
\end{aligned}
$$

Program Level

$$
\begin{aligned}
& x := -y \;{}_{\frac{1}{2}}\!\oplus\; x := +y \\
= \; & (x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y) \;{}_{\frac{3}{5}}\!\oplus\; (x := -y \;{}_{\frac{3}{4}}\!\oplus\; x := +y) \\
\sqsupseteq \; & (x := -y \;{}_{\frac{1}{3}}\!\oplus\; x := +y) \sqcap (x := -y \;{}_{\frac{3}{4}}\!\oplus\; x := +y)
\end{aligned}
$$

N.B.: Demonic choice can be refined by any probabilistic choice.

Interpretation of pGCL Expectations

- in full generality, an expectation is a function describing the value of a program state

Interpretation of pGCL Expectations

- in full generality, an expectation is a function describing the value of a program state
- where $[pred]$ is a special case assigning 0 or 1 as value

Interpretation of pGCL Expectations

- in full generality, an expectation is a function describing the value of a program state
- where $[pred]$ is a special case assigning 0 or 1 as value
- more general expectations: estimate the value of final state in the initial state

Interpretation of pGCL Expectations

- in full generality, an expectation is a function describing the value of a program state
- where $[pred]$ is a special case assigning 0 or 1 as value
- more general expectations: estimate the value of final state in the initial state
  $\Rightarrow$ summation over final states

Interpretation of pGCL Expectations

- in full generality, an expectation is a function describing the value of a program state
- where $[pred]$ is a special case assigning 0 or 1 as value
- more general expectations: estimate the value of final state in the initial state
  $\Rightarrow$ summation over final states
  $\Rightarrow$ the value of the final state multiplied by the probability the program "will go there" from the initial state

Interpretation of pGCL Expectations

- in full generality, an expectation is a function describing the value of a program state
- where $[pred]$ is a special case assigning 0 or 1 as value
- more general expectations: estimate the value of final state in the initial state
  $\Rightarrow$ summation over final states
  $\Rightarrow$ the value of the final state multiplied by the probability the program "will go there" from the initial state
- naturally "will go there" depends on "from where"

Interpretation of pGCL Expectations

- in full generality, an expectation is a function describing the value of a program state
- where $[pred]$ is a special case assigning 0 or 1 as value
- more general expectations: estimate the value of final state in the initial state
  $\Rightarrow$ summation over final states
  $\Rightarrow$ the value of the final state multiplied by the probability the program "will go there" from the initial state
- naturally "will go there" depends on "from where"

Analyses of programs S lead to conclusions like

$$p \equiv \textit{wp}.S.[P]$$

for some p and $[P]$.

Interpretation of pGCL Expectations

- in full generality, an expectation is a function describing the value of a program state
- where $[pred]$ is a special case assigning 0 or 1 as value
- more general expectations: estimate the value of final state in the initial state
  $\Rightarrow$ summation over final states
  $\Rightarrow$ the value of the final state multiplied by the probability the program "will go there" from the initial state
- naturally "will go there" depends on "from where"

Analyses of programs S lead to conclusions like

$$p \equiv \textit{wp}.S.[P]$$

for some p and $[P]$. We can interpret them in two equivalent ways:

1. the expected value $[P]$ of the final state is at least the value of p in the initial state; or

Interpretation of pGCL Expectations

- in full generality, an expectation is a function describing the value of a program state
- where $[pred]$ is a special case assigning 0 or 1 as value
- more general expectations: estimate the value of final state in the initial state
  $\Rightarrow$ summation over final states
  $\Rightarrow$ the value of the final state multiplied by the probability the program "will go there" from the initial state
- naturally "will go there" depends on "from where"

Analyses of programs S lead to conclusions like

$$p \equiv \mathit{wp}.S.[P]$$

for some p and $[P]$. We can interpret them in two equivalent ways:

1. the expected value $[P]$ of the final state is at least the value of p in the initial state; or
2. the probability that S will establish P is at least p.

Example

The probability that two fair coins, when flipped, show the same faces:

Example

The probability that two fair coins, when flipped, show the same faces:

$$\mathit{wp}.\begin{pmatrix} x := H \ {}_{\frac{1}{2}}\!\oplus\ x := T; \\ y := H \ {}_{\frac{1}{2}}\!\oplus\ y := T \end{pmatrix}.[x = y]$$

Example

The probability that two fair coins, when flipped, show the same faces:

$$
\begin{aligned}
& \mathit{wp}.\begin{pmatrix} x := H \ {}_{\frac{1}{2}}\!\oplus\ x := T; \\ y := H \ {}_{\frac{1}{2}}\!\oplus\ y := T \end{pmatrix}.[x = y] \\
\equiv\ & \mathit{wp}.(x := H \ {}_{\frac{1}{2}}\!\oplus\ x := T).\left(\frac{[x = H]}{2} + \frac{[x = T]}{2}\right)
\end{aligned}
$$

Example

The probability that two fair coins, when flipped, show the same faces:

$$
\begin{aligned}
& \textit{wp}.\begin{pmatrix} x := H \ {}_{\frac{1}{2}}\!\oplus\ x := T; \\ y := H \ {}_{\frac{1}{2}}\!\oplus\ y := T \end{pmatrix}.[x = y] \\
\equiv\ & \textit{wp}.(x := H \ {}_{\frac{1}{2}}\!\oplus\ x := T).\left(\frac{[x = H]}{2} + \frac{[x = T]}{2}\right) \\
\equiv\ & \frac{1}{2}\left(\frac{[H = H]}{2} + \frac{[H = T]}{2}\right) + \frac{1}{2}\left(\frac{[T = H]}{2} + \frac{[T = T]}{2}\right)
\end{aligned}
$$

Example

The probability that two fair coins, when flipped, show the same faces:

$$\begin{aligned}
& \textit{wp}.\begin{pmatrix} x := H \;{}_{\frac{1}{2}}\!\oplus\; x := T; \\ y := H \;{}_{\frac{1}{2}}\!\oplus\; y := T \end{pmatrix}.[x = y] \\
\equiv\; & \textit{wp}.(x := H \;{}_{\frac{1}{2}}\!\oplus\; x := T).\left(\frac{[x = H]}{2} + \frac{[x = T]}{2}\right) \\
\equiv\; & \frac{1}{2}\left(\frac{[H = H]}{2} + \frac{[H = T]}{2}\right) + \frac{1}{2}\left(\frac{[T = H]}{2} + \frac{[T = T]}{2}\right) \\
\equiv\; & \frac{1}{2}\left(\frac{1}{2} + \frac{0}{2}\right) + \frac{1}{2}\left(\frac{0}{2} + \frac{1}{2}\right)
\end{aligned}$$

Example

The probability that two fair coins, when flipped, show the same faces:

$$\begin{aligned}
& \mathit{wp}.\begin{pmatrix} x := H \ {}_{\frac{1}{2}}\!\oplus x := T; \\ y := H \ {}_{\frac{1}{2}}\!\oplus y := T \end{pmatrix}.[x = y] \\
\equiv\ & \mathit{wp}.(x := H \ {}_{\frac{1}{2}}\!\oplus x := T).\left(\frac{[x = H]}{2} + \frac{[x = T]}{2}\right) \\
\equiv\ & \frac{1}{2}\left(\frac{[H = H]}{2} + \frac{[H = T]}{2}\right) + \frac{1}{2}\left(\frac{[T = H]}{2} + \frac{[T = T]}{2}\right) \\
\equiv\ & \frac{1}{2}\left(\frac{1}{2} + \frac{0}{2}\right) + \frac{1}{2}\left(\frac{0}{2} + \frac{1}{2}\right) \equiv \frac{1}{2}
\end{aligned}$$

Example

The probability that two fair coins, when flipped, show the same faces:

$$
\begin{aligned}
& \mathit{wp}.\begin{pmatrix} x := H \;{}_{\frac{1}{2}}\!\oplus\; x := T; \\ y := H \;{}_{\frac{1}{2}}\!\oplus\; y := T \end{pmatrix}.[x = y] \\
\equiv\; & \mathit{wp}.(x := H \;{}_{\frac{1}{2}}\!\oplus\; x := T).\left(\frac{[x = H]}{2} + \frac{[x = T]}{2}\right) \\
\equiv\; & \frac{1}{2}\left(\frac{[H = H]}{2} + \frac{[H = T]}{2}\right) + \frac{1}{2}\left(\frac{[T = H]}{2} + \frac{[T = T]}{2}\right) \\
\equiv\; & \frac{1}{2}\left(\frac{1}{2} + \frac{0}{2}\right) + \frac{1}{2}\left(\frac{0}{2} + \frac{1}{2}\right) \equiv \frac{1}{2}
\end{aligned}
$$

Apply second interpretation: the faces are the same with probability $\frac{1}{2}$

Example

The probability that two fair coins, when flipped, show the same faces:

$$
\begin{aligned}
& wp.\begin{pmatrix} x := H \; {}_{\frac{1}{2}}\oplus \; x := T; \\ y := H \; {}_{\frac{1}{2}}\oplus \; y := T \end{pmatrix}.[x = y] \\
\equiv\; & wp.(x := H \; {}_{\frac{1}{2}}\oplus \; x := T).\left(\frac{[x = H]}{2} + \frac{[x = T]}{2}\right) \\
\equiv\; & \frac{1}{2}\left(\frac{[H = H]}{2} + \frac{[H = T]}{2}\right) + \frac{1}{2}\left(\frac{[T = H]}{2} + \frac{[T = T]}{2}\right) \\
\equiv\; & \frac{1}{2}\left(\frac{1}{2} + \frac{0}{2}\right) + \frac{1}{2}\left(\frac{0}{2} + \frac{1}{2}\right) \equiv \frac{1}{2}
\end{aligned}
$$

Apply second interpretation: the faces are the same with probability $\frac{1}{2}$
How to interpret the expectations in

$$
wp.(x := H \; {}_{\frac{1}{2}}\oplus \; x := T).\left(\frac{[x = H]}{2} + \frac{[x = T]}{2}\right)?
$$

How to interpret the expectations in

$$\mathit{wp}.(x := H \ {}_{\frac{1}{2}}\!\oplus\ x := T).\left(\frac{[x = H]}{2} + \frac{[x = T]}{2}\right)?$$

Interpretations:

How to interpret the expectations in

$$\mathit{wp}.(x := H \;{}_{\frac{1}{2}}\!\oplus\; x := T).\Big(\frac{[x = H]}{2} + \frac{[x = T]}{2}\Big)?$$

Interpretations:

2. the probability that S will establish P is at least p.
   $\Rightarrow$ will establish $\frac{[x=H]}{2} + \frac{[x=T]}{2}$

How to interpret the expectations in

$$wp.(x := H \ {}_{\frac{1}{2}}\oplus \ x := T).\left(\frac{[x = H]}{2} + \frac{[x = T]}{2}\right)?$$

Interpretations:

2. the probability that S will establish P is at least p.
   $\Rightarrow$ will establish $\frac{[x=H]}{2} + \frac{[x=T]}{2}$

How to interpret the expectations in

$$wp.(x := H \,{}_{\frac{1}{2}}\!\oplus\, x := T).\left(\frac{[x = H]}{2} + \frac{[x = T]}{2}\right)?$$

Interpretations:

1. the expected value $[P]$ of the final state is at least the value of p in the initial state
2. the probability that S will establish P is at least p.
   $\Rightarrow$ will establish $\frac{[x=H]}{2} + \frac{[x=T]}{2}$

How to interpret the expectations in

$$wp.(x := H \;{}_{\frac{1}{2}}\!\oplus\; x := T).\left(\frac{[x = H]}{2} + \frac{[x = T]}{2}\right)?$$

Interpretations:

1. the expected value $[P]$ of the final state is at least the value of p in the initial state
   $\Rightarrow$ the expected value of the expression $\frac{[x=H]}{2} + \frac{[x=T]}{2}$ after executing the program $x := H \;{}_{\frac{1}{2}}\!\oplus\; x := T$
2. the probability that S will establish P is at least p.
   $\Rightarrow$ will establish $\frac{[x=H]}{2} + \frac{[x=T]}{2}$

How to interpret the expectations in

$$\textit{wp}.(x := H \ {}_{\frac{1}{2}}\!\oplus \ x := T).\left(\frac{[x = H]}{2} + \frac{[x = T]}{2}\right)?$$

Interpretations:

1. the expected value $[P]$ of the final state is at least the value of p in the initial state
   $\Rightarrow$ the expected value of the expression $\frac{[x=H]}{2} + \frac{[x=T]}{2}$ after executing the program $x := H \ {}_{\frac{1}{2}}\!\oplus \ x := T$
2. the probability that S will establish P is at least p.
   $\Rightarrow$ will establish $\frac{[x=H]}{2} + \frac{[x=T]}{2}$

For our overall reasoning we only need the second interpretation and the first one is only "glue" that holds our reasoning together.

Properties of pGCL

All GCL commands satisfy conjunctivity:

$$\textit{wp}.S.(P \wedge P') = \textit{wp}.S.P \wedge \textit{wp}.S.P'$$

Properties of pGCL

All GCL commands satisfy conjunctivity:

$$\mathit{wp}.S.(P \wedge P') = \mathit{wp}.S.P \wedge \mathit{wp}.S.P'$$

Do we need that also for pGCL?

Properties of pGCL

All GCL commands satisfy conjunctivity:

$$\mathit{wp}.S.(P \wedge P') = \mathit{wp}.S.P \wedge \mathit{wp}.S.P'$$

Do we need that also for pGCL?

We do not have conjunctivity in pGCL, but sub-linearity (it generalizes conjunctivity): Let a, b, c be non-negative finite reals, and P, P' expectations, then all pGCL constructs satisfy

$$\mathit{wp}.S.(aP + bP' \ominus c) \Leftarrow a(\mathit{wp}.S.P) + b(\mathit{wp}.S.P') \ominus c$$

Properties of pGCL

All GCL commands satisfy conjunctivity:

$$\textit{wp}.S.(P \wedge P') = \textit{wp}.S.P \wedge \textit{wp}.S.P'$$

Do we need that also for pGCL?

We do not have conjunctivity in pGCL, but sub-linearity (it generalizes conjunctivity): Let a, b, c be non-negative finite reals, and P, P' expectations, then all pGCL constructs satisfy

$$\textit{wp}.S.(aP + bP' \ominus c) \Lleftarrow a(\textit{wp}.S.P) + b(\textit{wp}.S.P') \ominus c$$

where truncated subtraction $\ominus$ is defined as $x \ominus y := (x - y) \max 0$

Monotonicity

Increasing a post-expectation can only increase the pre-expectation. Suppose $P \Rrightarrow P'$ for two expectations P, P' then

$$\mathit{wp}.S.P \Rrightarrow \mathit{wp}.S.P'$$

Monotonicity

Increasing a post-expectation can only increase the pre-expectation. Suppose $P \Rrightarrow P'$ for two expectations P, P' then

$$\textit{wp}.S.P \Rrightarrow \textit{wp}.S.P'$$

Suppose $P \Rrightarrow P'$ for two expectations P, P':

Monotonicity

Increasing a post-expectation can only increase the pre-expectation. Suppose $P \Rrightarrow P'$ for two expectations P, P' then

$$\mathit{wp}.S.P \Rrightarrow \mathit{wp}.S.P'$$

Suppose $P \Rrightarrow P'$ for two expectations P, P':

$$\mathit{wp}.S.P'$$

Monotonicity

Increasing a post-expectation can only increase the pre-expectation. Suppose $P \Rrightarrow P'$ for two expectations P, P' then

$$\mathit{wp}.S.P \Rrightarrow \mathit{wp}.S.P'$$

Suppose $P \Rrightarrow P'$ for two expectations P, P':

$$\begin{aligned} & \mathit{wp}.S.P' \\ \equiv\ & \mathit{wp}.S.(P + (P' - P)) \end{aligned}$$

Monotonicity

Increasing a post-expectation can only increase the pre-expectation. Suppose $P \Rrightarrow P'$ for two expectations P, P' then

$$\mathit{wp}.S.P \Rrightarrow \mathit{wp}.S.P'$$

Suppose $P \Rrightarrow P'$ for two expectations P, P':

$$\begin{aligned} & \mathit{wp}.S.P' \\ \equiv\ & \mathit{wp}.S.(P + (P' - P)) \\ \Lleftarrow\ & \mathit{wp}.S.P + \mathit{wp}.S.(P' - P) \end{aligned}$$

Monotonicity

Increasing a post-expectation can only increase the pre-expectation. Suppose $P \Rrightarrow P'$ for two expectations P, P' then

$$\mathit{wp}.S.P \Rrightarrow \mathit{wp}.S.P'$$

Suppose $P \Rrightarrow P'$ for two expectations P, P':

$$\begin{aligned} & \mathit{wp}.S.P' \\ \equiv\ & \mathit{wp}.S.(P + (P' - P)) \\ \Lleftarrow\ & \mathit{wp}.S.P + \mathit{wp}.S.(P' - P) \\ \Lleftarrow\ & \mathit{wp}.S.P \end{aligned}$$

Feasibility

Pre-expectations cannot be “too large”.

$$\mathit{wp}.S.P \Rrightarrow \max P$$

Feasibility

Pre-expectations cannot be "too large".

$$\mathit{wp}.S.P \Rrightarrow \max P$$

Scaling

Multiplication by a non-negative constant distributes through commands. Note we already have one direction due to sub-linearity:

$$c * \mathit{wp}.S.P \Rrightarrow \mathit{wp}.S.(c * P)$$

Feasibility

Pre-expectations cannot be "too large".

$$\textit{wp}.S.P \Rrightarrow \max P$$

Scaling

Multiplication by a non-negative constant distributes through commands.

$$c * \textit{wp}.S.P \equiv \textit{wp}.S.(c * P)$$

Probabilistic Conjunctivity?

- standard "$\wedge$" is not defined over numbers

Probabilistic Conjunctivity?

- standard "$\wedge$" is not defined over numbers
- it should act analogue as "$\wedge$" due to embedded boolean

Probabilistic Conjunctivity?

- standard "$\wedge$" is not defined over numbers
- it should act analogue as "$\wedge$" due to embedded boolean

$$0 \mathbin{\&} 0 = 0$$
$$0 \mathbin{\&} 1 = 0$$
$$1 \mathbin{\&} 0 = 0$$
$$1 \mathbin{\&} 1 = 1$$

Probabilistic Conjunctivity?

- standard "$\wedge$" is not defined over numbers
- it should act analogue as "$\wedge$" due to embedded boolean
- obvious $\min$ and $*$ do not apply

$$0 \mathbin{\&} 0 = 0$$
$$0 \mathbin{\&} 1 = 0$$
$$1 \mathbin{\&} 0 = 0$$
$$1 \mathbin{\&} 1 = 1$$

Probabilistic Conjunctivity?

- standard "$\wedge$" is not defined over numbers
- it should act analogue as "$\wedge$" due to embedded boolean
- obvious $\min$ and $*$ do not apply

We define $\&$ as:

$$exp \;\&\; exp' := exp + exp' \ominus 1$$

$$0 \;\&\; 0 = 0$$
$$0 \;\&\; 1 = 0$$
$$1 \;\&\; 0 = 0$$
$$1 \;\&\; 1 = 1$$

Probabilistic Conjunctivity?

- standard "$\wedge$" is not defined over numbers
- it should act analogue as "$\wedge$" due to embedded boolean
- obvious $\min$ and $*$ do not apply

We define $\&$ as:

$$0 \;\&\; 0 = 0$$
$$0 \;\&\; 1 = 0$$
$$1 \;\&\; 0 = 0$$
$$1 \;\&\; 1 = 1$$

$$\mathit{exp} \;\&\; \mathit{exp'} := \mathit{exp} + \mathit{exp'} \ominus 1$$

Sub-Conjunctivity

As $\&$ sub-distributes through expectation transformers and from sub-linearity with $a, b, c := 1, 1, 1$ we have:

$$\mathit{wp}.S.P \;\&\; \mathit{wp}.S.P' \Rrightarrow \mathit{wp}.S.(P \;\&\; P')$$

for all S.

Overview

- Guarded Command Language (GCL)
- Probabilistic Guarded Command Language (pGCL)
- Abstraction and Refinement
- Probably Hoare? Hoare Probably!
- Abstraction Refinement and Proof for Probabilistic Systems

Do we Have to Deal with Probability in weakest pre-expectations/pre-conditions?

Do we Have to Deal with Probability in weakest pre-expectations/pre-conditions?

Short answer: **Yes.**

Do we Have to Deal with Probability in weakest pre-expectations/pre-conditions?

Short answer: **Yes.**

Why?

Probabilistic Hoare triples would allow easier reasoning:

$$p \vdash \{pre\}\ prog\ \{post\}$$

Hoare triple holds with at least probability p.

Do we Have to Deal with Probability in weakest pre-expectations/pre-conditions?

Short answer: **Yes.**

Why?

Probabilistic Hoare triples would allow easier reasoning:

$$p \vdash \{pre\}\ prog\ \{post\}$$

Hoare triple holds with at least probability p.

Fair & Unfair Coin

Consider the following programs *fair* & *unfair*:

fair $\quad x := A \sqcap (x := B \ {}_{\frac{1}{2}}\!\oplus x := C)$

Do we Have to Deal with Probability in weakest pre-expectations/pre-conditions?

Short answer: **Yes.**

Why?

Probabilistic Hoare triples would allow easier reasoning:

$$p \vdash \{pre\}\ prog\ \{post\}$$

Hoare triple holds with at least probability p.

Fair & Unfair Coin

Consider the following programs *fair* & *unfair*:

fair	$x := A \sqcap (x := B \ {}_{\frac{1}{2}}\!\oplus x := C)$
unfair	$(x := A \sqcap x := B) \ {}_{\frac{1}{2}}\!\oplus (x := A \sqcap x := C)$

Fair & Unfair Coin

Consider the following programs *fair* & *unfair*:

fair	$x := A \sqcap (x := B \ {}_{\frac{1}{2}}\!\oplus\ x := C)$
unfair	$(x := A \sqcap x := B) \ {}_{\frac{1}{2}}\!\oplus\ (x := A \sqcap x := C)$

Fair & Unfair Coin

Consider the following programs *fair* & *unfair*:

fair	$x := A \sqcap (x := B \ {}_{\frac{1}{2}}\!\oplus x := C)$
unfair	$(x := A \sqcap x := B) \ {}_{\frac{1}{2}}\!\oplus (x := A \sqcap x := C)$

The Programs Cannot Be Distinguished

all post-conditions	*fair* probabilities		*unfair* probabilities
false	0	$= 0 =$	0
$x = A$	$1 \min 0$	$= 0 =$	$\frac{1}{2}(1 \min 0) + \frac{1}{2}(1 \min 0)$
$x = B$	$0 \min \frac{1}{2}$	$= 0 =$	$\frac{1}{2}(0 \min 1) + \frac{1}{2}(0 \min 0)$
$x = C$	$0 \min \frac{1}{2}$	$= 0 =$	$\frac{1}{2}(0 \min 0) + \frac{1}{2}(0 \min 1)$
$x \neq A$	$0 \min 1$	$= 0 =$	$\frac{1}{2}(0 \min 1) + \frac{1}{2}(0 \min 1)$
$x \neq B$	$1 \min \frac{1}{2}$	$= \frac{1}{2} =$	$\frac{1}{2}(1 \min 0) + \frac{1}{2}(1 \min 1)$
$x \neq C$	$1 \min \frac{1}{2}$	$= \frac{1}{2} =$	$\frac{1}{2}(1 \min 1) + \frac{1}{2}(1 \min 0)$
true	1	$= 1 =$	1

Hoare Probably!

Let $preExp$, $postExp$ be real-valued expressions in the program variables:

$$\{preExp\}\ prog\ \{postExp\}$$

$preExp$ evaluated in the initial state gives a lower bound for the expected value of expression $postExp$.

Hoare Probably!

Let $preExp$, $postExp$ be real-valued expressions in the program variables:

$$\{preExp\}\ prog\ \{postExp\}$$

$preExp$ evaluated in the initial state gives a lower bound for the expected value of expression $postExp$. It subsumes our earlier defined probably Hoare semantics:

$$\{p \times [pre]\}\ prog\ \{[post]\}$$

From any initial state satisfying pre, $prog$ will reach a final state satisfying $post$ with probability p.

Hoare Probably!

Let $preExp$, $postExp$ be real-valued expressions in the program variables:

$$\{preExp\}\ prog\ \{postExp\}$$

$preExp$ evaluated in the initial state gives a lower bound for the expected value of expression $postExp$. It subsumes our earlier defined probably Hoare semantics:

$$\{p \times [pre]\}\ prog\ \{[post]\}$$

From any initial state satisfying pre, $prog$ will reach a final state satisfying $post$ with probability p.

$postExp$	*fair* $preExp$	*unfair* $preExp$
$[x = A] + 2[x = B]$	1	$\frac{1}{2}$

fair* refines *unfair

Sub-Linearity

For any reals $a, b, c \geq 0$ and expectations $preExp$, $preExp'$, $postExp$, $postExp'$, from

$$\{preExp\}\ S\ \{postExp\}$$
$$\text{and } \{preExp'\}\ S\ \{postExp'\}$$

follows

Sub-Linearity

For any reals $a, b, c \geq 0$ and expectations $preExp$, $preExp'$, $postExp$, $postExp'$, from

$$\{preExp\}\ S\ \{postExp\}$$
$$\text{and } \{preExp'\}\ S\ \{postExp'\}$$

follows

$$\{a \times preExp + b \times preExp' \ominus c\}\ S\ \{a \times postExp + b \times postExp' \ominus c\}$$

Monte Carlo Algorithms

A probabilistic primality testing algorithm establishes a number's prime, with arbitrary high probability, by repeated failure to show that it is composite $\Rightarrow$ example for iterated Monte-Carlo algorithm

Monte Carlo Algorithms

A probabilistic primality testing algorithm establishes a number's prime, with arbitrary high probability, by repeated failure to show that it is composite $\Rightarrow$ example for iterated Monte-Carlo algorithm

We want to decide a computationally expensive Boolean B (e.g. "a given number is prime", proof search). A Monte-Carlo algorithm for that is a computationally cheap and guaranteed-to-terminate procedure which probably decides B (no Las-Vegas procedure).

Monte Carlo Algorithms

A probabilistic primality testing algorithm establishes a number's prime, with arbitrary high probability, by repeated failure to show that it is composite $\Rightarrow$ example for iterated Monte-Carlo algorithm

We want to decide a computationally expensive Boolean B (e.g. "a given number is prime", proof search). A Monte-Carlo algorithm for that is a computationally cheap and guaranteed-to-terminate procedure which probably decides B (no Las-Vegas procedure).

Such a procedure for B could be specified as:

$$b := B \;{}_{\geq p}\!\oplus\; (b := True \sqcap b := False)$$

where B is the desired result.

Monte Carlo Algorithms

A probabilistic primality testing algorithm establishes a number's prime, with arbitrary high probability, by repeated failure to show that it is composite $\Rightarrow$ example for iterated Monte-Carlo algorithm

We want to decide a computationally expensive Boolean B (e.g. "a given number is prime", proof search). A Monte-Carlo algorithm for that is a computationally cheap and guaranteed-to-terminate procedure which probably decides B (no Las-Vegas procedure).

Such a procedure for B could be specified as:

$$b := B \;{}_{\geq p}\!\oplus\; (b := True \sqcap b := False)$$

where B is the desired result.
It is equal to:

$$b := B \sqcap \left(b := B \;{}_{p}\!\oplus\; (b := True \sqcap b := False)\right)$$

Probabilistic Primality

Instantiation with probabilistic primality:

if B **then** $b := True$ **else**

$b := False \,_{\geq \frac{1}{2}}\!\oplus\, b := True$

fi

Probabilistic Primality

Instantiation with probabilistic primality:

$$\textbf{if } B \textbf{ then } b := True \textbf{ else}$$
$$b := False \ {}_{\geq \frac{1}{2}}\oplus \ b := True$$
$$\textbf{fi}$$

We would like to have $[b = B]$ as post-expectation, meaning the program reveals in b the value of unknown B.

Probabilistic Primality

Instantiation with probabilistic primality:

$$\textbf{if } B \textbf{ then } b := True \textbf{ else}$$
$$b := False \;{}_{\geq \frac{1}{2}}\!\oplus\; b := True$$
$$\textbf{fi}$$

We would like to have $[b = B]$ as post-expectation, meaning the program reveals in b the value of unknown B.

$$\{ \qquad\qquad\qquad \} \; Decide \; \{b = B\}$$

Probabilistic Primality

Instantiation with probabilistic primality:

$$\textbf{if } B \textbf{ then } b := True \textbf{ else}$$
$$b := False \ {}_{\geq\frac{1}{2}}\!\oplus b := True$$
$$\textbf{fi}$$

We would like to have $[b = B]$ as post-expectation, meaning the program reveals in b the value of unknown B. As pre-expectation we use $1 \lhd B \rhd 1 - \frac{1}{2^N}$, so we seek for a program $Decide$ which such that

$$\{1 \lhd B \rhd 1 - \frac{1}{2^N}\}\ Decide\ \{b = B\}$$

Probabilistic Primality

Instantiation with probabilistic primality:

$$\textbf{if } B \textbf{ then } b := True \textbf{ else}$$
$$b := False \;_{\geq \frac{1}{2}}\!\oplus\; b := True$$
$$\textbf{fi}$$

We would like to have $[b = B]$ as post-expectation, meaning the program reveals in b the value of unknown B. As pre-expectation we use $1 \lhd B \rhd 1 - \frac{1}{2^N}$, so we seek for a program $Decide$ which such that

$$\{1 \lhd B \rhd 1 - \frac{1}{2^N}\}\ Decide\ \{b = B\}$$

Furthermore we need an invariant and choose:

$$Inv = [b] \lhd B \rhd 1 - \frac{[b]}{2^n}$$

Probabilistic Primality cont'd

$$
\begin{array}{l}
b, n := True, N; \\
\textbf{do } n \neq 0 \wedge b \rightarrow \\
\qquad CheckOnce; \\
\qquad n := n - 1 \\
\textbf{od}
\end{array}
$$

Probabilistic Primality cont'd

$$\{1 \lhd B \rhd 1 - \frac{1}{2^N}\}$$

$b, n := True, N;$

do $n \neq 0 \wedge b \rightarrow$

$\quad CheckOnce;$

$\quad n := n - 1$

od

$$\{b = B\}$$

Probabilistic Primality cont'd

We carry out the checks for the invariant.

Probabilistic Primality cont'd

We carry out the checks for the invariant.

- Check that the invariant is established at initialization:

$$\{1 \lhd B \rhd 1 - \frac{1}{2^N}\}\ b, n := True, N\ \{Inv\}$$

Probabilistic Primality cont'd

We carry out the checks for the invariant.

- Check that the invariant is established at initialization:

$$\{1 \lhd B \rhd 1 - \frac{1}{2^N}\}\ b, n := True, N\ \{Inv\}$$

  That is we want to check

$$1 \lhd B \rhd 1 - \frac{1}{2^N} \Rrightarrow Inv[b, n := True, N]$$

Probabilistic Primality cont'd

We carry out the checks for the invariant.

- Check that the invariant is established at initialization:

$$\{1 \lhd B \rhd 1 - \frac{1}{2^N}\}\ b, n := True, N\ \{Inv\}$$

  That is we want to check

$$\begin{aligned} & 1 \lhd B \rhd 1 - \frac{1}{2^N} \Rrightarrow Inv[b, n := True, N] \\ = & 1 \lhd B \rhd 1 - \frac{1}{2^N} \Rrightarrow [True] \lhd B \rhd 1 - \frac{[True]}{2^N} \end{aligned}$$

Probabilistic Primality cont'd

We carry out the checks for the invariant.

- Check that the invariant is established at initialization:

$$\{1 \lhd B \rhd 1 - \frac{1}{2^N}\}\ b, n := True, N\ \{Inv\}$$

That is we want to check

$$1 \lhd B \rhd 1 - \frac{1}{2^N} \Rrightarrow Inv[b, n := True, N]$$
$$= 1 \lhd B \rhd 1 - \frac{1}{2^N} \Rrightarrow [True] \lhd B \rhd 1 - \frac{[True]}{2^N}$$

- Check for post-condition on termination:
  When $n = 0$ or $\neg b$ holds:

$$1 \lhd B \rhd 1 - \frac{1}{2^N} \Rrightarrow [b] \lhd B \rhd [\neg b]$$

Probabilistic Primality cont'd

- Loop body including decrement in invariant:

$$\{1 \lhd B \rhd 1 - \frac{1}{2^n}\}\ CheckOnce\ \{1 \lhd B \rhd 1 - \frac{1}{2^{n-1}}\}$$

and we assume the truth of the loop guard $\Rightarrow n \neq 0 \wedge b$.

Probabilistic Primality cont'd

- Loop body including decrement in invariant:

$$\{1 \lhd B \rhd 1 - \frac{1}{2^n}\}\ CheckOnce\ \{1 \lhd B \rhd 1 - \frac{1}{2^{n-1}}\}$$

and we assume the truth of the loop guard $\Rightarrow n \neq 0 \wedge b$.

  + When B holds $CheckOnce$ should behave like **skip** and corresponds to the first part of our instantiation:

$$\textbf{if } B \textbf{ then } b := True \textbf{ else} \cdots$$

Probabilistic Primality cont'd

- Loop body including decrement in invariant:

$$\{1 \lhd B \rhd 1 - \frac{1}{2^n}\}\ CheckOnce\ \{1 \lhd B \rhd 1 - \frac{1}{2^{n-1}}\}$$

and we assume the truth of the loop guard $\Rightarrow n \neq 0 \wedge b$.

  + When B holds $CheckOnce$ should behave like **skip** and corresponds to the first part of our instantiation:

$$\textbf{if } B \textbf{ then } b := True \textbf{ else} \cdots$$

  + When B does not hold, we use $b := False$, which makes both expectations 1.
    $\Rightarrow$ Which is also part of our instantiation.

Probabilistic Primality cont'd

The other part is

$$b := False \ {}_{\frac{1}{2}}\!\oplus\ b := True$$

. For this we have an inference rule for probabilistic choice:

Probabilistic Primality cont'd

The other part is

$$b := False \;{}_{\frac{1}{2}}\!\oplus\; b := True$$

. For this we have an inference rule for probabilistic choice:

$$\frac{\begin{array}{lll}\{preExp\} & S & \{postExp\} \\ \{preExp'\} & S' & \{postExp\}\end{array}}{\{p \times preExp + (1-p) \times preExp'\}\ S \;{}_{p}\!\oplus\; S'\ \{postExp\}}$$

Probabilistic Primality cont'd

The other part is

$$b := False \;{}_{\frac{1}{2}}\!\oplus\; b := True$$

. For this we have an inference rule for probabilistic choice:

$$\frac{\begin{array}{lr} \{1\}\; b := False & \{1 - \frac{[b]}{2^{n-1}}\} \\ \{1 - \frac{1}{2^{n-1}}\}\; b := True & \{1 - \frac{[b]}{2^{n-1}}\} \end{array}}{\{\frac{1}{2}1 + \frac{1}{2}(1 - \frac{1}{2^{n-1}})\}\; b := False \;{}_{\frac{1}{2}}\!\oplus\; b := True\; \{1 - \frac{[b]}{2^{n-1}}\}}$$

Probabilistic Primality cont'd

The other part is

$$b := False \ {}_{\frac{1}{2}}\oplus \ b := True$$

. For this we have an inference rule for probabilistic choice:

$$\frac{\{1\}\ b := False\ \{1 - \frac{[b]}{2^{n-1}}\} \qquad \{1 - \frac{1}{2^{n-1}}\}\ b := True\ \{1 - \frac{[b]}{2^{n-1}}\}}{\{\frac{1}{2}1 + \frac{1}{2}(1 - \frac{1}{2^{n-1}})\}\ b := False\ {}_{\frac{1}{2}}\oplus\ b := True\ \{1 - \frac{[b]}{2^{n-1}}\}}$$

We just need to calculate the pre-expectation:

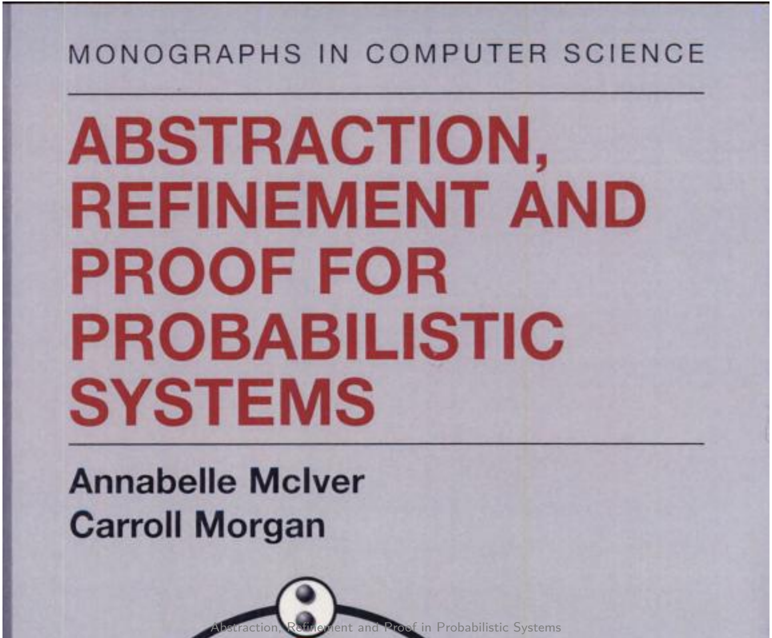
$$\frac{1}{2}1 + \frac{1}{2}(1 - \frac{1}{2^{n-1}}) = 1 - \frac{1}{2^n} = \text{“note that b holds”}\ 1 - \frac{[b]}{2^n}$$

Thus $CheckOnce$ is implemented by our instantiation.

Overview

- Guarded Command Language (GCL)
- Probabilistic Guarded Command Language (pGCL)
- Abstraction and Refinement
- Probably Hoare? Hoare Probably!
- Abstraction Refinement and Proof for Probabilistic Systems

Abstraction Refinement and Proof for Probabilistic Systems



Short Overview of the Book

- Part I: Probabilistic guarded commands: introduction + probabilistic loop invariants and variants

Short Overview of the Book

- Part I: Probabilistic guarded commands: introduction + probabilistic loop invariants and variants
- Part II: Semantic structures: this part develops in detail the mathematics on which the probabilistic logic is built and with which it is justified (correctness).
- Part III: Advanced topics: this part concentrates on more exotic methods of specification and design, in this case probabilistic temporal/modal logics.
- Part IV: Appendices, bibliography and indexes

Summary

- GCL $\Rightarrow$ pGCL
- *wp*-semantics of pGCL
- healthiness properties of pGCL
- probably Hoare semantics vs. Hoare probably semantics

Thank you for your attention!

References

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

C.C. Morgan and A.K. McIver.
pGCL: formal reasoning for random algorithms.
South African Computer Journal, 1999.

C.C. Morgan and A.K. McIver.
Probably Hoare? Hoare probably!
In *Millennial Perspectives in Computer Science, Cornerstones of Computing*, pages 271–282, 2000.

C.C. Morgan and A.K. McIver.
Abstraction, Refinement and Proof for Probabilistic Systems.
Springer, 2004.