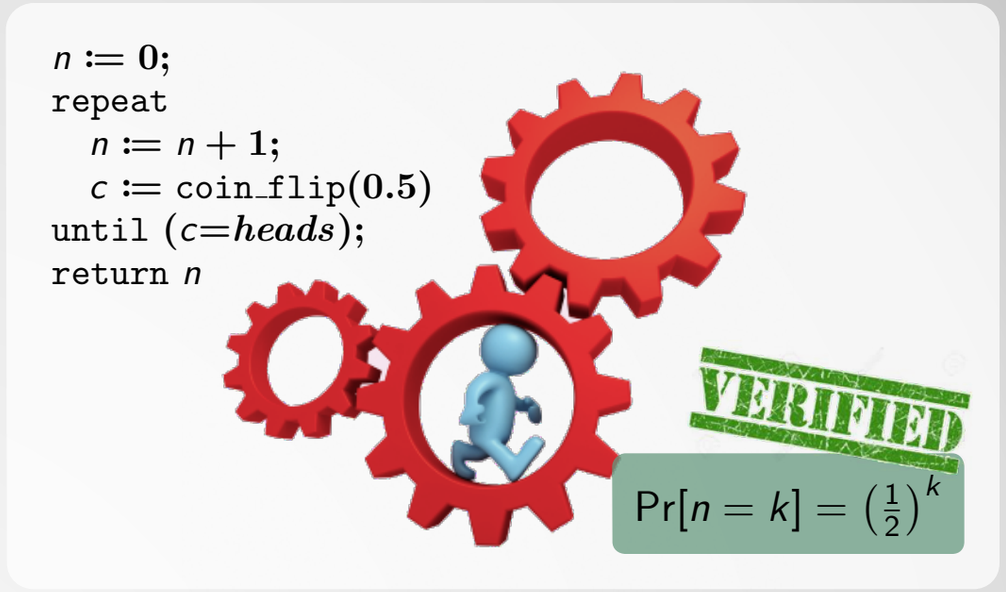
Seminar on

"Verification of Probabilistic Programs"



Lecture 2:

Probabilisitic Predicate Transformers I

Federico Olmedo

Software Modeling and Verification Group
RWTH AACHEN UNIVERSITY

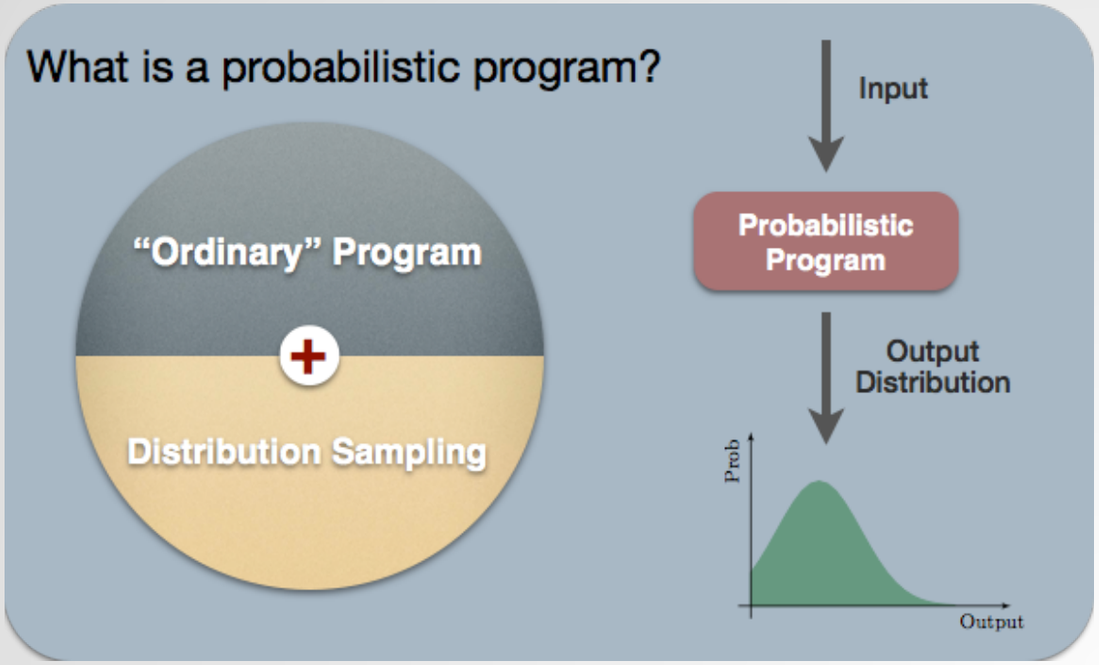
Agenda

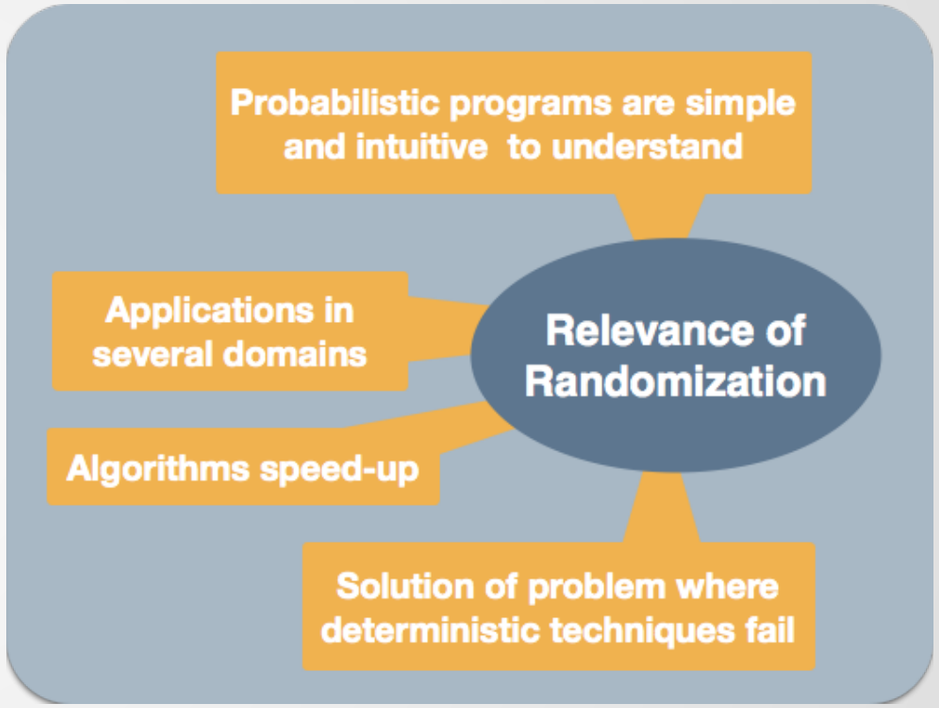
- Recap on previous lecture
- Predicate transformers for deterministic programs
- Predicate transformers for probabilistic programs
- Summary

Agenda

- Recap on previous lecture
- Predicate transformers for deterministic programs
- Predicate transformers for probabilistic programs
- Summary

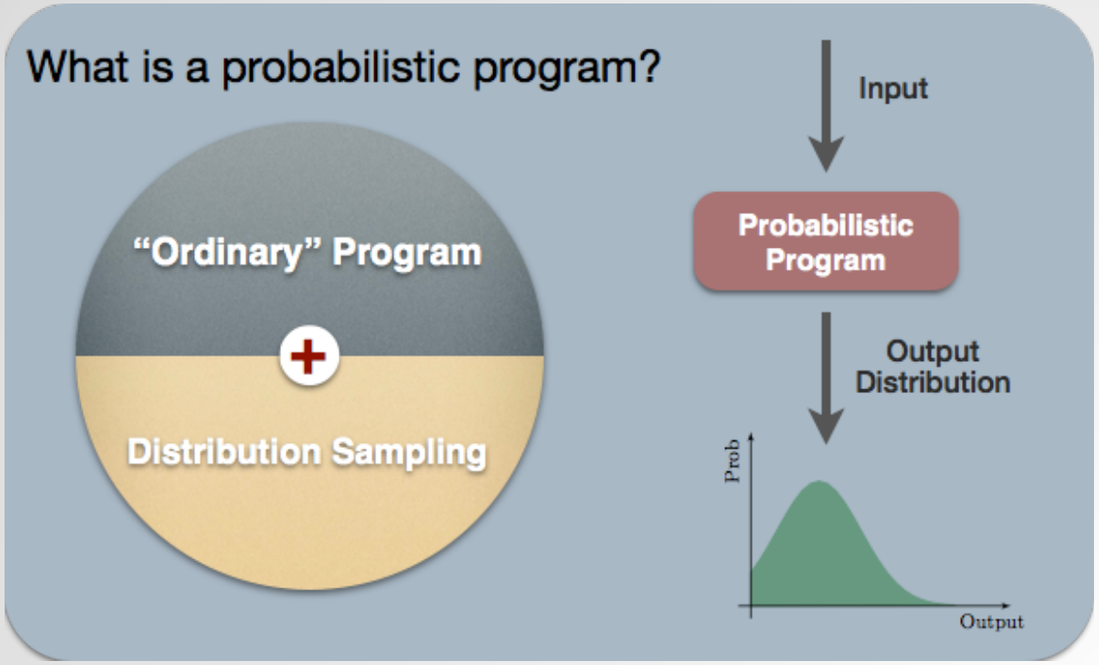
Recap on Previous Lecture





We can extend traditional program verification techniques to probabilistic programs.

Recap on Previous Lecture



We can extend traditional program verification techniques to probabilistic programs.

Probabilistic programs are simple and intuitive to understand

Applications in several domains

Relevance of Randomization

Algorithms speed-up

Solution of problem where deterministic techniques fail

Probabilistic Programs — Examples

```
c1 := coin_flip(0.5);
c2 := coin_flip(0.5);
return (c1, c2)
```

Output Distribution →

		c1	
		h	t
c2	h	1/4	1/4
	t	1/4	1/4

```
n := 0;
repeat
   n := n + 1;
   c := coin_flip(0.5)
until (c=heads);
return n
```

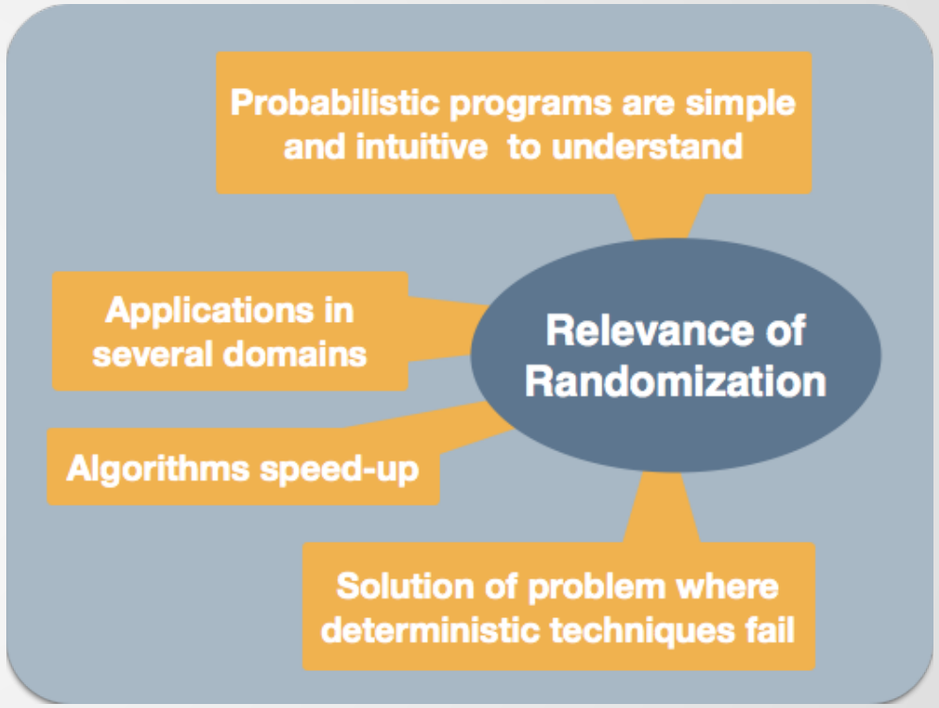
Output Distribution →

n				
1	2	…	k	…
1/2	1/4		$1/2^k$	

Recap on Previous Lecture

What is a probabilistic program?

We can extend traditional program verification techniques to probabilistic programs.



Randomization reduces Time Complexity

Quicksort:

$$
\begin{aligned}
&\mathrm{QS}(A) \triangleq \\
&\quad \texttt{if}\,(|A| \leq 1)\,\texttt{then return}\,(A); \\
&\quad i := \lfloor |A|/2 \rfloor; \\
&\quad A_< := \{a' \in A \mid a' < A[i]\}; \\
&\quad A_> := \{a' \in A \mid a' > A[i]\}; \\
&\quad \texttt{return}\,\left(\mathrm{QS}(A_<) +\!\!+ A[i] +\!\!+ \mathrm{QS}(A_>)\right)
\end{aligned}
$$

Problem of Quicksort:

In the average case, it performs fairly well:

On a random input of size *n*, it requires on average ***O****(n log(n))* comparisons (which matches information theory lower bound*)*.

But in the worst case, it does not:

There exist “ill-behaved” inputs of size *n* which require ***O****(n²)* comparisons.

How to narrow the gap between the worst and average case performance?

Randomization reduces Time Complexity

Quicksort:

$\mathsf{rQS}(A) \triangleq$
  $\texttt{if}\,(|A| \leq 1)\,\texttt{then return}\,(A);$
  $i := \mathsf{rand}[1 \ldots |A|];$
  $A_< := \{a' \in A \mid a' < A[i]\};$
  $A_> := \{a' \in A \mid a' > A[i]\};$
  $\texttt{return}\,\big(\mathsf{QS}(A_<) +\!\!+ A[i] +\!\!+ \mathsf{QS}(A_>)\big)$

Problem of Quicksort:

In the average case, it performs fairly well:

On a random input of size *n*, it requires on average ***O****(n log(n))* comparisons (which matches information theory lower bound*)*.

But in the worst case, it does not:

There exist “ill-behaved” inputs of size *n* which require ***O****(n²)* comparisons.

How to narrow the gap between the worst and average case performance?

Choose the pivot at random!

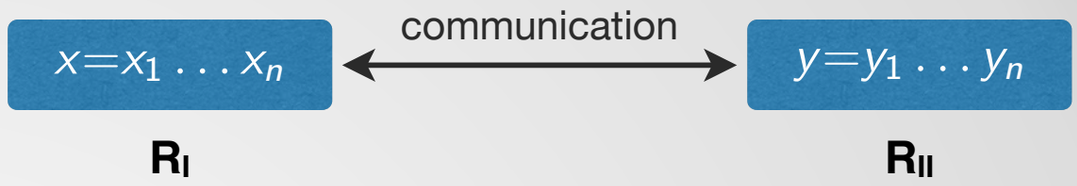
For **any** input, the expected number of comparisons matches the average case.

No “ill-behaved” input.

Randomization reduces Communication Complexity

Verifying Data Consistency

- Goal: R_I and R_{II} must communicate to verify whether $x=y$ $(x, y \in \{0,1\}^n)$.
- Requirement: minimize the # of bits exchanged.



Theorem: any deterministic protocol requires the exchange of (at least) n bits.

Randomized Algorithm

Idea: use random fingerprints of x and y.

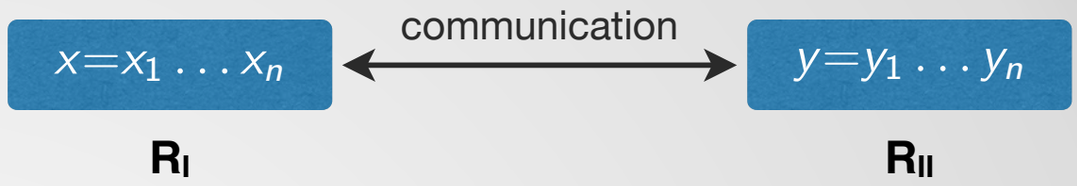
Routine of $\mathrm{R_I} \triangleq$
  $p :=$ rand$\{i \in [2, n^2] \mid prime(i)\}$;
  $s := x \bmod p$;
  `send`(p, s) `to` $\mathrm{R_{II}}$;

Routine of $\mathrm{R_{II}} \triangleq$
  `receive`(p, s) `from` $\mathrm{R_I}$;
  $t := y \bmod p$;
  `if` $(s=t)$ `then return` $(\text{“}x=y\text{”})$
    `else return` $(\text{“}x \neq y\text{”})$

Randomization reduces Communication Complexity

Verifying Data Consistency

- Goal: R_I and R_{II} must communicate to verify whether $x=y$ $(x, y \in \{0,1\}^n)$.
- Requirement: minimize the # of bits exchanged.



Theorem: any deterministic protocol requires the exchange of (at least) n bits.

Randomized Algorithm

Idea: use random fingerprints of x and y.

Routine of $\mathrm{R_I} \triangleq$
  $p :=$ rand$\{i \in [2, n^2] \mid prime(i)\}$;
  $s := x \bmod p$;
  send(p, s) to $\mathrm{R_{II}}$;

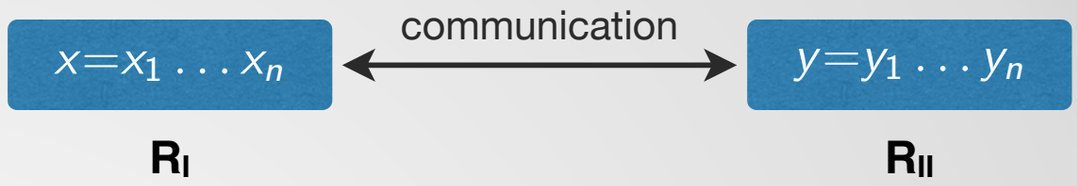
Routine of $\mathrm{R_{II}} \triangleq$
  receive(p, s) from $\mathrm{R_I}$;
  $t := y \bmod p$;
  if $(s=t)$ then return ("$x=y$")
    else return ("$x \neq y$")

	# bits exch.
$n=10^{10}$	133
$n=10^{20}$	266
$n=10^{30}$	398
$n=10^{40}$	532
$n=10^{50}$	664

Randomization reduces Communication Complexity

Verifying Data Consistency

- Goal: R_I and R_{II} must communicate to verify whether $x=y$ $(x, y \in \{0,1\}^n)$.
- Requirement: minimize the # of bits exchanged.



Theorem: any deterministic protocol requires the exchange of (at least) n bits.

Randomized Algorithm

Idea: use random fingerprints of x and y.

Routine of $R_I \triangleq$
  $p :=$ rand$\{i \in [2, n^2] \mid prime(i)\}$;
  $s := x \bmod p$;
  send(p, s) to R_{II};

Routine of $R_{II} \triangleq$
  receive(p, s) from R_I;
  $t := y \bmod p$;
  if $(s=t)$ then return $(\text{“}x=y\text{”})$
    else return $(\text{“}x \neq y\text{”})$

Prot. outputs “$x \neq y$” ➡ $x \neq y$

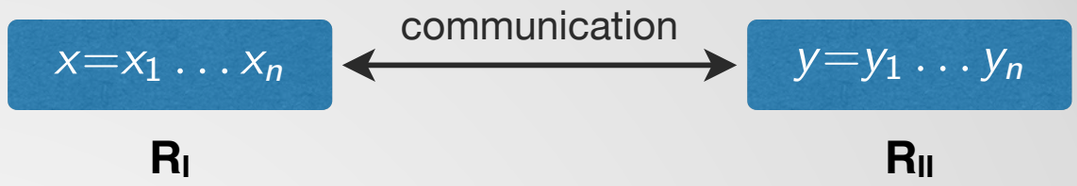
$$\Pr[x \neq y \mid \text{output “}x=y\text{”}] \leq \frac{\ln(n^2)}{n}$$

	# bits exch.
$n=10^{10}$	133
$n=10^{20}$	266
$n=10^{30}$	398
$n=10^{40}$	532
$n=10^{50}$	664

Randomization reduces Communication Complexity

Verifying Data Consistency

- Goal: R_I and R_{II} must communicate to verify whether $x=y$ $(x, y \in \{0,1\}^n)$.
- Requirement: minimize the # of bits exchanged.



Theorem: any deterministic protocol requires the exchange of (at least) n bits.

Randomized Algorithm

Idea: use random fingerprints of x and y.

Routine of $R_I \triangleq$
  $p :=$ rand$\{i \in [2, n^2] \mid prime(i)\}$;
  $s := x$ mod p;
  send(p, s) to R_{II};

Routine of $R_{II} \triangleq$
  receive(p, s) from R_I;
  $t := y$ mod p;
  if $(s=t)$ then return ("$x=y$")
    else return ("$x \neq y$")

Prot. outputs "$x \neq y$" ➡ $x \neq y$

$$\Pr[x \neq y \mid \text{output } \text{"}x=y\text{"}] \leq \frac{\ln(n^2)}{n}$$

	# bits exch.	prob. error
$n=10^{10}$	133	4.60×10^{-09}
$n=10^{20}$	266	9.21×10^{-19}
$n=10^{30}$	398	1.38×10^{-28}
$n=10^{40}$	532	1.84×10^{-38}
$n=10^{50}$	664	2.30×10^{-48}

Randomization circumvents the Limitations of Determinism

The Dinning Philosopher Problem

Theorem (Lehmann & Rabin '81) there exists no fully distributed and symmetric deterministic algorithm for the dining philosopher problem.

Randomized Algorithm

```
while (true) do
   (* Thinking Time *)
   trying := true
   while (trying) do
      s := rand{left, right}
      Wait until fork[s] is available and take it
      If fork[¬s] is available
                then take it and set trying to false
                else drop fork[s]
   (* Eating Time *)
   Drop both forks
```

Idea:

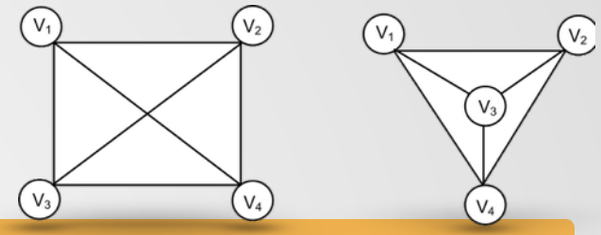
- Do not pick always the same fork first. Flip a coin to choose.
- If the second fork is not available, release the first and flip again the coin.

Algorithm is deadlock-free:

At any time, if there is a hungry philosopher, with probability one some philosopher will eventually eat. Algorithm can also be adapted to prevent starvation (ie the hungry philosopher will eventually eat).

Randomization has Countless Application Domains

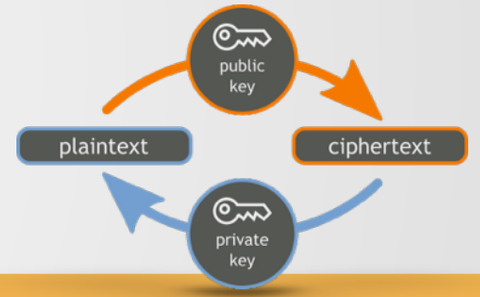
Skill Ranking System



Graph Isomorphism

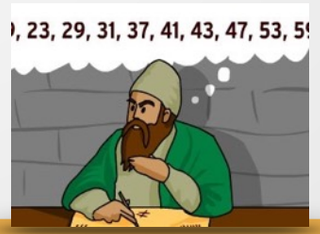
Probabilistic Programs

Predator—Prey Population Model



Public-Key Encryption

Dining Philosophers



Miller—Rabin Primality Test

Recap on Previous Lecture

We can extend traditional program verification techniques to probabilistic programs.

- Hoare logic
- Weakest pre-condition calculus

Recap on Previous Lecture

We can extend traditional program verification techniques to probabilistic programs.

Today

- Weakest pre-condition calculus

Agenda

- Recap on previous lecture
- Predicate transformers for deterministic programs
- Predicate transformers for probabilistic programs
- Summary

Predicate Transformers — Intuition

What must be true at the beginning (ie in the initial state) so that a given assertion holds at the end (ie in the final state) of a program execution?

Examples

- For which initial states does program $a := a+1;\ b := b-1$ lead to a final state where $a \cdot b = 0$?
  For those initial states where $a = -1 \vee b = 1$.
- For which initial states does program $\texttt{while}\ (i \neq 0)\ \texttt{do}\ i := i-1$ terminate?
  For those initial states where $i \geq 0$.

Predicate Transformers — Intuition

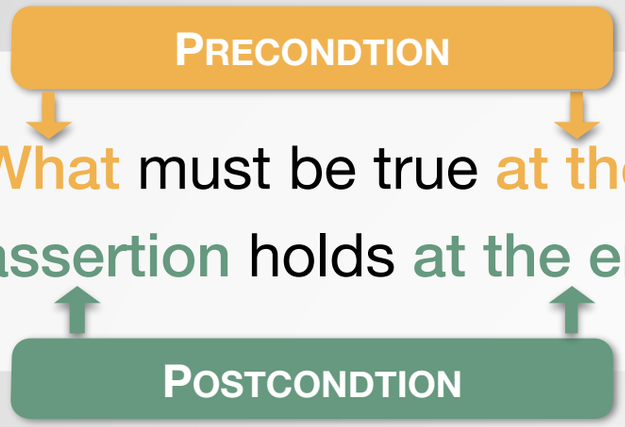
What must be true at the beginning (ie in the initial state) so that a given assertion holds at the end (ie in the final state) of a program execution?

Examples

- For which initial states does program $a := a+1;\ b := b-1$ lead to a final state where $a \cdot b = 0$?
  For those initial states where $a = -1 \vee b = 1$.

- For which initial states does program $\texttt{while}\ (i \neq 0)\ \texttt{do}\ i := i-1$ terminate?
  For those initial states where $i \geq 0$.
  Answer e.g. $i \geq 5$ is not right (it is sufficient, but **not necessary** for ensuring termination).

Predicate Transformers — Intuition



Examples

- For which initial states does program $a := a{+}1;\ b := b{-}1$ lead to a final state where $a \cdot b = 0$?
  For those initial states where $a{=}-1 \vee b{=}1$.

- For which initial states does program $\texttt{while}\ (i{\neq}0)\ \texttt{do}\ i := i{-}1$ terminate?
  For those initial states where $i{\geq}0$.
  Answer e.g. $i{\geq}5$ is not right (it is sufficient, but **not necessary** for ensuring termination).

Predicate Transformers — Intuition

PRECONDTION

What must be true at the beginning (ie in the initial state) so that a given assertion holds at the end (ie in the final state) of a program execution?

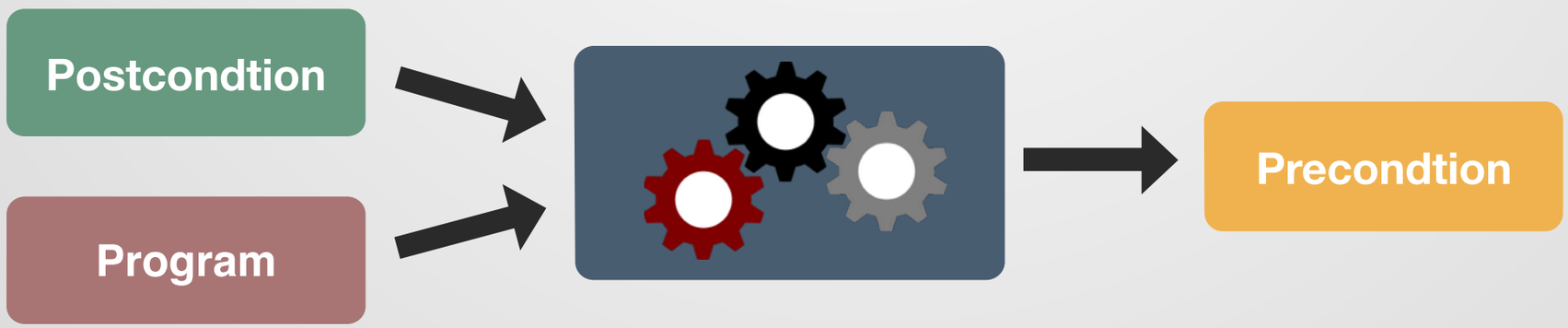
POSTCONDTION

Examples

- For which initial states does program $a := a{+}1;\ b := b{-}1$ lead to a final state where $a \cdot b = 0$?
  For those initial states where $a{=}-1 \vee b{=}1$.

- For which initial states does program $\texttt{while}\ (i{\neq}0)\ \texttt{do}\ i := i{-}1$ terminate?
  For those initial states where $i{\geq}0$.
  Answer e.g. $i{\geq}5$ is not right (it is sufficient, but **not necessary** for ensuring termination).

Our verification tool



Predicate Transformers — Intuition

PRECONDTION

What must be true at the beginning (ie in the initial state) so that a given assertion holds at the end (ie in the final state) of a program execution?

POSTCONDTION

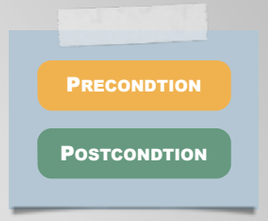
Examples

- For which initial states does program $a := a{+}1;\ b := b{-}1$ lead to a final state where $a \cdot b = 0$?
  For those initial states where $a{=}-1 \vee b{=}1$.

- For which initial states does program $\texttt{while}\ (i{\neq}0)\ \texttt{do}\ i := i{-}1$ terminate?
  For those initial states where $i{\geq}0$.
  Answer e.g. $i{\geq}5$ is not right (it is sufficient, but **not necessary** for ensuring termination).

Our verification tool

Predicate Transformers — Intuition



- *Program states* will be variable valuation, i.e. $\mathcal{S} \triangleq \mathsf{Var} \to \mathsf{Val}$.
- *Pre* and *postconditions* are predicates over program states, or equivalently, set of program states.
- For each program c, *predicate transformer*

$$\mathsf{wp}[c] : \mathcal{P}(\mathcal{S}) \to \mathcal{P}(\mathcal{S})$$

transforms postconditions into preconditions.

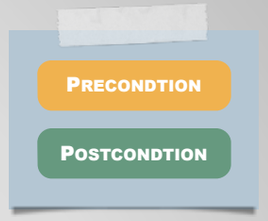
Examples

$$\mathsf{wp}[a \coloneqq a{+}1;\ b \coloneqq b{-}1](a \cdot b{=}0) \ = \ a{=}-1 \lor b{=}1$$

$$\mathsf{wp}[\texttt{while}\ (i{\neq}0)\ \texttt{do}\ i \coloneqq i{-}1](\mathsf{true}) \ = \ i{\geq}0$$

$$\mathsf{wp}[\texttt{if}\ (x{\geq}0)\ \texttt{then}\ \{y \coloneqq z{+}1\}\ \texttt{else}\ \{y \coloneqq z{+}2\}](y{\geq}0)$$

Predicate Transformers — Intuition



- *Program states* will be variable valuation, i.e. $\mathcal{S} \triangleq \mathsf{Var} \to \mathsf{Val}$.
- *Pre* and *postconditions* are predicates over program states, or equivalently, set of program states.
- For each program c, *predicate transformer*

$$\mathsf{wp}[c] : \mathcal{P}(\mathcal{S}) \to \mathcal{P}(\mathcal{S})$$

transforms postconditions into preconditions.

Examples

$$\mathsf{wp}[a := a{+}1;\ b := b{-}1](a \cdot b{=}0) \ = \ a{=}-1 \lor b{=}1$$

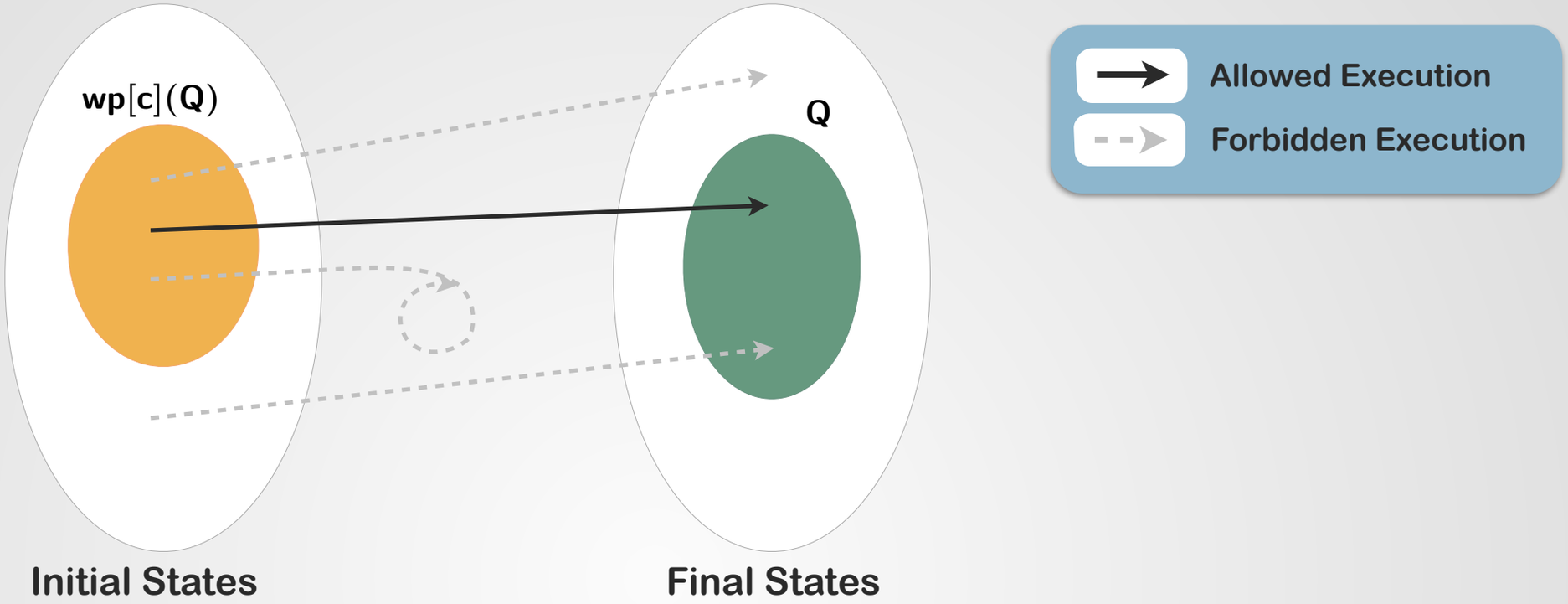
$$\mathsf{wp}[\texttt{while}\ (i{\neq}0)\ \texttt{do}\ i := i{-}1](\mathsf{true}) \ = \ i{\geq}0$$

$$\mathsf{wp}[\texttt{if}\ (x{\geq}0)\ \texttt{then}\ \{y := z{+}1\}\ \texttt{else}\ \{y := z{+}2\}](y{\geq}0)$$
$$= \ (x{\geq}0 \implies z{+}1{\geq}0) \land (x{<}0 \implies z{+}2{\geq}0)$$

Predicate Transformers — Intuition

Precondtion

Postcondtion



Allowed Execution

Forbidden Execution

(1) Every execution from ● terminates.

(2) Every execution from ● lands on ●.

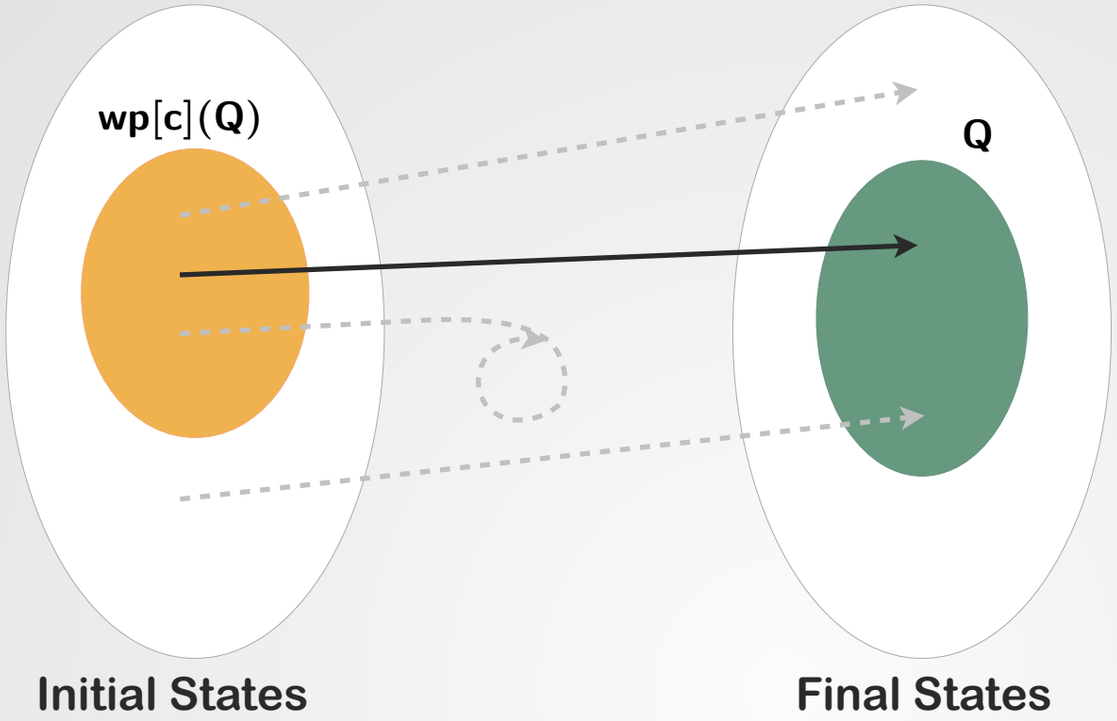
(3) No execution from ¬● lands on ●.

wp[c](Q) is the **weakest** precondition that guarantees Q: if initial state *s* leads to a final state in *Q*, then *s* **must** satisfy *wp[c](Q)*.

Predicate Transformers — Intuition

Precondtion

Postcondtion



Allowed Execution

Forbidden Execution

(1) Every execution from ● terminates.

(2) Every execution from ● lands on ●.

(3) No execution from ¬● lands on ●.

Summary

wp[c](Q) gives the **necessary** (3) **and sufficient** (1-2) condition for guaranteeing *Q*.

wp[c](Q) is the **weakest** precondition that guarantees Q: if initial state *s* leads to a final state in *Q*, then *s* **must** satisfy *wp[c](Q)*.

Predicate Transformers — Formal Definition

The GCL language

$$
\begin{array}{rcll}
\mathcal{C} & := & \texttt{skip} & \text{nop} \\
& | & \texttt{abort} & \text{abortion} \\
& | & x := E & \text{assignment} \\
& | & \texttt{if } G \texttt{ then } \mathcal{C} \texttt{ else } \mathcal{C} & \text{conditional} \\
& | & \texttt{while } G \texttt{ do } \mathcal{C} & \text{while loop} \\
& | & \mathcal{C}; \mathcal{C} & \text{sequence}
\end{array}
$$

Theorem (Dijkstra '75)

For **GCL** program *c*, predicate transformer *wp[c]* can be defined by induction on *c* structure.

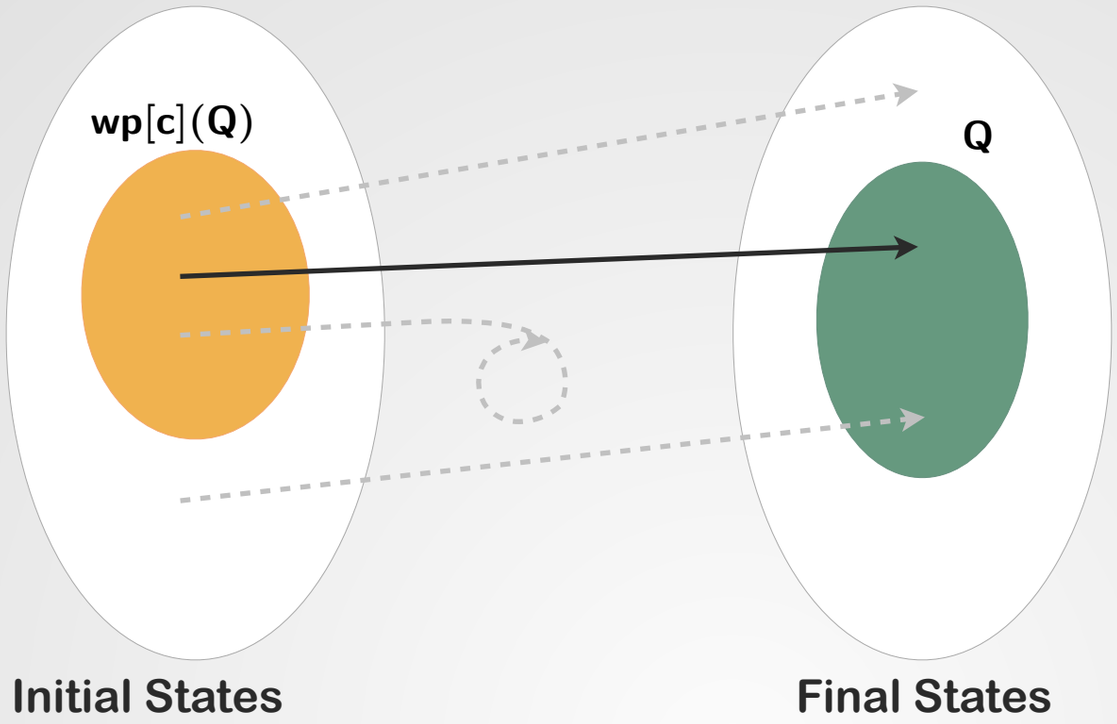
$$
\begin{aligned}
\mathrm{wp}[\texttt{skip}](Q) &= Q \\
\mathrm{wp}[\texttt{abort}](Q) &= \mathsf{false} \\
\mathrm{wp}[x := E](Q) &= Q[E/x] \\
\mathrm{wp}[\texttt{if } G \texttt{ then } c_1 \texttt{ else } c_2](Q) &= (G \Rightarrow \mathrm{wp}[c_1](Q)) \wedge (\neg G \Rightarrow \mathrm{wp}[c_2](Q)) \\
\mathrm{wp}[c_1; c_2](Q) &= (\mathrm{wp}[c_1] \circ \mathrm{wp}[c_2])\,(Q) \\
\mathrm{wp}[\texttt{while } G \texttt{ do } c](Q) &= \mu P \bullet ((G \Rightarrow \mathrm{wp}[c](P)) \wedge (\neg G \Rightarrow Q))
\end{aligned}
$$

Predicate Transformers — Application Example

$$
\begin{aligned}
\mathsf{wp}[x := E](Q) &= Q[E/x] \\
\mathsf{wp}[\texttt{if } G \texttt{ then } c_1 \texttt{ else } c_2](Q) &= (G \Rightarrow \mathsf{wp}[c_1](Q)) \wedge (\neg G \Rightarrow \mathsf{wp}[c_2](Q))
\end{aligned}
$$

$$
\begin{aligned}
& \mathsf{wp}[\texttt{if } (x{\geq}0) \texttt{ then } \{y := z{+}1\} \texttt{ else } \{y := z{+}2\}](y{\geq}0) \\
= \quad & \langle \text{rule for conditionals} \rangle \\
& (x{\geq}0 \implies \mathsf{wp}[y := z{+}1](y{\geq}0)) \wedge (x{<}0 \implies \mathsf{wp}[y := z{+}2](y{\geq}0)) \\
= \quad & \langle \text{rule for assignments, twice} \rangle \\
& (x{\geq}0 \implies z{+}1{\geq}0)) \wedge (x{<}0 \implies z{+}2{\geq}0)) \\
= \quad & \langle \text{simplification} \rangle \\
& (x{\geq}0 \wedge z{\geq}{-}1)) \vee (x{<}0 \wedge z{\geq}{-}2))
\end{aligned}
$$

Predicate Transformers — The Liberal Version



Precondtion

Postcondtion

Allowed Execution

Forbidden Execution

~~(1) Every execution from ● terminates.~~

(2) Every execution from ● lands on ●.

(3) No execution from ¬● lands on ●.

(4) No execution from ¬● diverges.

Summary

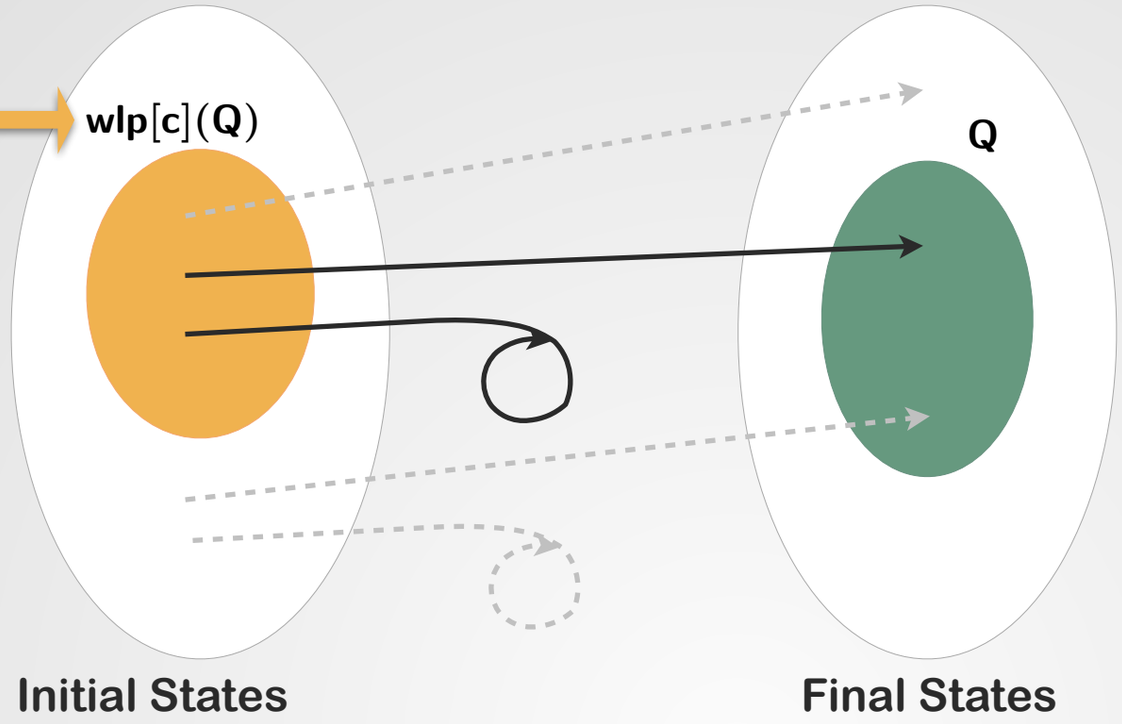
wp[c](Q) gives the necessary (3) and sufficient (1-2) condition for establishing *Q*.

Predicate Transformers — The Liberal Version

PRECONDTION

POSTCONDTION

WEAKEST LIBERAL PRECONDITION



Allowed Execution

Forbidden Execution

~~(1) Every execution from terminates.~~

(2) Every execution from lands on .

(3) No execution from ¬ lands on .

(4) No execution from ¬ diverges.

Summary

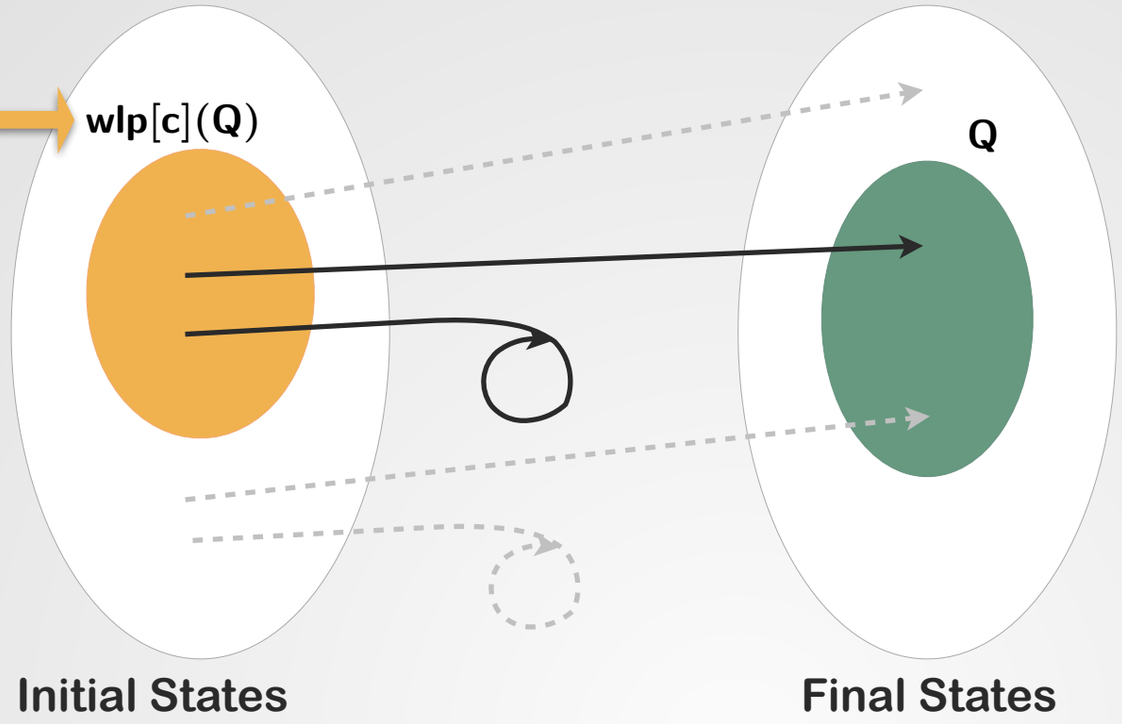
wlp[c](Q) gives the **necessary** (3) **and sufficient** (2) condition for establishing *Q or c* divergence.

Predicate Transformers — The Liberal Version

PRECONDTION

POSTCONDTION

WEAKEST LIBERAL PRECONDITION



Allowed Execution

Forbidden Execution

~~(1) Every execution from ● terminates.~~

(2) Every execution from ● lands on ●.

(3) No execution from ¬● lands on ●.

(4) No execution from ¬● diverges.

Summary

wlp[c](Q) gives the necessary (3) and sufficient (2) condition for establishing *Q or c* divergence.

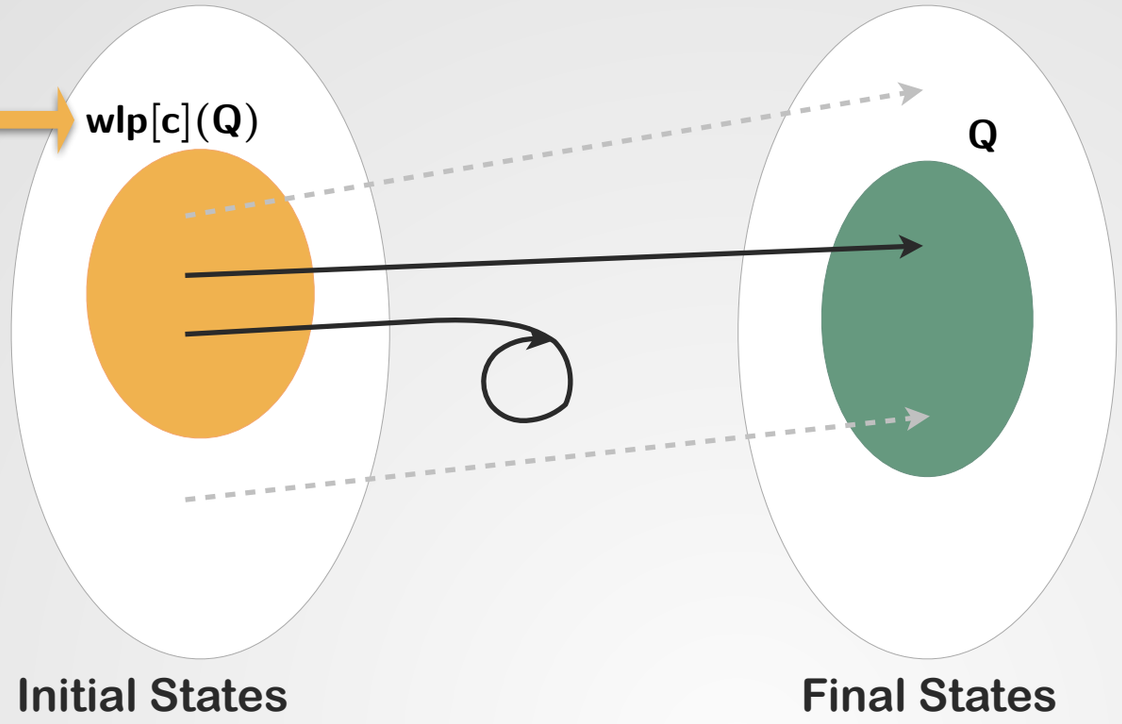
Connection between predicate transformers

Predicate Transformers — The Liberal Version

PRECONDTION

POSTCONDTION

WEAKEST LIBERAL PRECONDITION



Allowed Execution

Forbidden Execution

~~(1) Every execution from • terminates.~~

(2) Every execution from • lands on •.

(3) No execution from ¬• lands on •.

(4) No execution from ¬• diverges.

Summary

wlp[c](Q) gives the **necessary** (3) **and sufficient** (2) condition for establishing *Q or c* divergence.

Connection between predicate transformers

$$\mathsf{wp}[c](Q) \;=\; \mathsf{wlp}[c](Q) \wedge \mathsf{wp}[c](\mathsf{true})$$

Predicate Transformers — The Liberal Version

Only the rules for `abort` and `while`—loops change

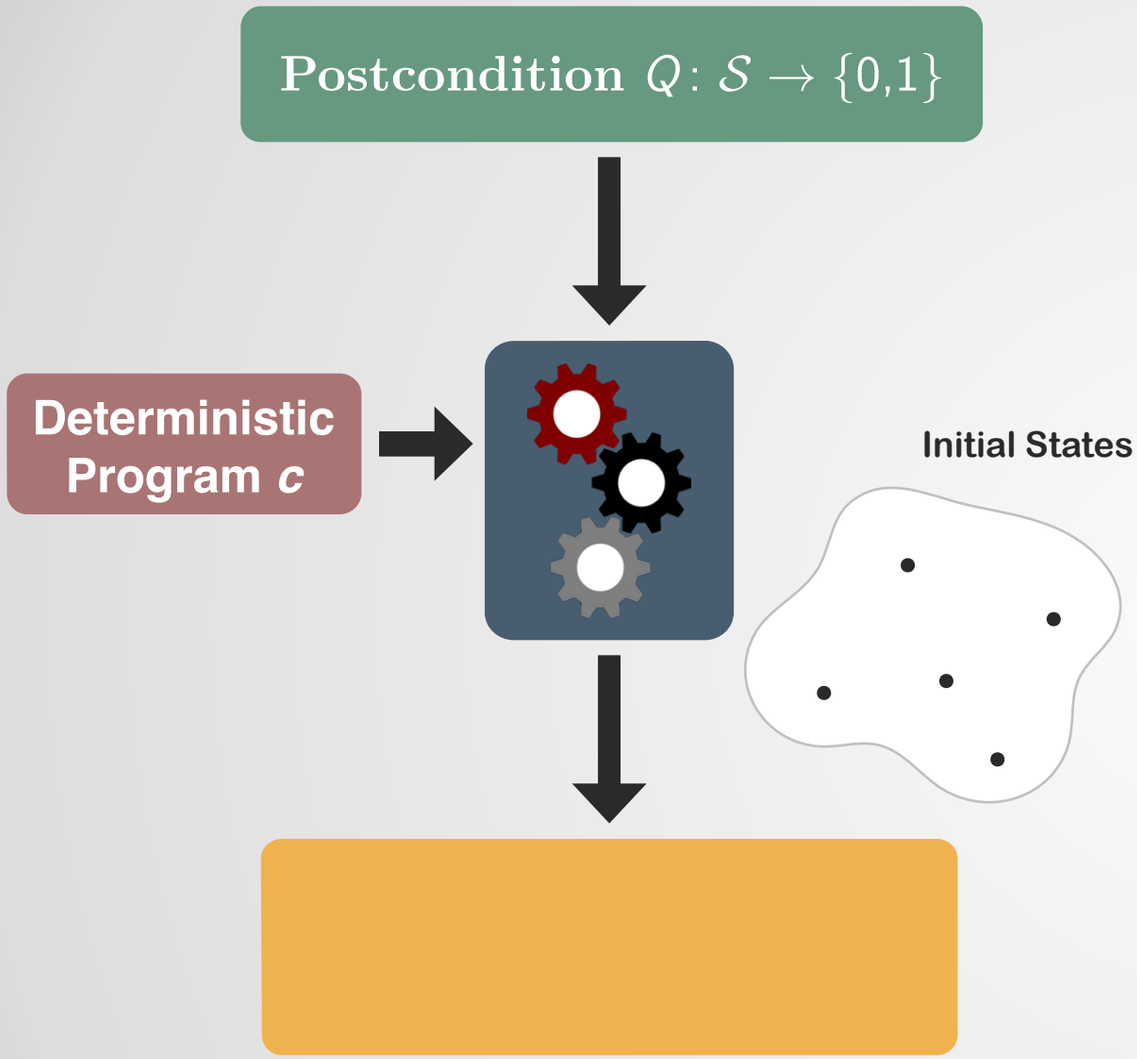
$$
\begin{aligned}
\mathrm{wlp}[\texttt{abort}](Q) &= \underline{\mathrm{true}} \\
\mathrm{wlp}[\texttt{while}\ G\ \texttt{do}\ c](Q) &= \nu P \bullet ((G \Rightarrow \mathrm{wlp}[c](P)) \wedge (\neg G \Rightarrow Q))
\end{aligned}
$$

Agenda

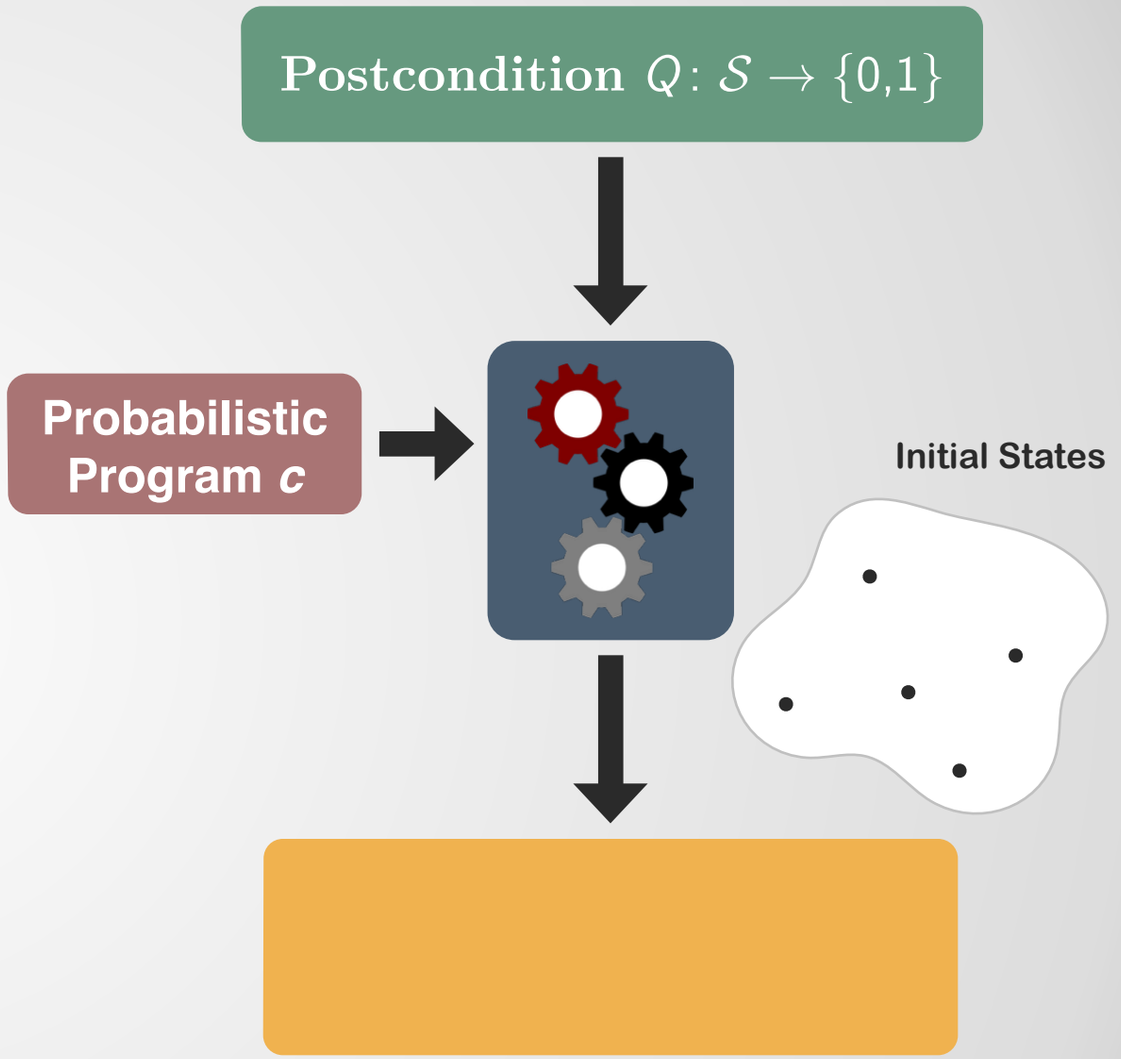
- Recap on previous lecture
- Predicate transformers for deterministic programs
- Predicate transformers for probabilistic programs
- Summary

The Program Verification Problem — Deterministic vs Probabilistic Setting

Deterministic Setting

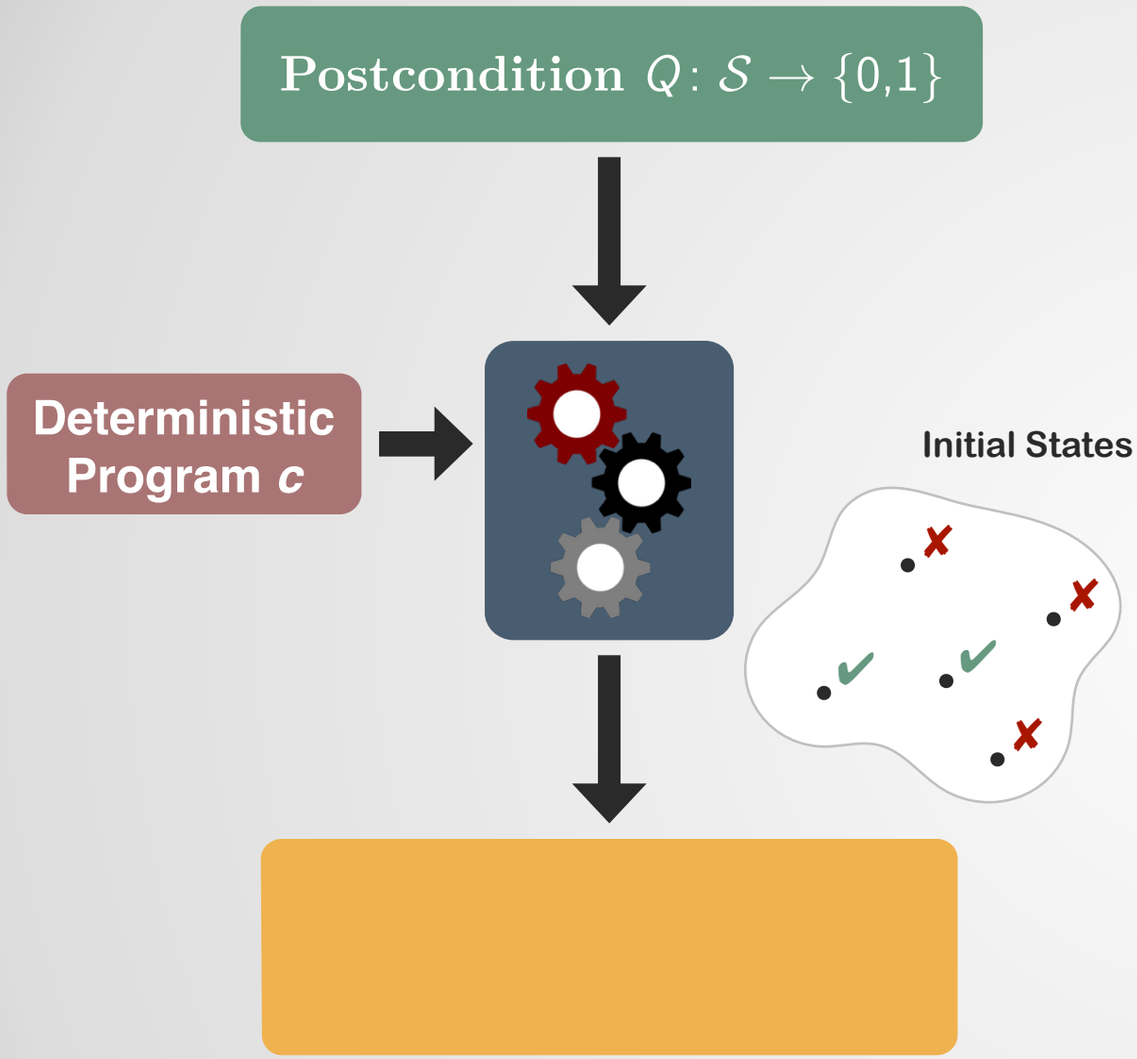


Probabilistic Setting

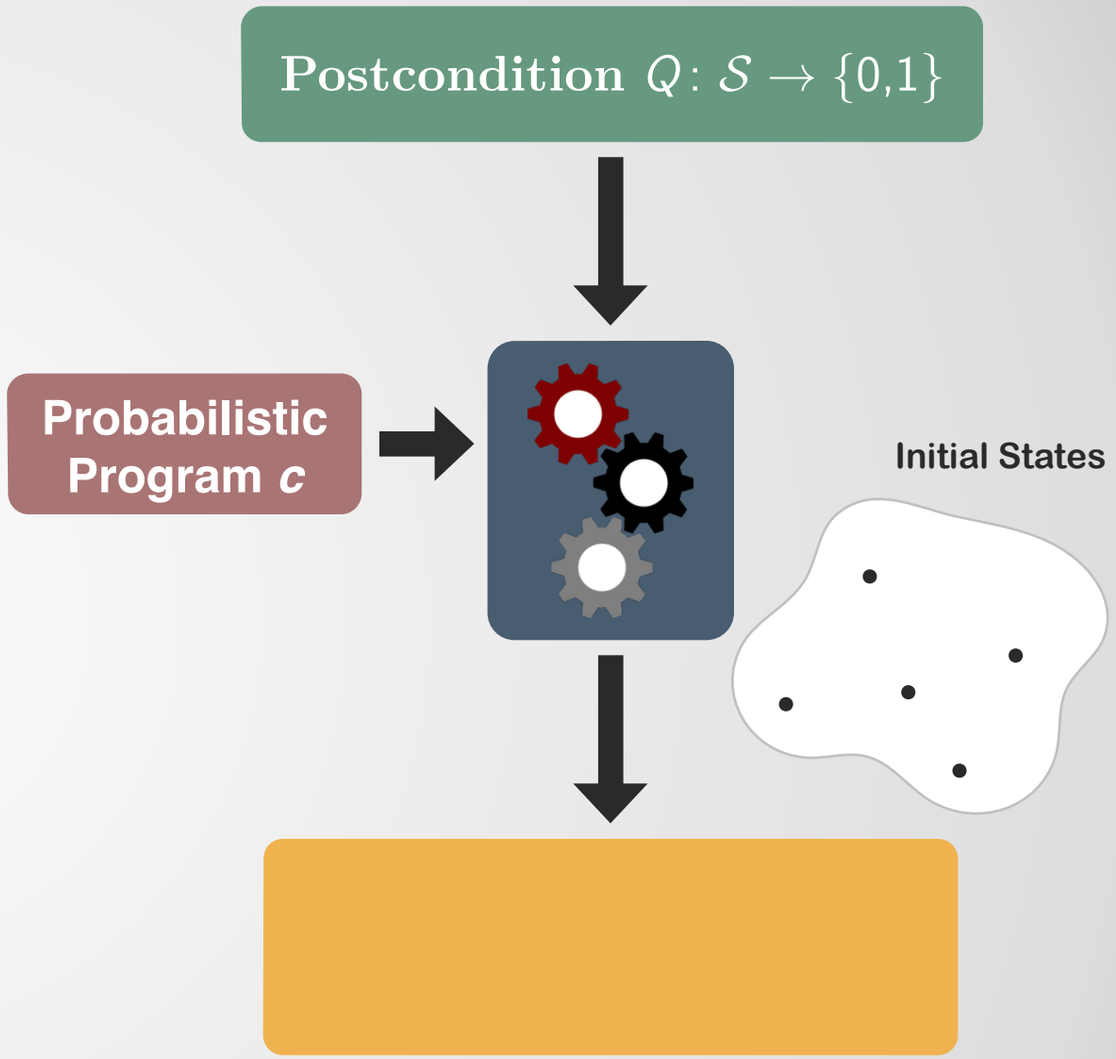


The Program Verification Problem — Deterministic vs Probabilistic Setting

Deterministic Setting

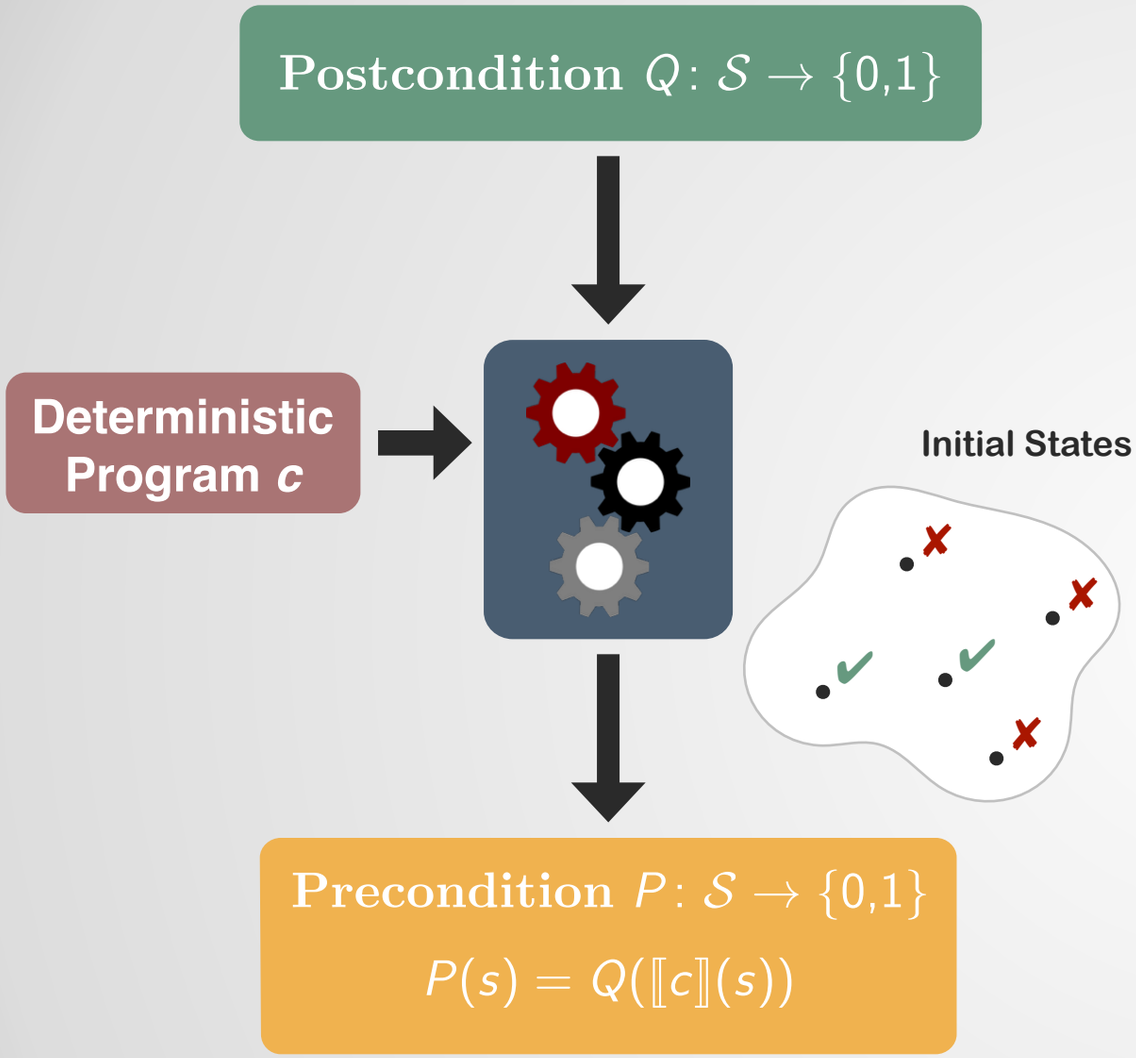


Probabilistic Setting

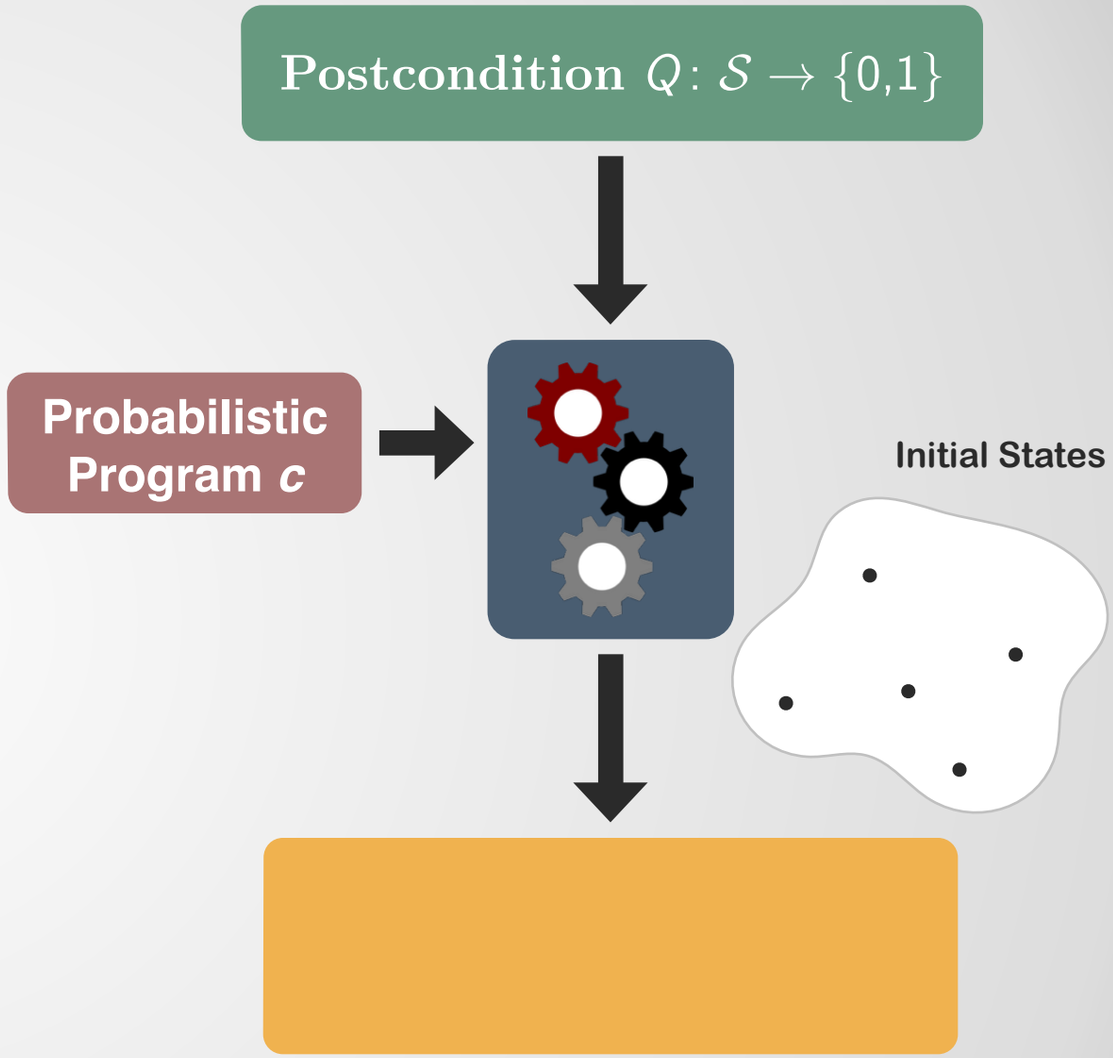


The Program Verification Problem — Deterministic vs Probabilistic Setting

Deterministic Setting



Probabilistic Setting

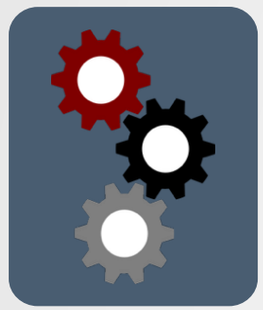


The Program Verification Problem — Deterministic vs Probabilistic Setting

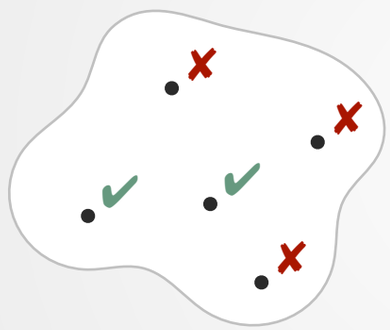
Deterministic Setting

Postcondition $Q\colon \mathcal{S} \to \{0,1\}$

Deterministic Program *c*



Initial States



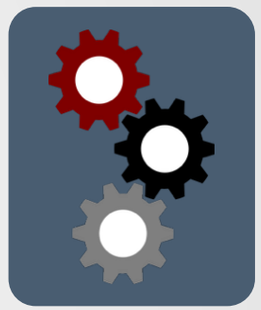
Precondition $P\colon \mathcal{S} \to \{0,1\}$

$P(s) = Q([\![c]\!](s))$

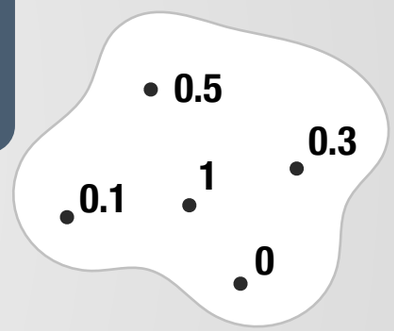
Probabilistic Setting

Postcondition $Q\colon \mathcal{S} \to \{0,1\}$

Probabilistic Program *c*

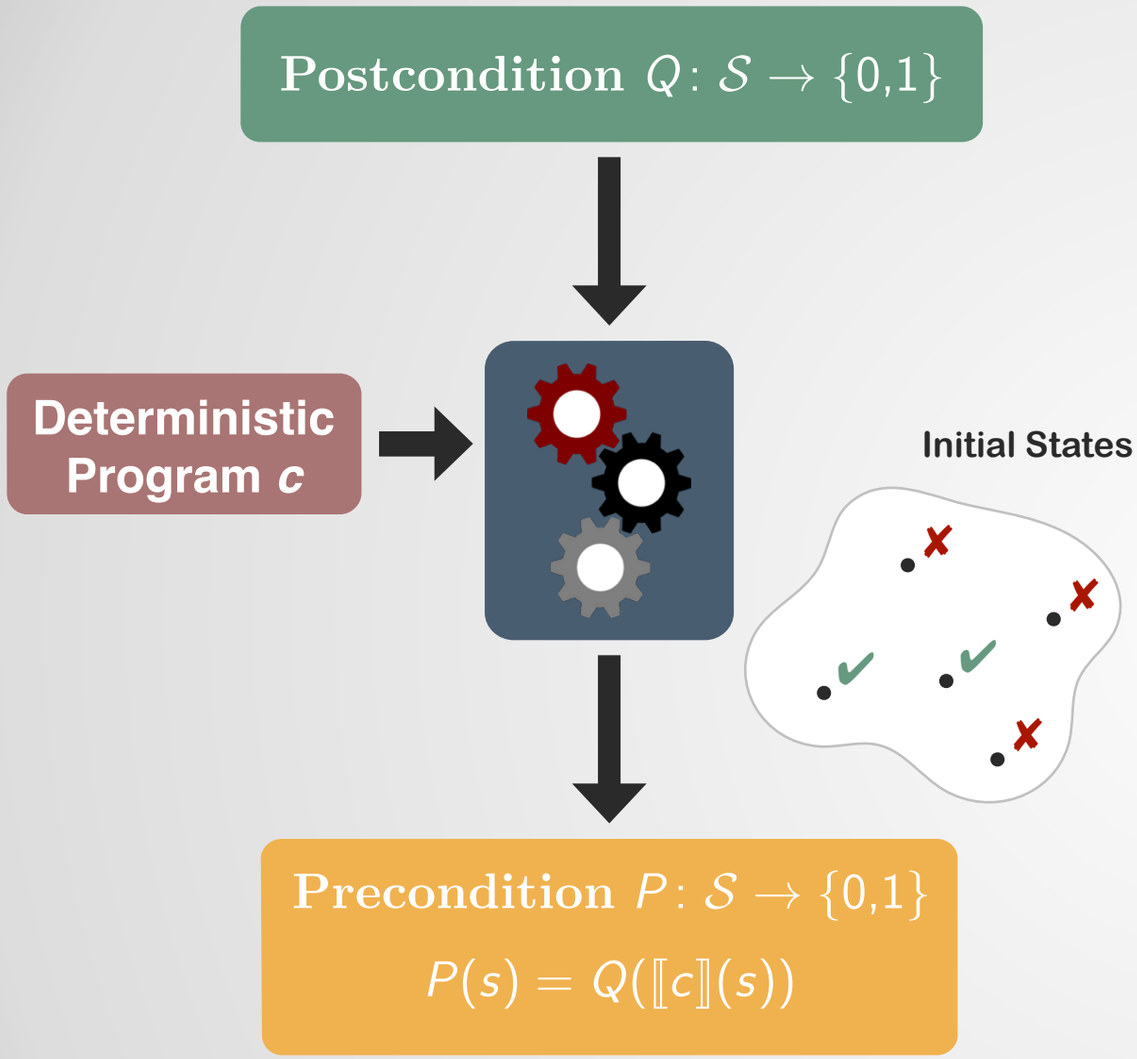


Initial States

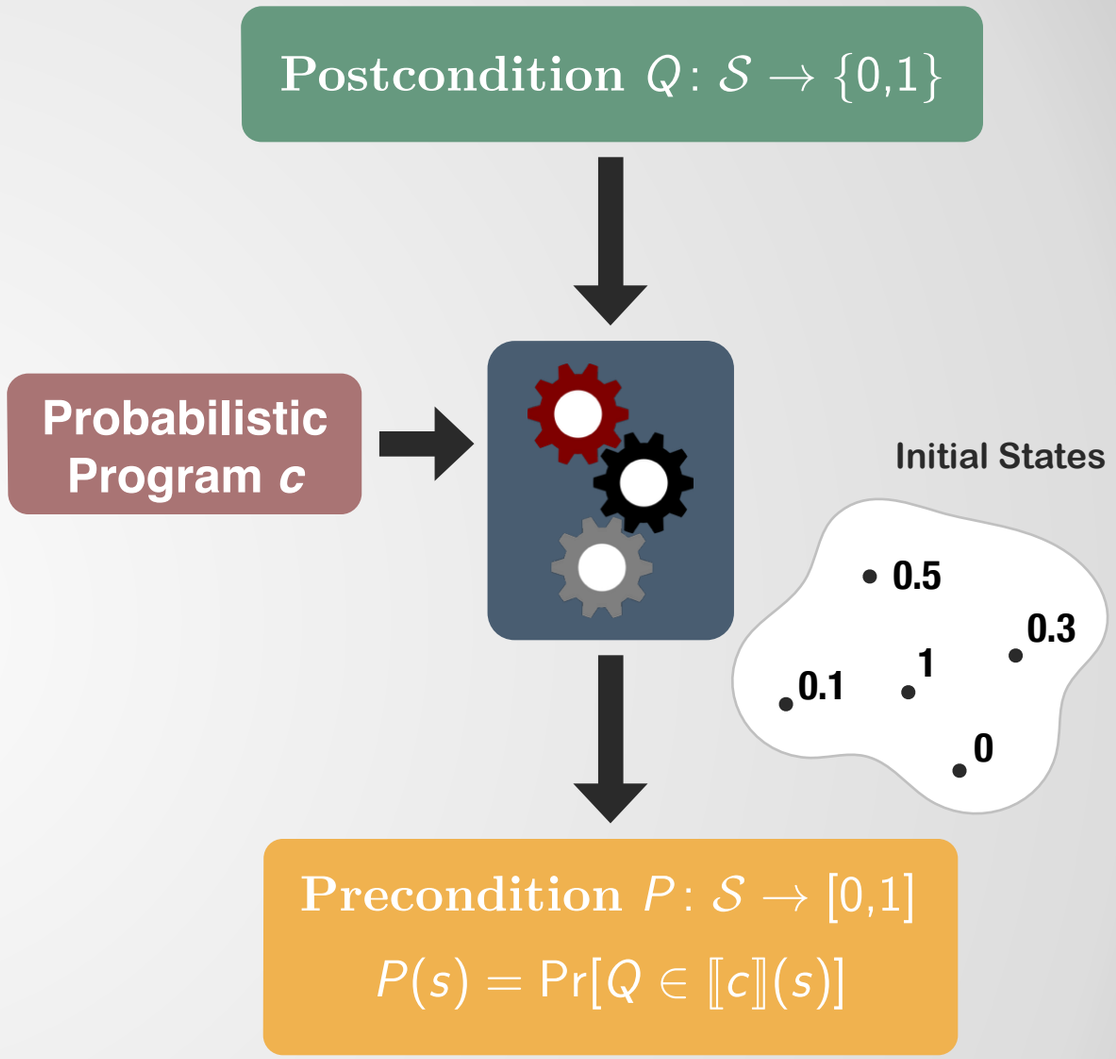


The Program Verification Problem — Deterministic vs Probabilistic Setting

Deterministic Setting

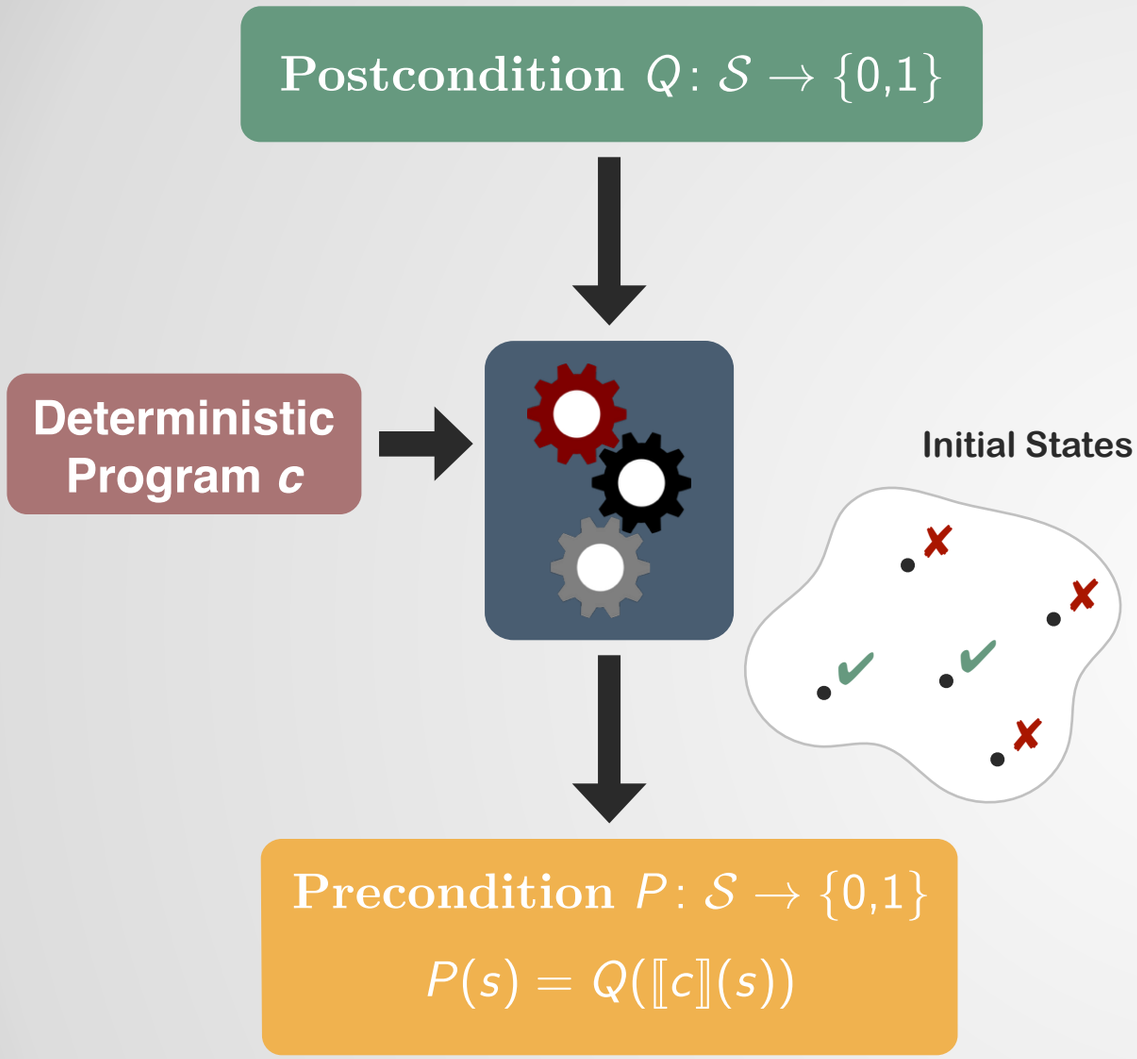


Probabilistic Setting

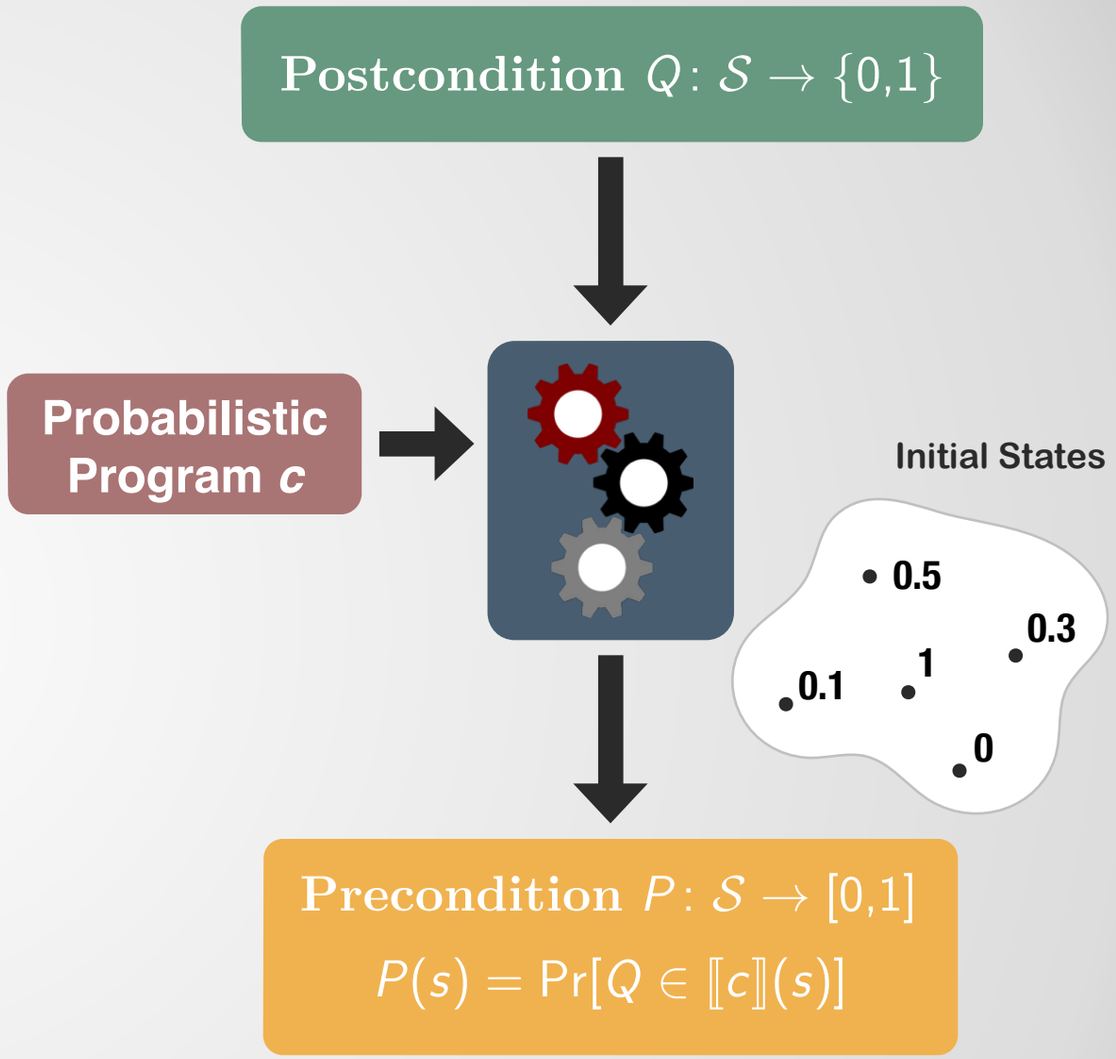


The Program Verification Problem — Deterministic vs Probabilistic Setting

Deterministic Setting



Probabilistic Setting



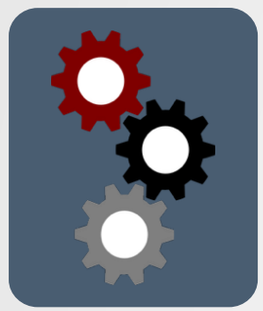
$$P(s) \;=\; \sum_{s' \in \mathcal{S}} \Pr[[\![c]\!](s) = s'] \cdot Q(s')$$

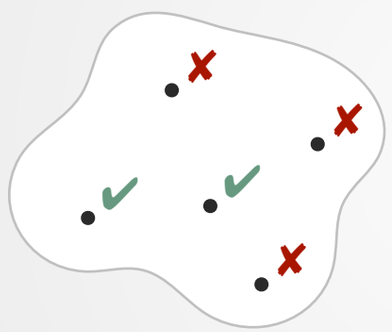
The Program Verification Problem — Deterministic vs Probabilistic Setting

Deterministic Setting

Postcondition $Q\colon \mathcal{S} \to \{0,1\}$

Deterministic Program *c*





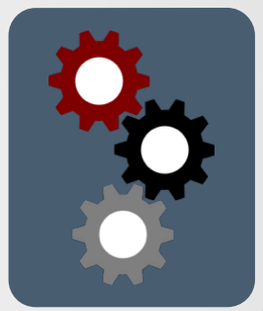
Precondition $P\colon \mathcal{S} \to \{0,1\}$

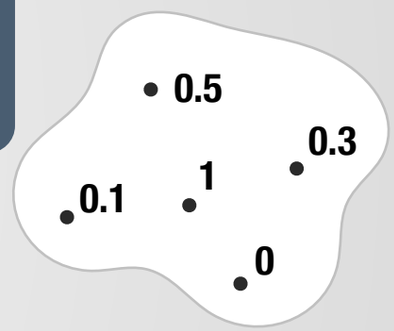
$P(s) = Q([\![c]\!](s))$

Probabilistic Setting

Postcondition $Q\colon \mathcal{S} \to [0,1]$

Probabilistic Program *c*





Precondition $P\colon \mathcal{S} \to [0,1]$

$P(s) = \Pr[Q \in [\![c]\!](s)]$

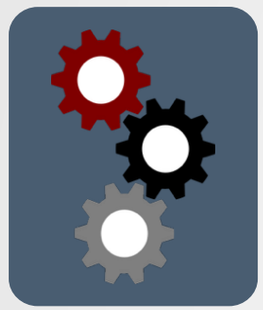
$$P(s) \;=\; \sum_{s' \in \mathcal{S}} \Pr[[\![c]\!](s) = s'] \cdot Q(s')$$

The Program Verification Problem — Deterministic vs Probabilistic Setting

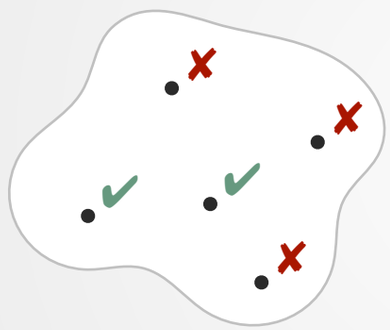
Deterministic Setting

Postcondition $Q\colon \mathcal{S} \to \{0,1\}$

Deterministic Program *c*



Initial States



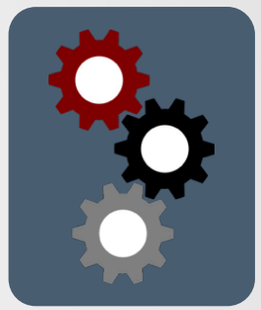
Precondition $P\colon \mathcal{S} \to \{0,1\}$

$P(s) = Q([\![c]\!](s))$

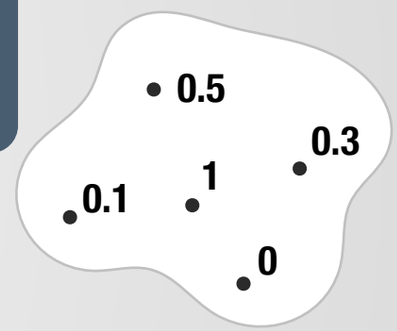
Probabilistic Setting

Postcondition $Q\colon \mathcal{S} \to [0,1]$

Probabilistic Program *c*



Initial States



Precondition $P\colon \mathcal{S} \to [0,1]$

$P(s) = \mathbf{EV}_{[\![c]\!](s)}(Q)$

$$P(s) = \sum_{s' \in \mathcal{S}} \Pr[[\![c]\!](s) = s'] \cdot Q(s')$$

Expected value of Q wrt distribution of final states

Probabilistic Predicate Transformer

$$\mathsf{wp}[c] : (\mathcal{S} \to [0,1]) \to (\mathcal{S} \to [0,1])$$
$$\mathsf{wp}[c](f) \;=\; \lambda s \bullet \mathbf{EV}_{[\![c]\!](s)}(f)$$

Probabilistic Predicate Transformer

- We let $\mathbb{E} \triangleq \mathcal{S} \to [0,1]$ be the set of $[0,1]$-valued expectations (formally, random variables).

- For each program c, *expectation transformer*

(post-)expectation

(pre-)expectation

$$\mathsf{wp}[c] : \mathbb{E} \to \mathbb{E}$$

$$\mathsf{wp}[c](f) \;=\; \lambda s \bullet \mathbf{EV}_{[\![c]\!](s)}(f)$$

transforms post-expectations into pre-expectations.

Probabilistic Predicate Transformer

- We let $\mathbb{E} \triangleq \mathcal{S} \to [0,1]$ be the set of $[0,1]$-valued expectations (formally, random variables).

- For each program *c*, *expectation transformer*

(post-)expectation

(pre-)expectation

$$\mathsf{wp}[c] : \mathbb{E} \to \mathbb{E}$$

$$\mathsf{wp}[c](f) \;=\; \lambda s \bullet \mathbf{EV}_{[\![c]\!](s)}(f)$$

transforms post-expectations into pre-expectations.

- In general, post-expectation *f* can be viewed as a reward function over the set of final states and *wp[c](f)* gives, for each initial state, the “avarage” reward of program *c*.

- If we instantiate *f = [Q]* with the characteristic function of predicate *Q*, we recover the probability of establishing *Q* at the end of the program execution.

Probabilistic Predicate Transformer

Examples

- c_1 :
$$\{x := 0\}\ [p]\ \{x := 1\};\ \{y := 0\}\ [q]\ \{y := 1\}$$

$$\begin{aligned} \mathsf{wp}[c_1](f) &= \lambda s \bullet\ pq \cdot f(s[x,y/0,0]) + p\bar{q} \cdot f(s[x,y/0,1]) \\ &\quad + \bar{p}q \cdot f(s[x,y/1,0]) + \bar{p}\bar{q} \cdot f(s[x,y/1,1]) \\ \Pr[x \leq y] &= pq + p\bar{q} + \bar{p}\bar{q} = p + \bar{p}\bar{q} \end{aligned}$$

- c_2 :
$n := 0;$
`repeat`
$n := n + 1;$
$c :=$ `coin_flip`(0.5)
`until` $(c{=}heads)$

$$\begin{aligned} \mathsf{wp}[c_2](f) &= \lambda s \bullet \sum_{i \geq 1} \left(\tfrac{1}{2}\right)^i f(s[c, n/heads, i]) \\ \Pr[n \leq 5] &= \sum_{1 \leq i \leq 5} \left(\tfrac{1}{2}\right)^i = \tfrac{31}{32} \end{aligned}$$

Probabilistic Predicate Transformers — Calculation

The pGCL language

$\mathcal{C}$	:=	`skip`	nop
	|	`abort`	abortion
	|	$x := E$	assignment
	|	`if` G `then` $\mathcal{C}$ `else` $\mathcal{C}$	conditional
	|	$\{\mathcal{C}\}\ [\mathbf{p}]\ \{\mathcal{C}\}$	**probabilistic choice**
	|	`while` G `do` $\mathcal{C}$	while loop
	|	$\mathcal{C}; \mathcal{C}$	sequence

Theorem (McIver & Morgan '96)

For **pGCL** program c, expectation transformer $wp[c]$ can be defined by induction on c structure.

$$
\begin{aligned}
\mathrm{wp}[\texttt{skip}](f) &= f \\
\mathrm{wp}[\texttt{abort}](f) &= \underline{0} \\
\mathrm{wp}[x := E](f) &= f[E/x] \\
\mathrm{wp}[\texttt{if } G \texttt{ then } c_1 \texttt{ else } c_2](f) &= [G] \cdot \mathrm{wp}[c_1](f) + [\neg G] \cdot \mathrm{wp}[c_2](f) \\
\mathrm{wp}[\{c_1\}\ [p]\ \{c_2\}](f) &= p \cdot \mathrm{wp}[c_1](f) + (1-p) \cdot \mathrm{wp}[c_2](f) \\
\mathrm{wp}[c_1; c_2](f) &= (\mathrm{wp}[c_1] \circ \mathrm{wp}[c_2])(f) \\
\mathrm{wp}[\texttt{while } G \texttt{ do } c](f) &= \mu h \bullet ([G] \cdot \mathrm{wp}[c](h) + [\neg G] \cdot f)
\end{aligned}
$$

Probabilistic Predicate Transformers — Calculation

Example

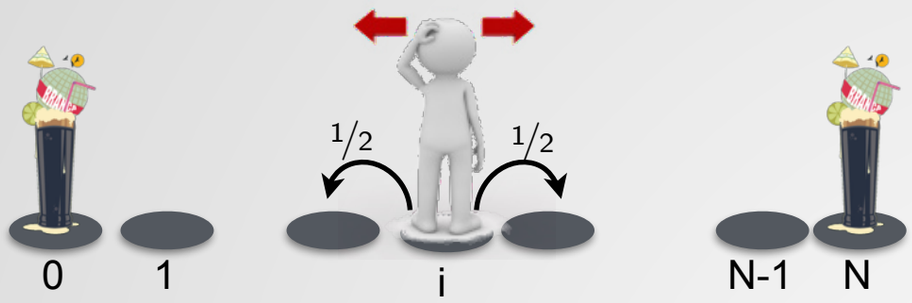
$$c_1: \quad \{x := 0\}\ [p]\ \{x := 1\};\ \{y := 0\}\ [q]\ \{y := 1\}$$

$$\begin{aligned}
\mathsf{wp}[x := E](f) &= f[E/x] \\
\mathsf{wp}[\{c_1\}\ [p]\ \{c_2\}](f) &= p \cdot \mathsf{wp}[c_1](f) + \bar{p} \cdot \mathsf{wp}[c_2](f) \\
\mathsf{wp}[c_1; c_2](f) &= \mathsf{wp}[c_1](\mathsf{wp}[c_2](f))
\end{aligned}$$

$$\begin{aligned}
& \mathsf{wp}[c_1]([x{\leq}y]) \\
= & \quad \langle \text{rule for sequential composition} \rangle \\
& \mathsf{wp}\big[\{x := 0\}\ [p]\ \{x := 1\}\big]\big(\mathsf{wp}\big[\{y := 0\}\ [q]\ \{y := 1\}\big]([x{\leq}y])\big) \\
= & \quad \langle \text{rule for probabilistic choice} \rangle \\
& \mathsf{wp}\big[\{x := 0\}\ [p]\ \{x := 1\}\big]\big(q \cdot \mathsf{wp}\big[y := 0\big]([x{\leq}y]) + \bar{q} \cdot \mathsf{wp}\big[y := 1\big]([x{\leq}y])\big) \\
= & \quad \langle \text{rule for assignment, twice} \rangle \\
& \mathsf{wp}\big[\{x := 0\}\ [p]\ \{x := 1\}\big]\big(q \cdot [x{\leq}0] + \bar{q} \cdot [x{\leq}1]\big) \\
= & \quad \langle \text{rule for probabilistic choice} \rangle \\
& p \cdot \mathsf{wp}\big[x := 0\big]\big(q \cdot [x{\leq}0] + \bar{q} \cdot [x{\leq}1]\big) + \bar{p} \cdot \mathsf{wp}\big[x := 1\big]\big(q \cdot [x{\leq}0] + \bar{q} \cdot [x{\leq}1]\big) \\
= & \quad \langle \text{rule for assignment, twice} \rangle \\
& p \cdot \big(q \cdot [0{\leq}0] + \bar{q} \cdot [0{\leq}1]\big) + \bar{p} \cdot \big(q \cdot [1{\leq}0] + \bar{q} \cdot [1{\leq}1]\big) \\
= & \quad \langle \text{algebra} \rangle \\
& \underline{p + \bar{p}\bar{q}}
\end{aligned}$$

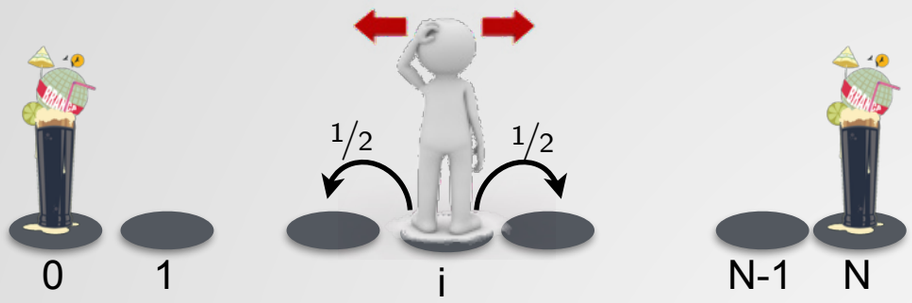
Probabilistic Predicate Transformers — Loop Rule

Will this man get his drink?



Probabilistic Predicate Transformers — Loop Rule

Will this man get his drink?

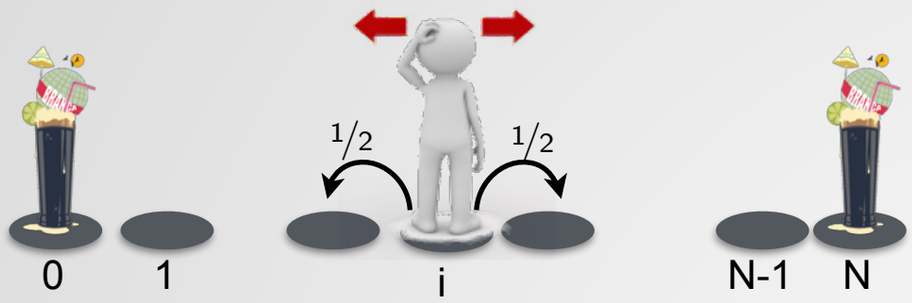


Does this program almost surely terminate?

$$\texttt{while}\,(i \bmod N \neq 0)\,\texttt{do}\;\; \{i := i{+}1\}\ [1/2]\ \{i := i{-}1\}$$

Probabilistic Predicate Transformers — Loop Rule

Will this man get his drink?



Does this program almost surely terminate?

$$\texttt{while}\,(i \bmod N \neq 0)\,\texttt{do}\;\; \{i := i{+}1\}\ [1/2]\ \{i := i{-}1\}$$

Loop Rule for Total Correctness

Consider loop `while` G `do` c and post-expectation f. Assume that

- there exists a standard (ie a predicate) loop invariant I, which restricted to $\neg G$ is stronger than the post-expectation f, and
- there exists a bounded variant e, which in each iteration decreases with at least a fixed probability $\epsilon{>}0$.

Then, $[\![I]\!]$ is a valid pre-expectation of the loop w.r.t. post-expectation f (but not necessarily the weakest).

$$\frac{[\neg G \wedge I] \Rrightarrow f \qquad [G \wedge I] \Rrightarrow \mathsf{wp}[c]([I]) \qquad \exists l,u \in \mathbb{Z} \bullet\ G \wedge I \Rightarrow l \leq e \leq u \qquad \exists \epsilon \in (0,1] \bullet\ \epsilon\,[G \wedge I \wedge e{=}n] \Rrightarrow \mathsf{wp}[c]([e{<}n])}{[I] \Rrightarrow \mathsf{wp}[\texttt{while}\ G\ \texttt{do}\ c](f)}$$

* $f \Rrightarrow g \triangleq \forall s \in S \bullet f(s) \leq g(s)$

Probabilistic Predicate Transformers — Loop Rule

Application example

$$\texttt{while}\,(i \bmod N \neq 0)\,\texttt{do}\quad \{i := i{+}1\}\ [1/2]\ \{i := i{-}1\}$$

$$\frac{\begin{array}{c}[\neg G \wedge I] \Rrightarrow f \qquad [G \wedge I] \Rrightarrow \mathrm{wp}[c]([I]) \\ \exists l,u \in \mathbb{Z} \bullet\ G \wedge I \Rightarrow l \leq e \leq u \qquad \exists \epsilon \in (0,1] \bullet\ \epsilon\,[G \wedge I \wedge e{=}n] \Rrightarrow \mathrm{wp}[c]([e{<}n])\end{array}}{[I] \Rrightarrow \mathrm{wp}[\texttt{while}\ G\ \texttt{do}\ c](f)}$$

The loop (above) terminates almost surely from any initial state. To conclude this, we apply the loop rule with instances

$$f = \underline{1} \qquad I = \underline{\text{true}} \qquad e = i \bmod N \qquad (l,u) = (0, N-1) \qquad \epsilon = 1/2$$

and get the following proof obligations:

- $[i \bmod N = 0] \Rrightarrow \underline{1}$
- $[i \bmod N \neq 0] \Rrightarrow \mathrm{wp}\big[\{i := i{+}1\}\ [1/2]\ \{i := i{-}1\}\big](\underline{1})$
- $0 \leq i \bmod N \leq N{-}1$
- $\frac{1}{2}\,[i \bmod N{\neq}0 \wedge i \bmod N{=}n] \Rrightarrow \mathrm{wp}\big[\{i := i{+}1\}\ [1/2]\ \{i := i{-}1\}\big]([i \bmod N{<}n])$

Probabilistic Predicate Transformers — Loop Rule

Application example

$$\texttt{while}\,(i \bmod N \neq 0)\,\texttt{do}\quad \{i \coloneqq i{+}1\}\ [1/2]\ \{i \coloneqq i{-}1\}$$

$$\frac{\begin{array}{c}[\neg G \wedge I] \Rrightarrow f \qquad [G \wedge I] \Rrightarrow \mathrm{wp}[c]([I]) \\ \exists l,u \in \mathbb{Z} \bullet\ G \wedge I \Rightarrow l \leq e \leq u \qquad \exists \epsilon \in (0,1] \bullet\ \epsilon\,[G \wedge I \wedge e{=}n] \Rrightarrow \mathrm{wp}[c]([e{<}n])\end{array}}{[I] \Rrightarrow \mathrm{wp}[\texttt{while}\ G\ \texttt{do}\ c](f)}$$

The loop (above) terminates almost surely from any initial state. To conclude this, we apply the loop rule with instances

$$f = \underline{1} \qquad I = \underline{\text{true}} \qquad e = i \bmod N \qquad (l,u) = (0, N-1) \qquad \epsilon = 1/2$$

and get the following proof obligations:

- $[i \bmod N = 0] \Rrightarrow \underline{1}$ ✔ (trivial since $\underline{1}$ is the weakest expectation)
- $[i \bmod N \neq 0] \Rrightarrow \mathrm{wp}\big[\{i \coloneqq i{+}1\}\ [1/2]\ \{i \coloneqq i{-}1\}\big](\underline{1})$
- $0 \leq i \bmod N \leq N{-}1$
- $\frac{1}{2}\,[i \bmod N{\neq}0 \wedge i \bmod N{=}n] \Rrightarrow \mathrm{wp}\big[\{i \coloneqq i{+}1\}\ [1/2]\ \{i \coloneqq i{-}1\}\big]([i \bmod N{<}n])$

Probabilistic Predicate Transformers — Loop Rule

Application example

$$\texttt{while}\,(i \bmod N \neq 0)\,\texttt{do}\quad \{i \coloneqq i{+}1\}\ [1/2]\ \{i \coloneqq i{-}1\}$$

$$\frac{\begin{array}{c}[\neg G \wedge I] \Rrightarrow f \qquad [G \wedge I] \Rrightarrow \mathrm{wp}[c]([I]) \\ \exists l,u \in \mathbb{Z} \bullet\ G \wedge I \Rightarrow l \leq e \leq u \qquad \exists \epsilon \in (0,1] \bullet\ \epsilon\,[G \wedge I \wedge e{=}n] \Rrightarrow \mathrm{wp}[c]([e{<}n])\end{array}}{[I] \Rrightarrow \mathrm{wp}[\texttt{while}\ G\ \texttt{do}\ c](f)}$$

The loop (above) terminates almost surely from any initial state. To conclude this, we apply the loop rule with instances

$$f = \underline{1} \qquad I = \underline{\text{true}} \qquad e = i \bmod N \qquad (l,u) = (0, N-1) \qquad \epsilon = 1/2$$

and get the following proof obligations:

- $[i \bmod N = 0] \Rrightarrow \underline{1}$ ✔ (trivial since $\underline{1}$ is the weakest expectation)
- $[i \bmod N \neq 0] \Rrightarrow \mathrm{wp}\big[\{i \coloneqq i{+}1\}\ [1/2]\ \{i \coloneqq i{-}1\}\big](\underline{1})$ ✔ (loop body is AST)
- $0 \leq i \bmod N \leq N{-}1$
- $\frac{1}{2}\,[i \bmod N{\neq}0 \wedge i \bmod N{=}n] \Rrightarrow \mathrm{wp}\big[\{i \coloneqq i{+}1\}\ [1/2]\ \{i \coloneqq i{-}1\}\big]([i \bmod N{<}n])$

Probabilistic Predicate Transformers — Loop Rule

Application example

$$\texttt{while}\,(i \bmod N \neq 0)\,\texttt{do}\quad \{i \coloneqq i{+}1\}\ [1/2]\ \{i \coloneqq i{-}1\}$$

$$\frac{[\neg G \wedge I] \Rrightarrow f \qquad [G \wedge I] \Rrightarrow \mathrm{wp}[c]([I]) \qquad \exists l,u \in \mathbb{Z} \bullet\ G \wedge I \Rightarrow l \leq e \leq u \qquad \exists \epsilon \in (0,1] \bullet\ \epsilon\,[G \wedge I \wedge e{=}n] \Rrightarrow \mathrm{wp}[c]([e{<}n])}{[I] \Rrightarrow \mathrm{wp}[\texttt{while}\ G\ \texttt{do}\ c](f)}$$

The loop (above) terminates almost surely from any initial state. To conclude this, we apply the loop rule with instances

$$f = \underline{1} \qquad I = \underline{\text{true}} \qquad e = i \bmod N \qquad (l,u) = (0, N-1) \qquad \epsilon = 1/2$$

and get the following proof obligations:

- $[i \bmod N = 0] \Rrightarrow \underline{1}$ ✔ (trivial since $\underline{1}$ is the weakest expectation)
- $[i \bmod N \neq 0] \Rrightarrow \mathrm{wp}\big[\{i \coloneqq i{+}1\}\ [1/2]\ \{i \coloneqq i{-}1\}\big](\underline{1})$ ✔ (loop body is AST)
- $0 \leq i \bmod N \leq N{-}1$ ✔ (trivial)
- $\frac{1}{2}\,[i \bmod N{\neq}0 \wedge i \bmod N{=}n] \Rrightarrow \mathrm{wp}\big[\{i \coloneqq i{+}1\}\ [1/2]\ \{i \coloneqq i{-}1\}\big]([i \bmod N{<}n])$

Probabilistic Predicate Transformers — Loop Rule

Application example

$$\texttt{while}\,(i \bmod N \neq 0)\,\texttt{do}\quad \{i \coloneqq i{+}1\}\ [1/2]\ \{i \coloneqq i{-}1\}$$

$$\frac{\exists l,u \in \mathbb{Z} \bullet\ G \wedge I \Rightarrow l \leq e \leq u \qquad \exists \epsilon \in (0,1] \bullet\ \epsilon\,[G \wedge I \wedge e{=}n] \Rrightarrow \mathrm{wp}[c]([e{<}n]) \qquad [\neg G \wedge I] \Rrightarrow f \qquad [G \wedge I] \Rrightarrow \mathrm{wp}[c]([I])}{[I] \Rrightarrow \mathrm{wp}[\texttt{while}\ G\ \texttt{do}\ c](f)}$$

The loop (above) terminates almost surely from any initial state. To conclude this, we apply the loop rule with instances

$$f = \underline{1} \qquad I = \underline{\text{true}} \qquad e = i \bmod N \qquad (l,u) = (0, N-1) \qquad \epsilon = 1/2$$

and get the following proof obligations:

- $[i \bmod N = 0] \Rrightarrow \underline{1}$ ✔ (trivial since $\underline{1}$ is the weakest expectation)
- $[i \bmod N \neq 0] \Rrightarrow \mathrm{wp}\big[\{i \coloneqq i{+}1\}\ [1/2]\ \{i \coloneqq i{-}1\}\big](\underline{1})$ ✔ (loop body is AST)
- $0 \leq i \bmod N \leq N{-}1$ ✔ (trivial)
- $\frac{1}{2}\,[i \bmod N{\neq}0 \wedge i \bmod N{=}n] \Rrightarrow \mathrm{wp}\big[\{i \coloneqq i{+}1\}\ [1/2]\ \{i \coloneqq i{-}1\}\big]([i \bmod N{<}n])$ ✔ (case analysis on $i = N{-}1$)

Probabilistic Predicate Transformers — Loop Rule

$$\frac{\exists l,u \in \mathbb{Z} \bullet\ G \wedge I \Rightarrow l \leq e \leq u \qquad \exists \epsilon \in (0,1] \bullet\ \epsilon\,[G \wedge I \wedge e{=}n] \Rrightarrow \mathsf{wp}[c]([I{<}n]) \qquad [\neg G \wedge I] \Rrightarrow f \qquad [G \wedge I] \Rrightarrow \mathsf{wp}[c]([I])}{[I] \Rrightarrow \mathsf{wp}[\texttt{while}\ G\ \texttt{do}\ c](f)}$$

- Rule is not complete

$i := 100;$

$\texttt{while}\,(i{>}0)\,\texttt{do}$

$\quad \{i := i{+}1\}\ [1/2]\ \{i := i{-}1\}$

Rule fails to prove AST of the LHS loop.

- Rule leads only {0,1}—valued pre-expectations.

Probabilistic Predicate Transformers — The Liberal Version

Intuition

$\mathsf{wp}\big[c\big]([P])$ ➡ probability of terminating and establishing P.

$\mathsf{wlp}\big[c\big]([P])$ ➡ probability of diverging or establishing P.

Example

$c:\quad \{\{x \coloneqq 0\}\ [p]\ \{x \coloneqq 1\}\}\ [q]\ \{\texttt{abort}\}$

$$\begin{aligned}\mathsf{wp}[c]([x{=}0]) &= \underline{pq}\\ \mathsf{wlp}[c]([x{=}0]) &= \underline{pq + \bar{q}}\end{aligned}$$

Formal Definition

Only the rules for abort and while—loops change

$$\begin{aligned}\mathsf{wlp}[\texttt{abort}](f) &= \underline{1}\\ \mathsf{wlp}[\texttt{while}\ G\ \texttt{do}\ c](f) &= \nu h \bullet ([G]\cdot \mathsf{wlp}[c](h) + [\neg G]\cdot f)\end{aligned}$$

Probabilistic Predicate Transformers — The Liberal Version

Proof rule for loops

No “variant” argument required (cf $\mathsf{wp}[\cdot]$)

$$\frac{[\neg G \wedge I] \Rrightarrow f \qquad [G \wedge I] \Rrightarrow \mathsf{wlp}[c]([I])}{[I] \Rrightarrow \mathsf{wlp}[\texttt{while}\ G\ \texttt{do}\ c](f)}$$

Relation between transformers

$$\mathsf{wp}[c](f) \;\Leftarrow\; \mathsf{wlp}[c](f)\ \&\ \mathsf{wp}[c](\underline{1})$$

where $a \,\&\, b \triangleq \max\{a+b-1, 0\}$

For deterministic programs,

$$\mathsf{wp}[c](Q) \;=\; \mathsf{wlp}[c](Q) \wedge \mathsf{wp}[c](\mathsf{true})$$

Deterministic vs probabilistic world

Deterministic vs probabilistic world

Program outcome	Deterministic Setting	Probabilistic Setting

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition		

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome		

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$
Weakest pre-condition		

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$
Weakest pre-condition	true	

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$
Weakest pre-condition	true	1

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$
Weakest pre-condition	true	1
Termination		

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$
Weakest pre-condition	$\underline{\text{true}}$	$\underline{1}$
Termination	$\text{wp}[\cdot](\underline{\text{true}})$	

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$
Weakest pre-condition	$\underline{\text{true}}$	$\underline{1}$
Termination	$\text{wp}[\cdot](\underline{\text{true}})$	$\text{wp}[\cdot](\underline{1})$

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$
Weakest pre-condition	$\underline{\text{true}}$	$\underline{1}$
Termination	$\text{wp}[\cdot](\underline{\text{true}})$	$\text{wp}[\cdot](\underline{1})$
Implication		

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$
Weakest pre-condition	$\underline{\text{true}}$	$\underline{1}$
Termination	$\text{wp}[\cdot](\underline{\text{true}})$	$\text{wp}[\cdot](\underline{1})$
Implication	$P \Rightarrow Q$	

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$
Weakest pre-condition	$\underline{\text{true}}$	$\underline{1}$
Termination	$\text{wp}[\cdot](\underline{\text{true}})$	$\text{wp}[\cdot](\underline{1})$
Implication	$P \Rightarrow Q$	$f \Rrightarrow g$

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$
Weakest pre-condition	$\underline{\text{true}}$	$\underline{1}$
Termination	$\text{wp}[\cdot](\underline{\text{true}})$	$\text{wp}[\cdot](\underline{1})$
Implication	$P \Rightarrow Q$	$f \Rrightarrow g$
Conjunction		

Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$
Weakest pre-condition	$\underline{\text{true}}$	$\underline{1}$
Termination	$\text{wp}[\cdot](\underline{\text{true}})$	$\text{wp}[\cdot](\underline{1})$
Implication	$P \Rightarrow Q$	$f \Rrightarrow g$
Conjunction	$P \wedge Q$	

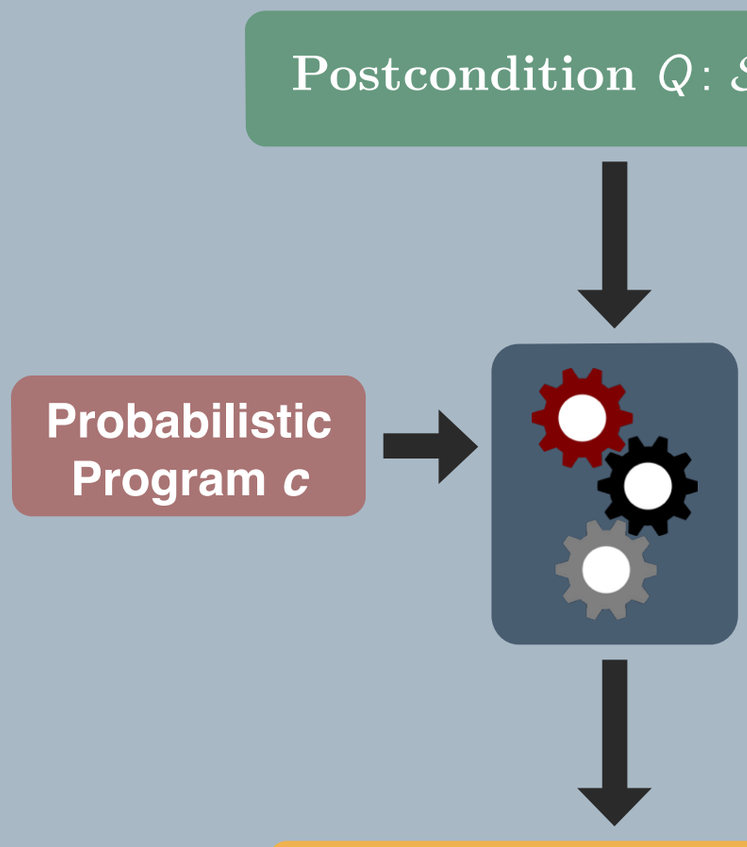
Deterministic vs probabilistic world

	Deterministic Setting	Probabilistic Setting
Program outcome	Final state $s' \in \mathcal{S}$	Distribution of final states $\mu' \in \mathcal{D}(\mathcal{S})$
Postcondition	Predicate over program state $Q : \mathcal{S} \to \{0,1\}$	[0,1]—valued expectation $f : \mathcal{S} \to [0,1]$
Evaluation of postcondition on program outcome	Membership (or satisfiability) $s' \in Q$ (or $s' \models Q$)	Expected value $\mathbf{E}_{\mu'}(f)$
Weakest pre-condition	$\underline{\text{true}}$	$\underline{1}$
Termination	$\text{wp}[\cdot](\underline{\text{true}})$	$\text{wp}[\cdot](\underline{1})$
Implication	$P \Rightarrow Q$	$f \Rrightarrow g$
Conjunction	$P \wedge Q$	$f \,\&\, g$

Agenda

- Recap on previous lecture
- Predicate transformers for deterministic programs
- Predicate transformers for probabilistic programs
- Summary

Summary



The pGCL language

$\mathcal{C}$	:=	`skip`	nop
	|	`abort`	abortion
	|	$x := E$	assignment
	|	`if` G `then` $\mathcal{C}$ `else` $\mathcal{C}$	conditional
	|	$\{\mathcal{C}\}\ [p]\ \{\mathcal{C}\}$	**probabilistic choice**
	|	`while` G `do` $\mathcal{C}$	while loop
	|	$\mathcal{C}; \mathcal{C}$	sequence

Theorem (McIver & Morgan '96)

For **pGCL** program c, expectation transformer *wp[c]* can be defined by induction on c structure.

$$\mathrm{wp}\big[\{c_1\}\ [p]\ \{c_2\}\big](Q) = p \cdot \mathrm{wp}[c_1](f) + (1-p) \cdot \mathrm{wp}[c_2](f)$$

$\mathrm{wp}\big[c\big]([P])$ → probability of terminating and establishing P.

$\mathrm{wlp}\big[c\big]([P])$ → probability of diverging or establishing P.

Backup Slide

Proof rules for loops (deterministic case)

$$\frac{I \wedge G \wedge e{>}k \implies \mathsf{wp}[c](I \wedge e{=}k) \qquad I \wedge \neg G \implies Q \qquad I \wedge e{\leq}0 \implies \neg G}{I \implies \mathsf{wp}[\texttt{while}\ G\ \texttt{do}\ c](Q)}$$

$$\frac{I \wedge G \implies \mathsf{wlp}[c](I \wedge e{=}k) \qquad I \wedge \neg G \implies Q}{I \implies \mathsf{wlp}[\texttt{while}\ G\ \texttt{do}\ c](Q)}$$

Alternative characterisation of expectation transformer

$$\forall s \in \mathcal{S} \bullet\ \mathsf{wp}[c](f)(s) = \mathbf{E}_{[\![c]\!]s}(f) \quad \equiv \quad \forall \mu \in \mathcal{D}(\mathcal{S}) \bullet\ \mathbf{E}_{\mu}(\mathsf{wp}[c](f)) = \mathbf{E}_{[\![c]\!]\mu}(f)$$