tightcenter

panikzettel.htwr-aachen.de

Probabilistic Programming Panikzettel

Philipp Schröer, Caspar Zecha

Version 6 — 31.12.2020

Contents

1 Introduction

This Panikzettel is about the lecture Probabilistic Programming by Prof. Katoen held in the winter semester 2018/2019.

This Panikzettel is Open Source. We appreciate comments and suggestions at `https://git.rwth-aachen.de/philipp.schroer/panikzettel`.

2 Markov Chains

Markov chains are essential in our definition of the semantics of the pGCL probabilistic programming language. A Markov chain is a transition system with a state set, an initial state and a transition probability function between states.

So instead of a simple transition relation between states as in usual transition systems, transitions now additionally have a probability. The probability of a transition from σ_1 to σ_2 is given by:

$$\underbrace{\mathbf{P}(\sigma_1)}_{\text{Dist}(\Sigma)}(\sigma_2) \subseteq [0,1]$$

If Σ is finite, we can also write $\mathbf{P}$ as a *transition probability matrix*. This matrix is a square, stochastic matrix, i.e. each row sums to one.

Definition: Probability distribution

A *probability distribution* on a countable set X is a function $\mu : X \to [0,1] \subseteq \mathbb{R}$ such that $\sum_{x \in X} \mu(x) = 1$.

We call $\{\, x \mid \mu(x) > 0 \,\}$ the *support set* of μ. Let $\text{Dist}(X)$ denote the set of probability distributions on X.

Definition: Markov chain

A *Markov chain* (MC) D is a triple $(\Sigma, \sigma_I, \mathbf{P})$:

- Σ being a countable set of *states*,
- $\sigma_I \in \Sigma$ the *initial state*,
- $\mathbf{P} : \Sigma \to \text{Dist}(\Sigma)$ the *transition probability function*.

A program execution will be a *path* through the Markov chain. A path in a Markov chain is defined as a (possibly infinite) sequence of states where each single transition must have a probability larger than zero.

We define the *cylinder set* of a finite path $\hat{\pi}$ as all infinite paths with prefix $\hat{\pi}$.

We can now define a *probability distribution* on cylinder sets Pr. Given a path finite $\hat{\pi}$, $\Pr(\text{Cyl}(\hat{\pi}))$ is defined as the probability of the transitions between the finite prefix path states.

Note that we define Pr only on cylinder sets, i.e. sets of infinite paths. However, the product is finite and only requires probabilities of the finite path.

Definition: Paths

$\pi = \sigma_0 \sigma_1 \ldots$ is a *path* through MC D where $\mathbf{P}(\sigma_i, \sigma_{i+1}) > 0 \quad \forall\, i \in \mathbb{N}$

Let $\text{Paths}(D)$ denote the set of paths in D starting in σ_I.

Definition: Cylinder set

Let $\hat{\pi} = \sigma_0 \sigma_1 \ldots \sigma_n$ be a finite path in MC D. Then we define the *cylinder set* $\text{Cyl}(\hat{\pi})$:

$$\text{Cyl}(\hat{\pi}) = \{\, \pi \in \text{Paths}(D) \mid \hat{\pi} \text{ is a prefix of } \pi \,\}$$

Definition: Cylinder probability

Pr is the unique *probability distribution on cylinder sets*. ($\mathbf{P}(\sigma_0) = 1$ iff $\sigma_0 = \sigma_I$).

$$\Pr(\text{Cyl}(\sigma_0 \ldots \sigma_n)) = \prod_{0 \le i < n} \mathbf{P}(\sigma_i, \sigma_{i+1})$$

2.1 Reachability

With the definition of the probability of a path, we now consider the more general problem of *reachability*: What is the probability to reach a set of states $G \subseteq \Sigma$ in MC D?

In our definition of reachability, this asks for the probability of all infinite paths containing a state in G anywhere.

Definition: Reachability

Let MC D with countable state space Σ and $G \subseteq \Sigma$ the set of *goal states*. The event *eventually reaching* G is defined by:

$$\Diamond G = \{ \pi \in \text{Paths}(D) \mid \exists i \in \mathbb{N}.\ \pi[i] \in G \}$$

If the Markov chain D has a finite state space, we can calculate the reachability probability by solving a linear equation system. We write D_σ to mean the MC D with initial state σ and $\Pr(\sigma \models \Diamond G) = \Pr_\sigma(\Diamond G) = \Pr(\{ \pi \in \text{Paths}(D_\sigma) \mid \pi \in \Diamond G \})$.

Theorem: Finite reachability solution

To solve $\Pr(\sigma \models \Diamond G)$, define

- $\Sigma_? = \text{Pre}^*(G) \setminus G$, the set of states that can reach G in > 0 steps,
- $A = (\mathbf{P}(\sigma, \tau))_{\sigma,\ \tau \in \Sigma_?}$, the transition probabilities in $\Sigma_?$, and
- $b = (b_\sigma)_{\sigma \in \Sigma_?}$, the probabilities to reach G in exactly one step, i.e. $b_\sigma = \sum_{\gamma \in G} \mathbf{P}(\sigma, \gamma)$.

Then $x = (x_\gamma)_{\gamma \in \Sigma_?}$ with $x_\sigma = \Pr(\sigma \models \Diamond G)$ is the unique solution of $x = A \cdot x + b$ or, equivalently $(I - A) \cdot x = b$.

2.2 State Classification

We will now classify states based on recurrence: Whether the MC is almost sure to return to a state (*recurrent*) or not (*transient*).

Note that the *first visit probability* requires a *first visit*, so only paths that contain τ exactly once at their end are considered.

The *return probability* also requires a first return. It can be calculated by summing up all first visit probabilities.

We now call a state σ *recurrent* if $f_\sigma = 1$ and *transient* if $f_\sigma < 1$. For a transient state σ, we say the MC *almost surely* returns to σ.

Definition: First visit probability

Let states $\sigma, \tau \in \Sigma$.

We define $f^{(n)}_{\sigma,\tau}$ as the probability of a first visit to τ after exactly n steps from σ.

Definition: Return probability

Let states $\sigma, \tau \in \Sigma$.

The *return probability* $f^{(n)}_\sigma$ is defined as the probability of the first return to σ (from σ) after exactly n steps. This is equivalent to: $f_\sigma = \sum_{i=1}^{\infty} f^{(n)}_{\sigma,\sigma}$.

For a recurrent state σ, we also define a *mean recurrence time*: The expected number of steps between two successive visits to σ.

This gives rise to the terms *null* and *postive recurrent*. It takes infinite on average to return to a null recurrent state, while positive recurrent states have a finite mean recurrence time.

Definition: Mean recurrence time

The *mean recurrence time* m_σ of a recurrent state σ is $m_\sigma = \sum_{n=1}^{\infty} n \cdot f_\sigma^{(n)}$

If $m_\sigma < \infty$, we call m_σ *positive recurrent*, otherwise *null recurrent*.

If the MC is finite, then we have a few properties w.r.t. recurrence:

1. Every state in a finite MC is either positive recurrent or transient.
2. At least one state in a finite MC is positive recurrent.
3. A finite MC has no null recurrent states.

To show these properties, we use Foster's theorem.

Theorem: Foster's theorem

A countable Markov chain is *non-dissipative* if almost every infinte path eventually enters and remains in positive recurrent states.

If the following conditions hold, the MC is non-dissipative: $\sum_{j \geq 0} j \cdot \mathbf{P}(i,j) \leq i \quad \forall$ states i

We can also classify Markov chains by periodicity. The definition is a bit tricky, so read it carefully.

Further a state is *ergodic* if it is positive and aperiodic. An MC is ergodic if all its states are ergodic.

Definition: Periodic state

A state σ is *periodic* if $f_\sigma^{(n)} > 0$ implies $n = k \cdot d$ where period $d > 1$. A state is *aperiodic* otherwise.

At this point we may notice that mutually reachable states must have the same types. More formally, if σ and τ are two mutually reachable states, then being transient, null-recurrent, positive recurrent and d-periodic holds for τ if the respective property holds for σ.

If a MC is *irreducible*, that is all states are mutually reachable, we can use Markov's theorem.

Theorem: Markov's theorem

A finite, irreducible MC D is positive recurrent.

If D is also aperiodic, then D is ergodic and $\mathbf{P}^\infty = \lim_{n\to\infty} \mathbf{P}^n = \begin{pmatrix} v \\ \vdots \\ v \end{pmatrix}$ where $v = \left(\frac{1}{m_1}, \ldots, \frac{1}{m_k}\right)$ and $k = |\Sigma|$.

The *stationary distribution* of MC D is a probability vector x where $x = x \cdot \mathbf{P}$.

$$x_\sigma = \sum_{\tau \in \Sigma} x_\tau \cdot \mathbf{P}(\tau, \sigma) \quad \text{iff} \quad \underbrace{x_\sigma \cdot (1 - \mathbf{P}(\sigma,\sigma))}_{\text{outflow of } \sigma} = \underbrace{\sum_{\tau \neq \sigma} x_\tau \cdot \mathbf{P}(\tau,\sigma)}_{\text{inflow of } \gamma}$$

An irreducible, positive recurrent MC has a unique stationary distribution satisfying $x_\sigma = \frac{1}{m_\sigma}$ for every state σ. If $\mathbf{P}$ is ergodic, then each row of $\mathbf{P}^\infty$ equals the limiting (stationary) distribution.

2.3 Rewards

We can also attach *rewards* to states of Markov chains. The reward $r(\sigma)$ is the reward earned on leaving the state σ. We can also calculate a *cumulative reward for reachability*.

Definition: MC with rewards

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

Definition: Cumulative reward for reachability

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

If $\pi \notin \diamond G$, then $r_G(\pi) = 0$.

Definition: Expected reward for reachability

The *expected reward* for reachability until reaching $G \subseteq \Sigma$ from $\sigma \in \Sigma$ is:

$$\mathrm{ER}(\sigma, \diamond G) = \sum_{\pi \models \diamond G} \Pr(\hat{\pi}) \cdot r_G(\hat{\pi})$$

where $\hat{\pi} = \sigma_0 \dots \sigma_k$ is the shortest prefix of π such that $\sigma_k \in G$ and $\sigma_0 = \sigma$.

Definition: Conditional expected reward

Let $\mathrm{ER}(\sigma, \diamond G | \neg \diamond F) = \frac{\mathrm{ER}(\sigma, \diamond G \cap \neg \diamond F)}{\Pr(\neg \diamond F)}$ be the *conditional expected reward* until reaching G under the condition that no states in $F \subseteq \Sigma$ are visited.

3 Probabilistic GCL (pGCL)

Elementary ingredients for pGCL are

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

Definition: pGCL syntax

`skip`	empty statement
`diverge`	divergence
$x := E$	assignment
$x :\approx \mu$	random assignment
$P1;\ P2$	sequential composition
`if` (G) $\{P1\}$ `else` $\{P2\}$	choice
$P1\ [p]\ P2$	probabilistic choice
`while` (G) $\{P\}$	iteration

For random assignment $x :\approx \mu$ we evaluate the distribution expression μ in the curent program state s. Then we sample from the resulting distribution $\mu(s)$ yielding value v with probability $\mu(s)(v)$ and assign the value v to x.

To prove correctness of programs we use formal semantics. There are different kind of semantics. We use operational semantics of pGCL on Markov chains to model the execution behaviour of a program.

Definition: pGCL operational semantics

The behaviour of a pGCL program P is modelled by the MC $[\![P]\!]$:

- States are of the form
  - $\langle Q, s\rangle$ where Q is the remaining program to be executed,
  - or $s = \lightning$ for violation of an observation (`observe (G)`),
  - or $\langle \mathit{sink}\rangle$ for successful program termination.
- $s : \mathrm{Vars} \to \mathbb{Q}$ is a *variable valuation*.
- $\sigma_I = \langle P, s\rangle$ is the initial state where s fulfills the initial conditions.
- The transition relation $\to$ is the smallest relation satisfying the SOS rules below.

The *output* of the program P is the unique probability distribution given by $\lambda s.\ \Pr(s \models \Diamond\langle\downarrow, \cdot\rangle)$.

$$\langle\downarrow, s\rangle \to \langle \mathit{sink}\rangle \qquad \langle \mathit{sink}\rangle \to \langle \mathit{sink}\rangle$$

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

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

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

$$\langle P\ [p]\ Q, s\rangle \xrightarrow{p} \langle P, s\rangle \qquad \langle P\ [p]\ Q, s\rangle \xrightarrow{1-p} \langle Q, s\rangle$$

$$\frac{\langle P, s\rangle \xrightarrow{a} \langle P', s'\rangle}{\langle P;\ Q, s\rangle \xrightarrow{a} \langle P';\ Q, s'\rangle} \qquad \frac{\langle Q, s\rangle \xrightarrow{a} \langle Q', s'\rangle}{\langle\downarrow;\ Q, s\rangle \xrightarrow{a} \langle Q', s'\rangle}$$

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

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

cpGCL (section **??**)

$$\frac{s \models G}{\langle \texttt{observe}\ (G), s\rangle \to \langle\downarrow, s\rangle} \qquad \frac{s \not\models G}{\langle \texttt{observe}\ (G), s\rangle \to \langle\lightning\rangle}$$

$$\langle\lightning\rangle \to \langle \mathit{sink}\rangle \qquad \frac{\langle P, s\rangle \to \langle\lightning\rangle}{\langle P;\ Q, s\rangle \to \langle\lightning\rangle}$$

pGCL as defined above does not feature recursion, but we define two additional statements to define functions, which we call *processes*. We write $P = P_1$ to define a process P that executes program P_1 and write `call` P to call P. Introducing recursion does not increase expressive power.

4 Domain Theory

Definition: Partial order

A *partial order* $(D, \sqsubseteq)$ has a domain D and a relation $\sqsubseteq \subseteq D \times D$, where $\forall\, d_1, d_2, d_3 \in D$:

Reflexivity
$d_1 \sqsubseteq d_1$

Transitivity
$d_1 \sqsubseteq d_2 \wedge d_2 \sqsubseteq d_3 \;\Rightarrow\; d_1 \sqsubseteq s_3$

Antisymmetry
$d_1 \sqsubseteq d_2 \wedge d_2 \sqsubseteq d_1 \;\Rightarrow\; d_1 = d_2$

In pGCL semantics, loops are defined as fixed points of functions. We will need domain theory to prove existence of these fixed points and to approximate them.

A *complete lattice* is a partial order with upper bounds for all subsets (also called *supremum*). Equivalently, one can require lower bounds for all subsets (also called *infimum*).

A *chain* $S \subseteq D$ comprises only ordered elements: $\forall\, d_1, d_2 \in S\colon\quad d_1 \sqsubseteq d_2$ or $d_2 \sqsubseteq d_1$.

If $F : D \to D'$ is a monotonic function between complete lattices and $S \subseteq D$ is a chain in D, then $F(S) := \{\, F(d) \mid d \in S \,\}$ is a chain in D' and $\bigsqcup F(S) \sqsubseteq_{D'} F(\bigsqcup S)$.

A *(Scott-)continuous* function is a generalisation of the continuity we know from analysis to complete lattices. Every continuous function is monotonic.

Definition: Fixed point

d is a *fixed point* of $\Phi : D \to D$ if $\Phi(d) = d$.

Definition: (Least) Upper bound, (Greatest) Lower bound

Let $(D, \sqsubseteq)$ be a partial order with $S \subseteq D$.

1. $d \in D$ is an *upper bound* of S ($S \sqsubseteq d$) if $s \sqsubseteq d \;\; \forall\, s \in S$.
2. d is a *least upper bound of* S ($d = \bigsqcup S$) if $d \sqsubseteq d'$ for every upper bound d' of S.

Analogous definitions for *lower bound* and *greatest lower bound*.

Definition: Complete lattice

A *complete lattice* is a
- partial order $(D, \sqsubseteq)$,
- such that all $S \subseteq D$ have least upper bounds, or equivalently, greatest lower bounds.

The least element is $\bot := \bigsqcup \emptyset$.
The greatest element is $\top := \bigsqcap \emptyset$.

Definition: Monotonicity

Let $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ be partial orders. $\Phi : D \to D'$ is *monotonic* if $\forall\, d_1, d_2 \in D$: $d_1 \sqsubseteq d_2 \Rightarrow \Phi(d_1) \sqsubseteq' \Phi(d_2)$.

Definition: Scott continuity

Let $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ be complete lattices and $F : D \to D'$ monotonic. F is called *continuous* if, for every non-empty chain $S \subseteq D$, $F(\bigsqcup S) = \bigsqcup F(S)$.

Theorem: Kleene's fixpoint theorem

Let $(D, \sqsubseteq)$ be a complete lattice and $\Phi : D \to D$ continuous.
Then F has a *least fixed point* lfp F and a greatest fixed point gfp F:

$$\text{lfp } F := \sup_{n \in \mathbb{N}} F^n(\bot) \quad \text{and} \quad \text{gfp } F := \inf_{n \in \mathbb{N}} F^n(\top)$$

where $F^0(d) = d$ and $F^{n+1}(d) = F(F^n(d))$.

5 Probabilistic Weakest Preconditions

We assume program variables $x \in \text{Vars}$ are in $\mathbb{Q}$. We denote arithmetic expressions E over program variables and boolean expressions over program variables G.

Usually μ is a *distribution expression* $\mu : \Sigma \to \text{Dist}(\mathbb{Q})$ and $p : \Sigma \to [0,1] \cap \mathbb{Q}$.

The *expected value* of a random variable $f : X \to \mathbb{R}$ under distribution μ is defined by:

$$E_\mu(f) = \sum_{x \in X} f(x) \cdot \mu(x) = \int_X f \, \mathrm{d}\mu$$

Note that the expectation below is a random variable, and distinct from an expected value.

Definition: Predicate

A *predicate* F maps program states to Booleans, i.e. $F : \mathbb{S} \to \mathbb{B}$.

Let $\mathbb{P}$ denote the set of all predicates and $F \sqsubseteq G$ iff $F \Rightarrow G$.

Definition: Expectation

An *expectation* f maps program states to $\mathbb{R}_{\geq 0} \cup \{\infty\}$, i.e. $f : \mathbb{S} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$.

Let $\mathbb{E}$ denote the set of all expectations and $f \sqsubseteq g$ if and only if $f(s) \leq g(s)$ for all $s \in \mathbb{S}$.

Operations

$$(\lambda s.\, k)(k) = 0 \quad f[x := E](s) = \begin{cases} f(y) & \text{if } x \neq y \\ [\![E]\!]_s & \text{otherwise} \end{cases} \quad (c \cdot f)(s) = c \cdot f(s) \quad (f+g)(s) = f(s) + g(s)$$

$(\mathbb{E}, \sqsubseteq)$ is a complete lattice with the least element $\lambda s.\, 0 =: \mathbf{0}$. The supremum of a subset $S \subseteq \mathbb{E}$ is given by $\sup S = \sup_{f \in S} f$.

We define *predicate* and *expectation transformers* as total functions between predicates $\mathbb{P}$ or expectations $\mathbb{E}$ respectively.

Definition: Weakest pre-expectation

For probabilistic program P and $e, f \in \mathbb{E}$, the expectation transformer $\text{wp}(P, \cdot) : \mathbb{E} \to \mathbb{E}$ is defined by $\text{wp}(P, f) = e$ iff e maps each initial state s to the expected value of f after executing P on s.

$$\text{wp}(P, f) = \lambda s. \int_{\mathbb{S}} f \, \mathrm{d}P_s$$

where P_s is the distribution over the final states (reached on termination of P) when executing P on the initial state s.

Definition: Weakest liberal pre-expectation

The *weakest liberal precondition* is the expected value of f after executing P on s plus the probability that P diverges on s.

$$\text{wlp}(P, f) = \lambda s. \int_{\mathbb{S}} f \, \mathrm{d}P_s + \left(1 - \int_{\mathbb{S}} 1 \mathrm{d}P_s\right)$$

$\text{wlp}(P, \cdot) : \mathbb{E}_{\leq 1} \to \mathbb{E}_{\leq 1}$ is defined on *bounded expectations*, i.e. $\mathbb{E}_{\leq 1} = \{ f \in \mathbb{E} \mid f \sqsubseteq \mathbf{1} \}$.

5.1 Expectation Transformer Semantics of pGCL

P	$\mathrm{wp}(P,f)$	$\mathrm{wlp}(P,f)$
`skip`	f	
`diverge`	0	1
$x := E$	$f[x := E]$	
`observe` (G)	$[G] \cdot f$	
$x :\approx \mu$	$\lambda s.\ \int_{\mathbb{Q}} (\lambda v.\ f(s[x := v]))\ \mathrm{d}\mu_s$	
$P_1;\ P_2$	$\mathrm{wp}(P_1, \mathrm{wp}(P_2, f))$	$\mathrm{wlp}(P_1, \mathrm{wlp}(P_2, f))$
`if` (G) $\{P_1\}$ `else` $\{P_2\}$	$[G] \cdot \mathrm{wp}(P_1, f) + [\neg G] \cdot \mathrm{wp}(P_2, f)$	$[G] \cdot \mathrm{wlp}(P_1, f) + [\neg G] \cdot \mathrm{wlp}(P_2, f)$
$P_1\ [p]\ P_2$	$p \cdot \mathrm{wp}(P_1, f) + (1-p) \cdot \mathrm{wp}(P_2, f)$	$p \cdot \mathrm{wlp}(P_1, f) + (1-p) \cdot \mathrm{wlp}(P_2, f)$
`while` (G) $\{P\}$	$\mathrm{lfp}\ X.\ ([G] \cdot \mathrm{wp}(P, X) + [\neg G] \cdot f)$	$\mathrm{gfp}\ X.\ ([G] \cdot \mathrm{wlp}(P, X) + [\neg G] \cdot f)$

Theorem: Properties of wp

- Continuity:
  $\mathrm{wp}(P, \cdot)$ is continuous on $(\mathbb{E}, \sqsubseteq)$.
- Monotonicity:
  $f \leq g$ implies $\mathrm{wp}(P, f) \leq \mathrm{wp}(P, g)$.
- Feasibility:
  $f \leq \mathbf{k}$ implies $\mathrm{wp}(P, f) \leq \mathbf{k}$.
- Linearity: $\forall r \in \mathbb{R}_{\geq 0}$
  $\mathrm{wp}(P, r \cdot f + g) = r \cdot \mathrm{wp}(P, f) + \mathrm{wp}(P, g)$.
- Strictness:
  $\mathrm{wp}(P, \mathbf{0}) = \mathbf{0}$.

Theorem: Properties of wlp

- Continuity:
  $\mathrm{wlp}(P, \cdot)$ is continuous on $(\mathbb{E}_{\leq 1}, \sqsubseteq)$.
- Monotonicity:
  $f \leq g$ implies $\mathrm{wlp}(P, f) \leq \mathrm{wlp}(P, g)$.
- Superlinearity: $\forall r \in \mathbb{R}_{\geq 0}$
  $\mathrm{wlp}(P, r \cdot f + g) \leq r \cdot \mathrm{wlp}(P, f) + \mathrm{wlp}(P, g)$.
- Duality:
  $\mathrm{wlp}(P, f) = \mathrm{wp}(P, f) + (1 - \mathrm{wp}(P, \mathbf{1}))$
- Coincidence: for a.s.-terminating P
  $\mathrm{wlp}(P, f) = \mathrm{wp}(P, f)$
- Co-strictness:
  $\mathrm{wlp}(P, \mathbf{1}) = \mathbf{1}$.

Warning! Not all properties hold for programs with `observe (G)`.
E.g. co-strictness: $\mathrm{wlp}(\texttt{observe (false)}, \mathbf{1}) = \mathbf{0}$.

$\mathrm{wp}(P, \mathbf{1})$ = termination probability of program P.

Using Kleene's Fixpoint Theorem, we can calculate the fixed points for the loops.
For $\mathrm{lfp}\ \Phi = \sup_{n \in \mathbb{N}} \Phi^n(\bot)$ and for $\mathrm{gfp}\ \Psi = \inf_{n \in \mathbb{N}} \Psi^n(\top)$.

6 Loops and Proof Rules

Reasoning about loops is the hardest task in program verification. The weakest preconditions of loops are defined as fixed points and can be approximated iteratively. But recognizing patterns to

yield a closed formula and finding its fixed point is undecidable. We try to capture the effect of a loop by using a *loop invariant*.

We summarise the results from this section in the table below.

	lower bounds	**upper bounds**
wp	wp-ω-subinvariant	wp-superinvariant
wlp	wlp-subinvariant	wlp-ω-superinvariant

6.1 Predicate Invariants

We start with non-probabilistic loop invariants. A non-probabilistic loop invariant I for a postcondition F is a predicate that holds whenever the loop guard holds, that establishes the postcondition after termination, and also holds during execution of the loop body.

Definition: Loop invariant

A predicate $I \in \mathbb{P}$ is a loop invariant if:

- $G \Rightarrow I$,
- $\neg G \wedge I \Rightarrow F$, and
- $G \wedge I \Rightarrow \text{wlp}(P, I)$.

Why does this definition make sense? The theorem on the right assures us that a predicate invariant is always a sound approximation of the loop. We have shown this directly for probabilistic programs with (non-probabilistic) predicate invaraints.

$\Psi_{[F]}$ is the wlp-characteristic function of the probabilistic loop for postcondition $[F]$.

Theorem: Invariants make sense

For $I, F \in \mathbb{P}$ and probabilistic loop `while` (G) $\{P\}$ it holds:

$$\neg G \wedge I \Rightarrow F \text{ and } G \wedge I \Rightarrow wlp(P, I)$$
$$\text{iff}$$
$$[I] \sqsubseteq \Psi_{[F]}([I])$$

6.2 Probabilistic Invariants

Invariants that are not predicates are a bit harder: We use super- and subinvariants. We need this distinction because only wp-superinvariants are useful for upper bounds for wp, and wlp-subinvariants are useful for lower bounds for wlp. The other bounds require the respective ω-invariants.

Definition: Probabilistic invariants

Let Φ_f be the wp-characteristic function of $P' =$ `while` (G) $\{P\}$ with respect to post-expectation $f \in \mathbb{E}$ and let $I \in \mathbb{E}$.

- I is a *wp-superinvariant* of P' w.r.t. f iff $\Phi_f(I) \leq I$.
- I is a *wp-subinvariant* of P' w.r.t. f iff $I \leq \Phi_f(I)$.

Analogously defined for wlp with bounded expectations $\mathbb{E}_{\leq 1}$: Replace Φ_f by Ψ_f.

With Park's lemma, we can do (co-)induction to find bounds on weakest pre-expectations. Induction gives us **upper bounds for wp**:

$$\underbrace{\Phi_f(I) \leq I}_{\text{wp-superinvariant}} \text{ implies } \mathrm{wp}(\texttt{while}\ (G)\ \{P\}, f) \leq I$$

Co-induction gives us **lower bounds for wlp**:

$$\underbrace{I \leq \Psi_f(I)}_{\text{wlp-subinvariant}} \text{ implies } I \leq \mathrm{wlp}(\texttt{while}\ (G)\ \{P\}, f)$$

Theorem: Park's lemma

Let $(D, \sqsubseteq)$ be a complete lattice and $\Phi : D \to D$ continuous. Then:

$$\forall d \in D.\ \Phi(d) \sqsubseteq d \quad \text{implies} \quad \mathrm{lfp}\ \Phi \sqsubseteq d$$
$$\forall d \in D.\ d \sqsubseteq \Phi(d) \quad \text{implies} \quad d \sqsubseteq \mathrm{gfp}\ \Phi$$

Note that in general, versions of the equations above with $\sqsubseteq$ lfp and gfp $\sqsubseteq$ are not valid!

We can verify a loop invariant I by pushing it through the characteristic function of the loop once, i.e. $\Phi(I)$. For induction, we then only need to verify that $\Phi(I) \sqsubseteq I$, and for co-induction $I \sqsubseteq \Phi(I)$.

6.3 ω-invariants

ω-invariants give us the missing two bounds we did not get from Park's lemma. **wp-ω-subinvariants give lower bounds for** wp**, and** wlp-**ω-superinvariants give upper bounds for** wlp.

Definition: ω-invariants

Let $n \in \mathbb{N}, f \in \mathbb{E}$ and Φ_f be the wp-characteristic function of the loop $\texttt{while}\ (G)\ \{P\}$.

Monotonically increasing ($\sqsubseteq$) sequence $(I)_{n\in\mathbb{N}}$ is a wp-*ω-subinvariant* of the loop w.r.t. f iff

$$I_0 \leq \Phi_f(0) \text{ and } I_{n+1} \leq \Phi_f(I_n) \qquad \text{for all } n \in \mathbb{N}.$$

wlp-*ω-superinvariants* are defined similarly, where $(I)_{n\in\mathbb{N}} \in \mathbb{E}^{\mathbb{N}}_{\leq 1}$ is monotonically decreasing and Φ_f is replaced by Ψ_f.

As before, we have two soundness statements for ω-invariants.

Theorem: Bounds on loops using ω-invariants

1. Let $(I)_{n\in\mathbb{N}}$ be a wp-ω-subinvariant of $\texttt{while}\ (G)\ \{P\}$ w.r.t. $f \in \mathbb{E}$. Then:
$$\sup_{n\in\mathbb{N}} I_n \leq \mathrm{wp}(\texttt{while}\ (G)\ \{P\}, f)$$
2. Let $(I)_{n\in\mathbb{N}}$ be a wlp-ω-superinvariant of $\texttt{while}\ (G)\ \{P\}$ w.r.t. f. Then:
$$\mathrm{wlp}(\texttt{while}\ (G)\ \{P\}, f) \leq \inf_{n\in\mathbb{N}} I_n$$

To verify loops using ω-invariants, the following procedure can be used:

1. Find an appropriate ω-invariant $(I)_{n\in\mathbb{N}}$.
2. Check that $(I)_{n\in\mathbb{N}}$ is indeed an ω-invariant:
   a) Push I_n through the characteristic function

b) Check whether this took us above I_{n+1} (for wp) or below I_{n+1} (for wlp) in the partial order $\leq$.

3. Find the supremum (for wp) or the infimum (for wlp) of $(I)_{n\in\mathbb{N}}$ as a lower bound respective upper bound for wp/wlp.

7 Conditioning

We now add `observe (G)` statements. The idea is that only executions that satisfy all `observe (G)` during execution contribute to the resulting probability distribution. For this, we need new rules to the operational semantics to pGCL (already on **??**).

7.1 Conditional Expectations

Then we need normalisation: We want to divide by the probability of not satisfying the observe statements. We define cwp: as a tuple (f, g). The intuitive interpretation is that the resulting expected value is given by f/g.

Definition: Conditional expectation

A *conditional expectation* is a pair (f, g) with expectation $f \in \mathbb{E}$ and bounded expectation $g \in \mathbb{E}_{\leq 1}$.

Let $\mathbb{C} = \mathbb{E} \times \mathbb{E}_{\leq 1}$ denote the set of conditional expectations.

$$(f, g) \trianglelefteq (f', g') \text{ iff } f \leq f' \wedge g \geq g'$$

While we have explicit semantics for cwp, it's simplest to just always use the Decoupling property and calculate wp and wlp separately.

Theorem: Properties of cwp

Let $z, z' = (f, g), (f', g') \in \mathbb{C}$.

- Decoupling:
  $\text{cwp}(P, (f, g)) = (\text{wp}(P, f), \text{wlp}(P, g))$.
- Continuity:
  $\text{cwp}(P, z)$ is continuous on $(\mathbb{C}, \trianglelefteq)$.
- Monotonicity:
  $z \trianglelefteq z'$ implies $\text{cwp}(P, z) \trianglelefteq \text{cwp}(P, z')$.
- Linearity: $\forall r \in \mathbb{R}_{\geq 0}$
  $\text{cwp}(P, (r \cdot f + g, g')) = (r \cdot \text{wp}(P, f) + \text{wp}(P, g), \text{wlp}(P, g'))$.
- Strictness:
  $\text{cwp}(P, (\mathbf{0}, \mathbf{1})) = (\mathbf{0}, g)$ where $g = \text{wlp}(P, 1)$.
- Feasibility: If $\forall s \in \mathbb{S}.\ g(s) > 0 \Rightarrow f(s)/g(s)$ exists,
  $\forall s \in \mathbb{S}.\ g'(s) = 0$ implies $f'(s) = 0$
  where $\text{cwp}(P, (f, g)) = (f', g')$.

7.2 Program Transformations

It turns out `observe (G)` is entirely syntactic sugar: We can transform a program with `observe (G)` statements to one without while preserving semantics.

7.2.1 Rejection Sampling

The idea is to restart an infeasable run until all `observe (G)` statements are fulfilled.

We introduce a `flag` variable to signal violation of an `observe (G)` and new variables sx_i for every variable x_i in the original program. The sx_i variables are used to reset the variables x_i in case an

`observe (G)` is violated and we need to restart. Initially store the value of x_1 in xs_1 and reset x_1 to xs_1 in the beginning of every loop iteration.

We modify the original program *prog* to *mprog*:

$$\texttt{observe}\ (G) \rightsquigarrow flag := !G \,||\, flag$$
$$\texttt{diverge} \rightsquigarrow \texttt{if}\ (!flag)\ \{abort\}\ \texttt{else}\ \{\}$$
$$\texttt{while}\ (G)\ \{prog\} \rightsquigarrow \texttt{while}\ (G\ \&\&\ !flag)\ \{prog\}$$

The result is something like:

```
sx1 ,... := x1 ,...;
flag := true;
while(flag){
    flag := false;
    x1 ,... := sx1 ,...;
    mprog;
}
```

This transformation is correct: For a cpGCL program P and $\hat{P}$ the result of the above transformation we have:

$$\mathrm{cwp}(P, (f, 1)) = \mathrm{wp}(\hat{P}, f)$$

We can also go the other way: If a loop is *iid*, then $\mathrm{cwp}(\texttt{repeat}\ P\ \texttt{until}\ (G), (f, g))$ equals $\mathrm{cwp}(P; \texttt{observe}\ (G), (f, g))$.

iid means that G holds after P independently of the expected value of f after P.

Definition: iid-loop

A loop $\texttt{while}\ (G)\ \{P\}$ is *iid* iff for any expectation f:

$$\mathrm{wp}(P, [G] \cdot \mathrm{wp}(P, f)) = \mathrm{wp}(P, [G]) \cdot \mathrm{wp}(P, f).$$

7.2.2 Hoisting

A second option to deal with `observe`-statements is to use *hoisting*. It's a wee bit more complicated and the correctness theorem only holds for programs with at least one feasible run. Hoisting removes the `observe`-statements and transforms the probabilities accordingly.

Definition: Hoisting

$$\begin{aligned}
T(\texttt{skip}, f) &= (\texttt{skip}, f) \\
T(\texttt{diverge}, f) &= (\texttt{diverge}, 1) \\
T(x := E, f) &= (x := E, f[x := E]) \\
T(\texttt{observe}\ (G), f) &= (\texttt{skip}, [G] \cdot f) \\
T(P_1;\ P_2, f) &= (Q_1;\ Q_2, h) \text{ where } (Q_2, g) = T(P_2, f) \text{ and } (Q_1, h) = T(P_1, g) \\
T(\texttt{if}\ (G)\ \{P_1\}\ \texttt{else}\ \{P_2\}, f) &= (\texttt{if}\ (G)\ \{Q_1\}\ \texttt{else}\ \{Q_2\}, [G] \cdot g + [\neg G] \cdot h) \\
&\quad \text{where } (Q_1, g) = T(P_1, f) \text{ and } (Q_2, h) = T(P_2, f) \\
T(P_1\ [p]\ P_2, f) &= (Q_1\ [q]\ Q_2, p \cdot g + (1-p) \cdot h) \text{ where } q = \frac{p \cdot g}{p \cdot g + (1-p) \cdot h}, \\
&\quad (Q_1, g) = T(P_1, f) \text{ and } (Q_2, h) = T(P_2, f) \\
T(\texttt{while}\ (G)\ \{P\}, f) &= (\texttt{while}\ (G)\ \{Q\}, g) \text{ where } g = \mathrm{gfp}\ H \\
&\quad \text{with } H(h) = [G] \cdot (\pi_2 \odot T)(P, h) + [\neg G] \cdot f \text{ and } (Q, \cdot) = T(P, g)
\end{aligned}$$

Theorem: Correctness of hoisting

For any cpGCL program P with at least one feasible run and $f \in \mathbb{E}$:

$$\mathrm{cwp}(P, (f, 1)) = \mathrm{wp}(Q, f) \text{ with } T(P, 1) = (Q, h).$$

The component h represents the probability that P satisfies all its `observe`-statements.

8 Arithmetical Hierarchy

In this section, we rank different undecidable decision problems. Although each of them is undecidable, some are still harder than others. The *arithmetical hierarchy* classifies decision problems by the complexity of characterising formulas in first-order Peano arithmetic.

Definition: Arithmetical hierarchy

The *arithmetical hierarchy* consists of three types of classes: Σ_n, Π_n and Δ_n for each $n \in \mathbb{N}$.

Classes Σ_n, where R is a decidable relation:

$$\Sigma_n = \{A \mid A = \{x \mid \exists y_1 \forall y_2 \exists y_3 ... \forall / \exists y_n : (x, y_1, ..., y_n) \in R\}\}.$$

Classes Π_n, where R is a decidable relation:

$$\Pi_n = \{A \mid A = \{x \mid \forall y_1 \exists y_2 \forall y_3 ... \forall / \exists y_n : (x, y_1, ..., y_n) \in R\}\}$$

Classes Δ_n are defined as $\Delta_n = \Sigma_n \cap \Pi_n$.

Theorem: Elementary properties of the arithmetical hierarchy

- Δ_1 is the class of decidable problems.
- Classes Σ_n, Π_n and Δ_n are closed under conjunction and disjunction.
- Δ_n is closed under negation.
- The classes Σ_n and Π_n are complementary.
- There is a strict relation between classes: $\Sigma_n \subset \Delta_{n+1} \subset \Pi_{n+1}$ and $\Pi_n \subset \Delta_{n+1} \subset \Sigma_{n+1}$.
- If problem A is Σ_n-complete, then its complement is Π_n-complete, and vice versa.

As with polynomial complexities, we also have a notion of *completeness* of a problem in a certain complexity class here.

By *Davis' theorem*, we know that if problem A is Σ_n-complete, then $A \in \Sigma_n \setminus \Pi_n$, and vice versa.

Some simple examples for well-known decision problems follow.

Definition: Reducibility and completeness

$A \subseteq X$ is *reducible* to $B \subseteq X$ if there is a computable function $f : X \to X$ such that

$$\forall\, x \in X.\ x \in A \text{ iff } f(x) \in B.$$

Decision problem A is Γ_n*-hard* iff every $B \in \Gamma_n$ can be reduced to A.

A is Γ_n*-complete* iff $A \in \Gamma_n$ and A is Γ_n-hard.

Halting problem ($H \in \Sigma_1$, Σ_1-complete) Program P, state s. $(P,s) \in H$ iff: $\exists k \in \mathbb{N}, s' \in \mathbb{S}.P$ terminates on input s in k steps in state s'.

Universal halting problem ($UH \in \Pi_2$, Π_2-complete) Program P, state s. $P \in UH$ iff: $\forall s \in \mathbb{S}.(P,s) \in H$.

Co-finiteness problem ($COF \in \Sigma_3, \Sigma_3$-complete) Program P. $P \in COF$ iff: $\{ s \in \mathbb{S} \mid (P,s) \in H \}$ is co-finite where a subset A of X is *co-finite* if $X \setminus A$ is finite.

The important results are about the decision problems $LEXP$, $REXP$, EXP and $FEXP$.

$LEXP$ is Σ_1-complete: $\exists y.\ q < \sum_{k=0}^{y} \mathrm{wp}^{=k}(P,f)(s)$

$REXP$ is Σ_2-complete: $\exists \delta > 0.\ \forall y : q - \delta > \sum_{k=0}^{y} \mathrm{wp}^{=k}(P,f)(s)$

EXP is Π_2-complete. $FEXP$ is Σ_2-complete.

Note the δ in $REXP$ which is (intentionally) missing in $LEXP$.

Definition: The decision problems $LEXP$, $REXP$, EXP and $FEXP$

Let P be a pGCL program, $s \in \mathbb{S}$ a variable solution, $q \in \mathbb{Q}_{\geq 0}$ and $f : \mathbb{S} \to \mathbb{Q}_{\geq 0}$ a computable function. Then:

$$(P,s,f,q) \in LEXP \text{ iff } q < \mathrm{wp}(P,f)(s)$$
$$(P,s,f,q) \in REXP \text{ iff } q > \mathrm{wp}(P,f)(s)$$
$$(P,s,f,q) \in EXP \text{ iff } q = \mathrm{wp}(P,f)(s)$$
$$(P,s,f) \in FEXP \text{ iff } \mathrm{wp}(P,f)(s) < \infty$$

9 Almost-Sure Termination

With probabilistic programs, termination has a few different degrees:

- *Certain termination*: Literally every single program execution terminates.
- *Almost-sure termination*: Termination with probability one, but there may still be runs with infinite runtime.
  - + *Positive almost-sure termination*: Expected finite number of steps.
  - + *Null almost-sure termination*: Expected infinite number of steps.

AST and $UAST$ are both Π_2-complete. $UPAST$ is Π_3-complete. $PAST$ is Σ_2-complete: $\exists c.\ \forall l\ \mathrm{ert}^{\leq l}(P,s) < c$.

Definition: Decision problems AST and $UAST$

Let P be a program, $s \in \mathbb{S}$ a valuation.

$$(P,s) \in AST \text{ iff } \mathrm{wp}(P,1)(s) = 1$$
$$P \in UAST \text{ iff } \forall s \in \mathbb{S}.(P,s) \in AST$$

Definition: Decision problems $PAST$ and $UPAST$

Let P be a program, $s \in \mathbb{S}$ a valuation.

$$(P,s) \in PAST \text{ iff } \mathrm{ert}(P,s) < \infty$$
$$P \in UPAST \text{ iff } \forall s \in \mathbb{S}.(P,s) \in PAST$$

9.1 Proving Termination

Our aim is to prove termination by using a *variant function* (or *ranking function*) for the state space of a program that is monotonically decreasing in every loop iteration. The function deacreases with respect to a (strict) well-founded relation.

We know that every universally terminating (non-probabilistic) loop `while` (G) $\{P\}$ has a variant function.

Below are two theorems: The first is to prove positive almost-sure termination using a *ranking super-invariant*. The second theorem is for (positive/null) almost-sure termination and requires a variant I, a decrease probability function p and a decrease function d.

Definition: Well-founded relation

Let $(D, \sqsubset)$ be a strict partial order. The relation $\sqsubset$ is well-founded if there is no infinite sequence $d_1, d_2, d_3, \ldots$ with $d_i \in D$ such that $d_i \sqsubset d_{i+1}$ for all $i \in \mathbb{N}$.

Definition: Variant function

A *variant* function $V : \mathbb{S} \to \mathbb{R}$ for GCL-loop `while` (G) $\{P\}$ is a function that satisfies for every $s \in \mathbb{S}$:

- If $s \models G$, then the execution of P on s terminates in a state t with:

$$V(t) \leq V(s) - \varepsilon \text{ for some fixed } \varepsilon > 0$$

- If $V(s) \leq 0$, then $s \not\models G$.

Theorem: Proving positive almost-sure termination (PAST)

Let `while` (G) $\{P\}$ be a loop where P terminates universally certainly (P is loop-free), and let $I \in \mathbb{E}$ be a *ranking super-invariant* of the loop w.r.t. expectation 0, i.e., $I \leq \infty$ and for some constants ε and K with $0 < \varepsilon < K$ it holds:

1. $[\neg G] \cdot I \leq K$
2. $[G] \cdot K \leq [G] \cdot I + [\neg G]$
3. $\Phi(I) \leq [G] \cdot (I - \varepsilon)$

Then: `while` (G) $\{P\}$ terminates universally positively almost-surely.

Theorem: Proof rule for almost-sure termination (AST)

Let $I \in \mathbb{P}$, (variant) function $V : \mathbb{S} \to \mathbb{R}_{\geq 0}$, (probability) function $p : \mathbb{R}_{\geq 0} \to (0, 1]$ be antitone, (decrease) function $d : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ be antitone. If:

1. $[I]$ is a wp-subinvariant of `while` (G) $\{P\}$ w.r.t. $[I]$
2. $V = 0$ indicates termination, i.e. $[\neg G] = [V = 0]$
3. V is a super-invariant of `while` (G) $\{P\}$ w.r.t. V
4. V satisfies the progress condition

$$p \circ (V \cdot [G] \cdot [I]) \leq \lambda s.\ \mathrm{wp}(P, [V \leq V(s) - d(V(s))])(s)$$

Then: The loop `while` (G) $\{P\}$ terminates from any state s satisfying the invariant I, i.e.,

$$[I] \leq \mathrm{wp}(\texttt{while}\ (G)\ \{P\}, 1)$$

10 Expected Runtimes

Definition: Runtimes

A runtime $t : \mathrm{S} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$.

Let $\mathbb{T}$ denote the set of all runtimes.

In our time model we use a single time unit for *skip*, any assignment, evaluating a guard or probabilistic choice. Sequential composition does not take any time.

For every pGCL program P and input state s:

$$\underbrace{\mathrm{ert}(P, \mathbf{0})(s) < \infty}_{\text{positive a.s.-termination on } s} \text{ implies } \underbrace{\mathrm{wp}(P, \mathbf{1})(s) = \mathbf{1}}_{\text{a.s.-termination on } s}$$

And:

$$\underbrace{\mathrm{ert}(P, \mathbf{0}) < \infty}_{\text{universal positive a.s.-termination}} \text{ implies } \underbrace{\mathrm{wp}(P, \mathbf{1}) = \mathbf{1}}_{\text{universal a.s.-termination}}$$

Theorem: Properties of cwp

- Continuity:
  $\mathrm{ert}(P, t)$ is continuous on $(\mathbb{T}, \leq)$.
- Monotonicity:
  $t \leq t'$ implies $\mathrm{ert}(P, t) \leq \mathrm{ert}(P, t')$.
- Constant propagation:
  $\mathrm{ert}(P, k + t) = k + \mathrm{ert}(P, t)$.
- Preservation of ∞:
  $\mathrm{ert}(P, \infty) = \infty$.
- Connection to wp:
  $\mathrm{ert}(P, t) = \mathrm{ert}(P, 0) + \mathrm{wp}(P, t)$.
- Affinity:
  $\mathrm{ert}(P, a \cdot t + t') = \mathrm{ert}(P, 0) + a \cdot \mathrm{ert}(P, t) + \mathrm{ert}(P, t')$.

P	$\mathrm{ert}(P, t)$
`skip`	$1 + t$
`diverge`	∞
$x := E$	$1 + t[x := E]$
$x :\approx \mu$	$1 + \lambda s.\ \int_{\mathrm{Q}} (\lambda v.\ t(s[x := v]))\ \mathrm{d}\mu_s$
$P_1;\ P_2$	$\mathrm{ert}(P_1, \mathrm{ert}(P_2, t))$
`if` (G) $\{P_1\}$ `else` $\{P_2\}$	$1 + [G] \cdot \mathrm{ert}(P_1, t) + [\neg G] \cdot \mathrm{ert}(P_2, t)$
$P_1\ [p]\ P_2$	$1 + p \cdot \mathrm{ert}(P_1, t) + (1 - p) \cdot \mathrm{ert}(P_2, t)$
`while` (G) $\{P\}$	$\mathrm{lfp}\ X.(1 + [G] \cdot \mathrm{ert}(P, X) + [\neg G] \cdot t)$

We can also add rewards corresponding to the runtimes to the Markov chain for a program. State $\langle \downarrow, s \rangle$ gets reward $t(s)$. State $\langle \texttt{diverge}, s \rangle$ gets reward ∞. State $\langle P_1;\ P_2, s \rangle$ gets reward 0. All other states get reward 1. Then: $\mathrm{ert}(P, 0)(s) = ER^{[\![P]\!]}(s, \diamond sink)$.

Using the cwp-calculus, we were able to show that *PAST* is not compositional, i.e. for programs P_1, P_2 that are positive a.s.-terminating, $P_1;\ P_2$ is not necessarily also positive a.s.-terminating.

Similar to wp-calculus, we can also define *runtime-ω-subinvariants* for lower bounds and *runtime-superinvariants* for upper bounds (see **??**).

11 Bayesian Networks

A more traditional approach to statistical inference are *Bayesian networks*. A Bayesian network is a directed acyclic graph of nodes representing random variables. Edges represent causal relationships. Each random variable has an associated *conditional probability table* that maps all values of a node's k parents to a probability distribution. If there are no parents, then $\Theta_v : () \to \text{Dist}(D) \equiv \text{Dist}(D)$.

Definition: Bayesian network

A *Bayesian network* (BN) is a tuple $B = (V, E, \Theta)$ where

- (V, E) is a directed acyclic graph with
  - finite V in which each $v \in V$ represents a random variable with values from finite domain D, and
  - $(v, w) \in E$ represents the (causal) dependencies of w on v.
- For each vertex v with k parents, the function $\Theta_v : D^k \to \text{Dist}(D)$ is the *conditional probability table* of (the random variable represented by) vertex v.

$w \in V$ is a *parent* of $v \in V$ whenever $(w, v) \in E$.

The *joint probability function* of a Bayesian network gives a semantics to the network: By simple recursive multiplication of probabilities, we can calculate any joint probability of variables in the network.

Definition: Joint probability function of a BN

Let $B = (V, E, \Theta)$ be a BN, and $W \subseteq V$ be a downward closed set of vertices where $w \in W$ has value $\underline{w} \in D$. The (unique) *joint probability* function of BN B in which the nodes in W have values $\underline{W}$ equals:

$$\Pr(W = \underline{W}) = \prod_{w \in W} \Pr(w = \underline{w} \mid \mathit{parents}(w) = \underline{\mathit{parents}(w)}) = \prod_{w \in W} \Theta_w(\underline{\mathit{parents}(w)})(\underline{w}).$$

The *conditional probability distribution* of $W \subseteq V$ given observations on a set $O \sqsubseteq V$ is given by

$$\Pr(W = \underline{W} \mid O = \underline{O}) = \frac{\Pr(W = \underline{W} \wedge O = \underline{O})}{\Pr(O = \underline{O})}.$$

11.1 Conditional Independence

Two independent events may become dependent given some observation.

Definition: Conditional independence

Let X, Y, Z be (discrete) random variables. X is *conditionally independent* of Y given Z, denoted $I(X, Z, Y)$, whenever:

$$\Pr(X \wedge Y \mid Z) = \Pr(X \mid Z) \cdot \Pr(Y \mid Z) \text{ or } \Pr(Z) = 0.$$

Theorem: Graphoid axioms

Conditional independence satisfies the following axioms for disjoint sets of random variables W, X, Y, Z:

1. Symmetry: $I(X, Z, Y)$ iff $I(Y, Z, X)$
2. Decomposition: $I(X, Z, Y \cup W)$ implies ($I(X, Z, Y)$ and $I(X, Z, W)$)
3. Weak union: $I(X, Z, Y \cup W)$ implies $I(X, Z \cup Y, W)$
4. Contraction: ($I(X, Z, Y)$ and $I(X, Z \cup Y, W)$) implies $I(X, Z, Y \cup W)$
5. Triviality: $I(X, Z, \emptyset)$

D-separation[a] is a sufficient condition for conditional independence. Define all undirect paths in the DAG of the BN as a *pipe* and every vertex on a path as a *valve*. Valves are either *open* or *closed*. A pipe is *blocked* if at least one valve on the path is closed.

A valve v is closed for a variable set Z:

1. *Sequential*: $v \in Z$ is a child of one neighbour and a parent of the other neighbour.
2. *Divergent*: $v \in Z$ is a parent of both neighbours.
3. *Convergent*: neither v nor any of its directly reachable descendants are in Z.

The algorithm on the right is a polynomial time check for d-seperation: $dsep_G(X, Y, Z)$ iff X and Y are disconnected in $prune_{X,Y,Z}(G)$.

And since $dsep_G(X, Y, Z)$ implies $I(X, Y, Z)$, we can sometimes provide a guarantee that sets of nodes are conditionally independent in polynomial time.

[a] see also our Artifical Intelligence Panikzettel for another take at d-separation.

Definition: d-seperation

Let X, Y, Z be disjoint sets of vertices in the DAG G. X and Y are *d-seperated* by Z in G, denoted $dsep_G(X, Z, Y)$, iff:

Every (undirected) path between a vertex in X and a vertex in Y is blocked by some vertex in Z.

Algorithm: d-separation polynomial time

Input: DAG G and disjoint sets of vertices X, Y, Z.

Output: DAG $prune_{X,Y,Z}(G)$.

1. Repeat as long as possible:
   Eliminate any leaf vertex v from G with $v \notin X \cup Y \cup Z$.
2. Eliminate all edges emanating from vertices in Z.
3. Return remaining graph.

The complexity of inference on a BN is measured in terms of the *Markov blanket*, a degree of dependence in the BN. The less dependent the BN is the simpler is the probabilistic inference.

Definition: Markov blanket

The Markov blanket for a vertex v in a BN is the set ∂v of vertices composed of v, v's parents, its children, and its children's other parents.

The average Markov blanket of BN B is the average size of the Markov blanket of all its vertices, that is, $\frac{1}{|V|} \sum_{v \in V} |\partial v|$.

Every set of vertices in a BN is conditionally independent of v when conditioned on ∂v. Thus, for distinct vertices v and w:

$$\Pr(v \mid w \wedge \partial v) = \Pr(v \mid \partial v) \text{ which is equivalent to } I(\{v\}, \{w\}, Z).$$

11.2 Probabilistic Inference

The decision problems TI and STI are PP-complete.

PP (*Probabilistic Polynomial-Time*) is the class of decision problems solvable by a probabilistic Turing machine in polynomial time with an error probability $< \frac{1}{2}$.

We have shown PP-completeness by reducing MAJSAT, another PP-complete problem, to STI. And since STI is a special case of TI, MAJSAT can also be reduced to STI.

Definition: Probabilistic inference problems

Let B be a BN with set V of vertices, the *evidence* $E \subseteq V$ and the *questions* $Q \subseteq V$.

The *probabilistic inference problem* is to determine the conditional probability:

$$\Pr(Q = q \mid E = e) = \frac{\Pr(Q = q \wedge E = e)}{\Pr(E = e)}$$

Variants for probability $p \in \mathbb{Q} \cap [0, 1)$:

- *Threshold Inference* (TI):
  Is $\Pr(Q = q \mid E = e) > p$?
- *Simple TI* (STI):
  Is $\Pr(E = e) > p$?

BNs correspond to "simple" probabilistic programs as there is no "data-flow" between loop iterations. Such programs are called iid. If $\texttt{while}\ (G)\ \{P\}$ is iid for expectation f, it holds for every state s:

$$\text{wp}(\texttt{while}\ (G)\ \{P\}, f)(s) = [G](s) + \frac{\text{wp}(P, [\neg G] \cdot f)(s)}{1 - \text{wp}(P, [G])(s)} + [\neg G](s) \cdot f(s)$$

where we let $\frac{0}{0} = 0$.

We can also use our ert-calculus to calculate the expected sample time for a BN. This is very helpful to prove that there is no way the Windows printer troubleshooter will ever return a good result. Similar to the wp-rule for iid-loops above, we have for **a.s.-terminating iid loops**:

$$\text{ert}(\texttt{while}\ (G)\ \{P\}, t) = \mathbf{1} + \frac{\mathbf{1} + \text{ert}(P, [\neg G] \cdot t)}{1 - \text{wp}(P, [G])} + [\neg G](s) \cdot t.$$