Probabilistic Programming

Lecture #6: Syntax and Operational Semantics of pGCL

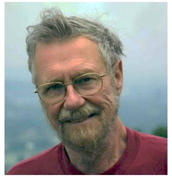
Joost-Pieter Katoen

RWTH Lecture Series on Probabilistic Programming 2022-23

Overview

Overview

Dijkstra's guarded command language: Syntax



- `skip` — empty statement
- `diverge` — divergence
- `x := E` — assignment
- `prog1 ; prog2` — sequential composition
- `if (G) prog1 else prog2` — choice
- `prog1 [] prog2` — non-deterministic choice
- `while (G) prog` — iteration

Elementary `pGCL` ingredients

- Program variables $x \in \mathit{Vars}$ whose values are fractional numbers
- Arithmetic expressions E over the program variables
- Boolean expressions G (aka: guards) over the program variables
- Distribution expressions $\mu : \Sigma \to \mathit{Dist}(\mathbb{Q})$
- Probability expressions $p : \Sigma \to [0, 1] \cap \mathbb{Q}$

where Σ is the set of program states; made precise later

Probabilistic GCL: Syntax

Dexter Kozen

Annabelle McIver

Carroll Morgan

- **`skip`** — empty statement
- **`diverge`** — divergence
- `x := E` — assignment
- `x :r= mu` — **random assignment** ($x :\approx \mu$)
- `prog1 ; prog2` — sequential composition
- **`if`** `(G) prog1` **`else`** `prog2` — choice
- `prog1 [p] prog2` — **probabilistic choice**
- **`while`** `(G) prog` — iteration

Conditioning in the form of observe-statements omitted for now.

Let's start simple

```
x := 0  [0.5]  x := 1;
y := -1 [0.5]  y := 0
```

This program admits four runs and yields the outcome:

$$Pr[x=0, y=0] \;=\; Pr[x=0, y=-1] \;=\; Pr[x=1, y=0] \;=\; Pr[x=1, y=-1] \;=\; 1/4$$

A loopy program

For $0 < p < 1$ an arbitrary probability:

```
bool c := true;
int i := 0;
while (c) {
    i++;
    (c := false [p] c := true)
}
```

The loopy program models a geometric distribution with parameter p.

$$Pr[i = N] \;=\; (1-p)^{N-1} \cdot p \quad \text{for } N > 0$$

On termination

```
bool c := true;
int i := 0;
while (c) {
    i++;
    (c := false [p] c := true)
}
```

This program does not always terminate. It almost surely terminates.

The good, the bad, and the ugly

Duelling cowboys

```
int cowboyDuel(float a, b) {
  int t := A [0.5] t := B;
  bool c := true;
  while (c) {
    if (t = A) {
      (c := false [a] t := B);
    } else {
      (c := false [b] t := A);
    }
  }
  return t;
}
```


Random assignments

The random assignment $x :\approx \mu$ works as follows:

1. evaluate distribution expression μ in the current program state s
2. sample from the resulting probability distribution $\mu(s)$

   this yields the value v with probability $\mu(s)(v)$
3. assign the value v to the variable x.

For denoting distribution expressions, we use the bra–ket notation.

$$\frac{1}{2} \cdot [a\rangle + \frac{1}{3} \cdot [b\rangle + \frac{1}{6} \cdot [c\rangle$$

denotes the distribution μ with $\mu(a) = 1/2$, $\mu(b) = 1/3$, and $\mu(c) = 1/6$. The support set of μ equals $\{ a, b, c \}$

Examples on the black board.

Overview

1. Probabilistic Guarded Command Language
2. Operational semantics of pGCL
3. Expected Rewards
4. Recursion

Why formal semantics matters

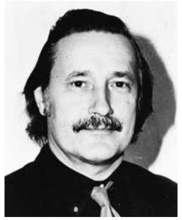
- Unambiguous meaning to all programs
- Basis for proving correctness
  - of programs
  - of program transformations
  - of program equivalence
  - of static analysis
  - of compilers
  - ......

The inventors of semantics

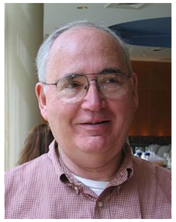
Tony Hoare

Robert W. Floyd

Gordon Plotkin



Christopher Strachey



Dana Scott

Approaches to semantics

- Operational semantics: (developed by Plotkin)
  - The meaning of a program = how it executes on an abstract machine.
  - Useful for modelling the execution behaviour of a program.
- Axiomatic semantics: (developed by Floyd and Hoare)
  - Provides correctness assertions for each program construct.
  - Useful for verifying that the program's computed results are correct with respect to the specification.
- Denotational semantics: (developed by Strachey and Scott)
  - Provides a mapping of language constructs onto mathematical objects.
  - Useful for obtaining an abstract insight into the working of a program.

Today: operational semantics of `pGCL` in terms of Markov chains.

Later: denotational semantics of `pGCL` in terms of weakest preconditions.

Structural operational semantics: ingredients

- Variable valuation $s : Vars \to \mathbb{Q}$ maps each program variable onto a value (here: rational numbers)
- Variable update: for variable x and value $v \in \mathbb{Q}$, let

$$s[x := v](y) = s(y) \quad \text{if } x \neq y \quad \text{and} \quad s[x := v](y) = v \text{ otherwise.}$$

- Let $[\![E]\!]$ denote the valuation of expression E
- Program configuration (aka: state) $\langle P, s\rangle$ denotes that
  - program P is next to be executed (aka: program counter), and
  - the current variable valuation equals s.
- Transition rules for the execution of commands: $\langle P, s\rangle \longrightarrow \langle P', s'\rangle$

  denoted as $\dfrac{\text{premise}}{\text{conclusion}}$ where the premise is omitted if it equals true.

Operational semantics

Aim: Model the behaviour of a program P by the MC $[\![P]\!]$.

This MC is defined using structured operational semantics

Operational semantics

Aim: Model the behaviour of a program P by the MC $[\![P]\!]$.

Approach:

- Take states of the form
  - $\langle Q, s\rangle$ with program Q or $\downarrow$, and variable valuation $s : Var \to \mathbb{Q}$
  - $\langle sink\rangle$ models program termination (successful)
- Take initial state $\langle P, s\rangle$ where s fulfils the initial conditions
- Take transition relation $\to$ as smallest relation satisfying the transition rules

Transition rules (1)

$$\langle \texttt{skip}, s\rangle \to \langle \downarrow, s\rangle$$

$$\langle \downarrow, s\rangle \to \langle sink\rangle \qquad \langle sink\rangle \to \langle sink\rangle$$

$$\langle x := E, s\rangle \to \langle \downarrow, s[x := s([\![E]\!])]\rangle$$

$$\frac{\mu(s)(v) = a > 0}{\langle x :\approx \mu, s\rangle \xrightarrow{a} \langle \downarrow, s[x := v]\rangle}$$

$$\langle P[\,p\,]\,Q, s\rangle \to \mu \text{ with } \mu(\langle P, s\rangle) = p \text{ and } \mu(\langle Q, s\rangle) = 1-p$$

Random assignments

The random assignment $x :\approx \mu$ works as follows:

1. evaluate distribution expression μ in the current program state s
2. sample from the resulting probability distribution $\mu(s)$

   this yields the value v with probability $\mu(s)(v)$
3. assign the value v to the variable x.

For denoting distribution expressions, we use the bra–ket notation.

$$\frac{1}{2} \cdot [a\rangle + \frac{1}{3} \cdot [b\rangle + \frac{1}{6} \cdot [c\rangle$$

denotes the distribution μ with $\mu(a) = 1/2$, $\mu(b) = 1/3$, and $\mu(c) = 1/6$. The support set of μ equals $\{ a, b, c \}$

Examples on the black board.

Transition rules (2)

$$\frac{\langle P, s\rangle \to \mu}{\langle P; Q, s\rangle \to \nu} \text{ with } \nu(\langle P'; Q', s'\rangle) = \mu(\langle P', s'\rangle) \text{ where } \downarrow; Q = Q$$

$$\frac{s \models G}{\langle \text{if } (G)\{P\} \text{ else } \{Q\}, s\rangle \to \langle P, s\rangle} \qquad \frac{s \not\models G}{\langle \text{if } (G)\{P\} \text{ else } \{Q\}, s\rangle \to \langle Q, s\rangle}$$

$$\frac{s \models G}{\langle \text{while}(G)\{P\}, s\rangle \to \langle P; \text{while } (G)\{P\}, s\rangle} \qquad \frac{s \not\models G}{\langle \text{while}(G)\{P\}, s\rangle \to \langle \downarrow, s\rangle}$$

Example

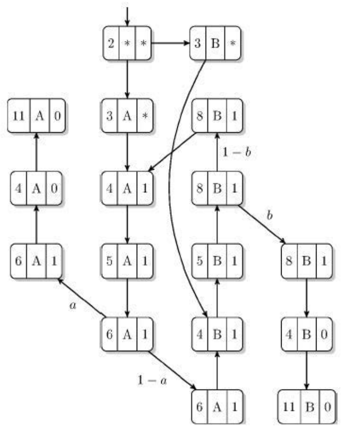
```
x := 0 [1/3] x := 1;
y := unif[1..5]
```

```
x := unif[1..5];
if (x >= 2) { y := unif[1..x] } else { y := x }
```

```
x +:= 1 [1/(abs(x)+1)] x -:= 1
```


Duelling cowboys

```
int cowboyDuel(float a, b) {
  int t := A [0.5] t := B;
  bool c := true;
  while (c) {
    if (t = A) {
      (c := false [a] t := B);
    } else {
      (c := false [b] t := A);
    }
  }
  return t;
}
```



This (parametric) MC is finite. Once we count the number of shots before one of the cowboys dies, the MC becomes countably infinite.

Duelling cowboys

```
int cowboyDuel(float a, b) { // 0 < a < 1, 0 < b < 1
   int t := A [0.5] t := B; // decide who shoots first
   bool c := true;
   while (c) {
     if (t = A) {
         (c := false [a] t := B); // A shoots B with prob. a
     } else {
         (c := false [b] t := A); // B shoots A with prob. b
     }
   }
return t; // the survivor
}
```

Claim:

Cowboy A wins the duel with probability $\frac{(1-b)\cdot\frac{1}{2}a}{a+b-a\cdot b}$.

Proof

The outcome of a pGCL program

Unlike a deterministic program, a pGCL program P has not a deterministic output for a given input. Instead, it yields a unique probability distribution over its final states.

In fact, this is a sub-distribution (probability mass at most one), as with a (possibly positive) probability, P may diverge.

Let P be a pGCL program and s an input state. Then the distribution over final states obtained by running P starting in s is given by $Pr(s \models \Diamond \langle\downarrow, \cdot\rangle)$.

If P is a program whose MC is finite-state, then $Pr(s \models \Diamond \langle\downarrow, \cdot\rangle)$ can be determined by solving a linear equation system.

Overview

Rewards

To reason about resource usage in MCs: use rewards.

A reward MC is a pair (D, r) with D an MC with state space Σ and $r : \Sigma \to \mathbb{R}$ a function assigning a real reward to each state.

The reward $r(\sigma)$ stands for the reward earned on leaving state σ.

Let $\pi = \sigma_0 \ldots \sigma_n$ be a finite path in (D, r) and $G \subseteq \Sigma$ a set of target states with $\pi \in \Diamond G$. The cumulative reward along π until reaching G is:

$$r_G(\pi) = r(\sigma_0) + \ldots + r(\sigma_{k-1}) \text{ where } \sigma_i \notin G \text{ for all } i < k \text{ and } \sigma_k \in G.$$

If $\pi \notin \Diamond G$, then $r_G(\pi) = \infty$.

Expected reward for reachability

Let σ be such that $Pr(\sigma \models \Diamond G) = 1$.

Then: the expected reward until reaching $G \subseteq \Sigma$ from $\sigma \in \Sigma$ is:

$$\mathrm{ER}(\sigma, \Diamond G) \ = \ \sum_{\widehat{\pi}} Pr(\widehat{\pi}) \cdot r_G(\widehat{\pi})$$

where $\widehat{\pi} = \sigma_0 \ldots \sigma_k$ is such that $\sigma_k \in G$, $\sigma_0 = \sigma$ and $\sigma_i \notin G$ for all $i < k$.

If $Pr(\sigma \models \Diamond G) < 1$, then let $\mathrm{ER}(\sigma, \Diamond G) = \infty$.

On computing expected rewards

Expected rewards in finite Markov chains can be computed in polynomial time by solving a system of linear equations.
(details on the black board.)

Equation system for expected rewards

Overview

1. Probabilistic Guarded Command Language
2. Operational semantics of pGCL
3. Expected Rewards
4. Recursion

Probabilistic GCL with recursion: Syntax

- `skip` — empty statement
- `x := E` — assignment
- `x :r= mu` — random assignment ($x :\approx \mu$)
- `prog1 ; prog2` — sequential composition
- `if (G) prog1 else prog2` — choice
- `prog1 [p] prog2` — probabilistic choice
- `while (G) prog` — iteration
- `proc P = prog` — **process definition**
- `call P` — **process invocation**

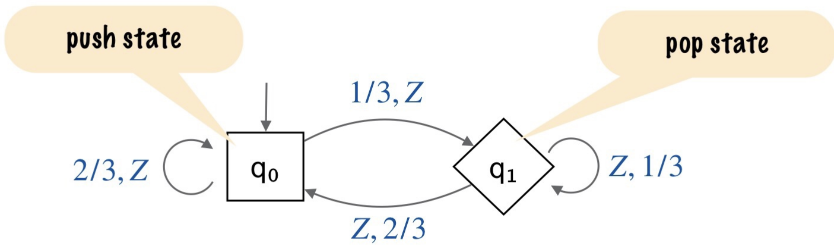
Recursion does not increase the expressive power, but is often convenient.

Pushdown Markov chains

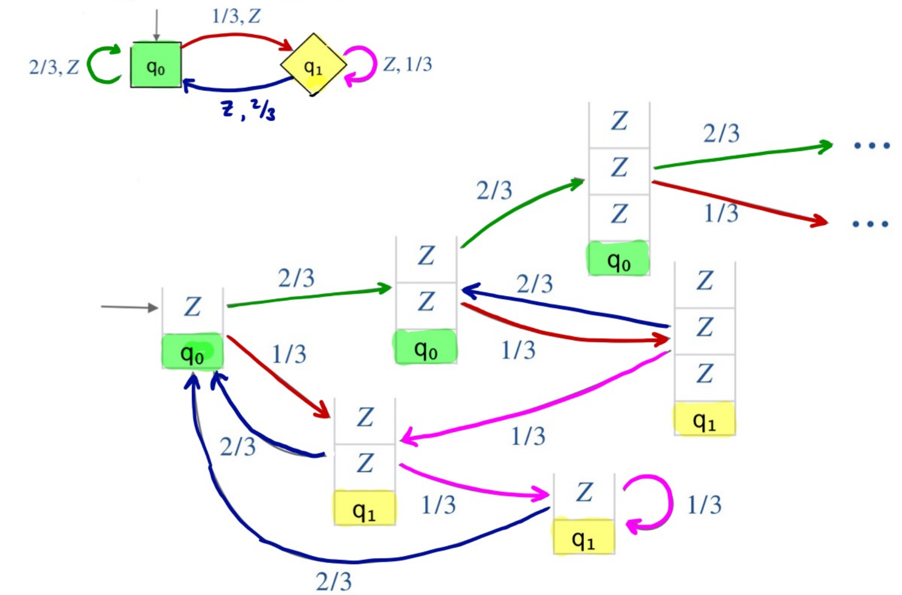
A pushdown Markov chain consists of:

- $\Sigma = (\Sigma_{push}, \Sigma_{pop}, \Sigma_{int})$, a finite set of states
- Γ is a finite stack alphabet
- $(\sigma_I, Z_0) \in \Sigma \times \Gamma$, the initial configuration
- probabilistic transition functions:
  - $P_{push} : \Sigma_{push} \to Dist(\Sigma \times \Gamma)$ — push transitions
  - $P_{pop} : \Sigma_{pop} \times \Gamma \to Dist(\Sigma)$ — pop transitions
  - $P_{int} : \Sigma_{int} \to Dist(\Sigma)$ — internal transitions

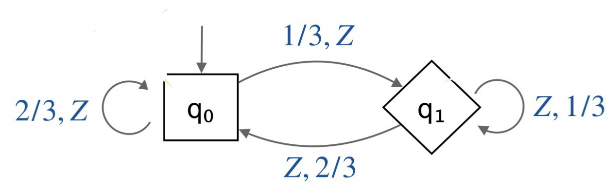
Pushdown MC = Markov chain + stack

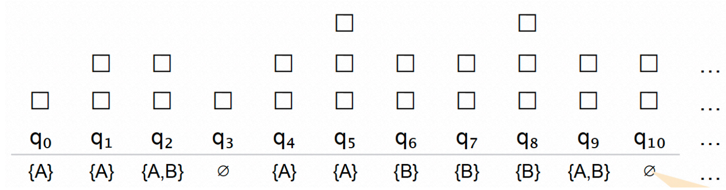


Configuration graph



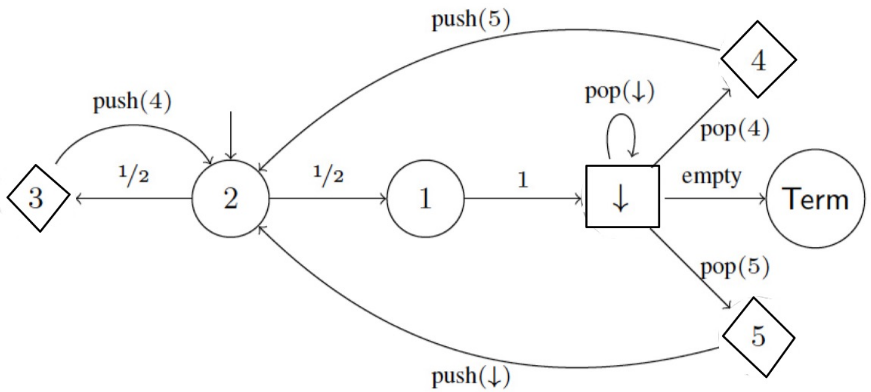
Runs of a pushdown MC





Assuming states are labeled with sets of atomic propositions

From programs to pushdown MCs



{skip^1} $[1/2]^2$ {call P^3; call P^4; call P^5 }

Take-home messages

- pGCL is a "base" imperative probabilistic programming language
- Key ingredients: probabilistic choice and random assignments
- A pGCL program corresponds to a (countably infinite) Markov chain
- Computing expected rewards in finite MCs = solving linear equations
- Recursion can be added to pGCL and yields pushdown MCs

Next lecture

Tuesday Nov 8, 16:30

No lecture on Nov 3;

next exercise class Nov 4