PGF transformer semantics

Cheng Peng   Dantong Liu[1]

June 12, 2024

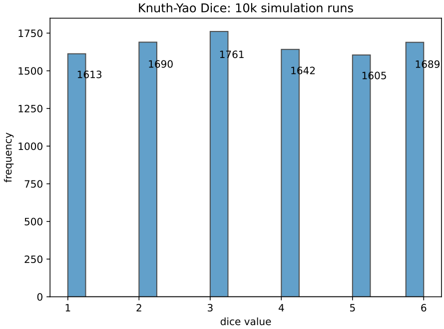
[1] External collaborator, BME undergrad at ShanghaiTech

Motivating Example: Knuth-Yao Dice I

```
nat s; nat die;

while (s < 7) {
          if(s = 0) { { s:=1 }[1/2]{ s:=2 } }
  else { if(s = 1) { { s:=3 }[1/2]{ s:=4 } }
  else { if(s = 2) { { s:=5 }[1/2]{ s:=6 } }
  else { if(s = 3) { { s:=1 }[1/2]{ s:=7; die:=1 } }
  else { if(s = 4) { { s:=7; die:=2 }[1/2]{ s:=7; die:=3 } }
  else { if(s = 5) { { s:=7; die:=4 }[1/2]{ s:=7; die:=5 } }
  else { if(s = 6) { { s:=2 }[1/2]{ s:=7; die:=6 } }
  else { skip } } } } } } } }
```

Motivating Example: Knuth-Yao Dice II




```
die := unif(1,6);
```

The main theme

PGF transform semantics based equivalence checking of pGCL programs.

1. A denotational semantics capturing the dynamics of all possible executions.
2. The specification will be given as a pGCL program.
3. Focus on a linear fragment of pGCL, called ReDiP.

Remarks

- Equivalence checking of pGCL is undecidable
- Linearity. No normalization is involved.

Contents in this section

Backgrounds

The rise of probabilistic programming

Randomness is ineluctable.

- Simulating the physics world.
- Building statistical models.
- Deriving efficient approximate algorithms.

The need to verify stochastic programs

Your property and life are at stake.

- Security-critical applications: cryptography systems
- Safety-critical applications: cyber-physics systems

A trust problem: simulation can demonstrate unreliability but not prove reliability.

Insufficiency of previous works

matured approaches

- Evaluating moments of variables[10]
- Deriving assertion violation probability[11]
- Establishing lower/upper bounds[1]

insufficiency

- Sensitivity to slight perturbation.
- Expressiveness: marginal/conditional. tail/concentration. parameter synthesis.

Our goal: to enable precise and versatile verification

Contents in this section

Overview of the solution

Input program A program in ReDiP, a fragment of pGCL

Specification program A loop-free ReDiP program: distribution of the final program state

Loop invariants Fixed point of loops, encoded as ReDiP programs.

Verifier check semantic equivalence, where the semantics of a program is

- A computational tree of configurations and transitions.
- A mapping from initial configuration to a distribution of output states.
- A (linear) transformer maps a PGF to another PGF.

Output True/False.

Contents in this section

$$
\begin{array}{rlr}
C \longrightarrow & \texttt{skip} & \text{(effectless program)} \\
& \mid \ \texttt{diverge} & \text{(freeze)} \\
& \mid \ x := E & \text{(assignment)} \\
& \mid \ x :\approx \mu & \text{(random assignment)} \\
& \mid \ C \fatsemi C & \text{(sequential composition)} \\
& \mid \ \texttt{if}\,(\varphi)\,\{C\}\,\texttt{else}\,\{C\} & \text{(conditional choice)} \\
& \mid \ \{C\}\,\square\,\{C\} & \text{(nondeterministic choice)} \\
& \mid \ \{C\}\,[p]\,\{C\} & \text{(probabilistic choice)} \\
& \mid \ \texttt{while}(\varphi)\{C\}\,, & \text{(while loop)}
\end{array}
$$

configuration $\mathcal{K} = \langle C, \sigma, n, \theta, \eta, q \rangle$.

transition relation $\vdash \subseteq \mathbb{K} \times \mathbb{K}$.

computational tree (initial configurations, reachable configuration, transitions).

$$\frac{}{\langle \texttt{skip}, \sigma, n, \theta, \eta, q\rangle \vdash \langle \downarrow, \sigma, n+1, \theta, \eta, q\rangle}\;(\text{skip})$$

$$\frac{}{\langle \texttt{diverge}, \sigma, n, \theta, \eta, q\rangle \vdash \langle \texttt{diverge}, \sigma, n+1, \theta, \eta, q\rangle}\;(\text{diverge})$$

$$\frac{v = \sigma(E)}{\langle x := E, \sigma, n, \theta, \eta, q\rangle \vdash \langle \downarrow, \sigma[x \mapsto v], n+1, \theta, \eta, q\rangle}\;(\text{assign})$$

$$\frac{\mu(\sigma)(v) = a > 0 \qquad \aleph^{-1}(v) = i}{\langle x :\approx \mu, \sigma, n, \theta, \eta, q\rangle \vdash \langle \downarrow, \sigma[x \mapsto v], n+1, \theta i, \eta, q \cdot a\rangle}\;(\text{rnd–assign})$$

$$\frac{\langle C_1, \sigma, n, \theta, \eta, q\rangle \vdash \langle C_1', \sigma', n+1, \theta', \eta', q'\rangle \qquad C_1' \neq \downarrow}{\langle C_1 \fatsemi C_2, \sigma, n, \theta, \eta, q\rangle \vdash \langle C_1' \fatsemi C_2, \sigma', n+1, \theta', \eta', q'\rangle}\;(\text{seq1})$$

$$\frac{\langle C_1, \sigma, n, \theta, \eta, q\rangle \vdash \langle \downarrow, \sigma', n+1, \theta', \eta', q'\rangle}{\langle C_1 \fatsemi C_2, \sigma, n, \theta, \eta, q\rangle \vdash \langle C_2, \sigma', n+1, \theta', \eta', q'\rangle}\;(\text{seq2})$$

$$\frac{\varphi(\sigma) = \mathsf{true}}{\langle \texttt{if}\,(\varphi)\,\{C_1\}\,\texttt{else}\,\{C_2\}, \sigma, n, \theta, \eta, q\rangle \vdash \langle C_1, \sigma, n+1, \theta, \eta, q\rangle}\;(\text{if1})$$

$$\frac{\varphi(\sigma) = \mathsf{false}}{\langle \texttt{if}\,(\varphi)\,\{C_1\}\,\texttt{else}\,\{C_2\}, \sigma, n, \theta, \eta, q\rangle \vdash \langle C_2, \sigma, n+1, \theta, \eta, q\rangle}\;(\text{if2})$$

$$\frac{}{\langle \{C_1\} \square \{C_2\}, \sigma, n, \theta, \eta, q\rangle \vdash \langle C_1, \sigma, n+1, \theta, \eta L, q\rangle}\;(\text{nondet1})$$

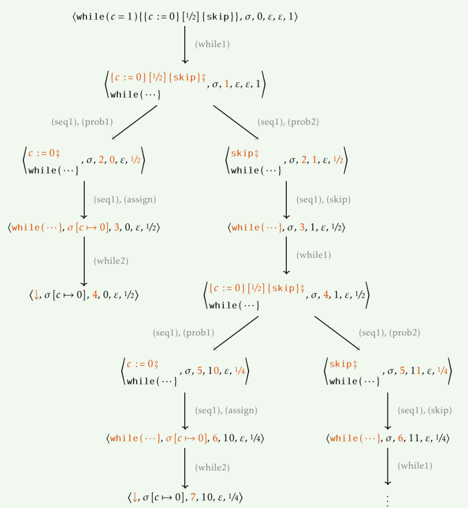
$$\frac{}{\langle \{C_1\} \square \{C_2\}, \sigma, n, \theta, \eta, q\rangle \vdash \langle C_2, \sigma, n+1, \theta, \eta R, q\rangle}\;(\text{nondet2})$$

$$\frac{p(\sigma) = a}{\langle \{C_1\}\,[p]\,\{C_2\}, \sigma, n, \theta, \eta, q\rangle \vdash \langle C_1, \sigma, n+1, \theta 0, \eta, q \cdot a\rangle}\;(\text{prob1})$$

$$\frac{p(\sigma) = a}{\langle \{C_1\}\,[p]\,\{C_2\}, \sigma, n, \theta, \eta, q\rangle \vdash \langle C_2, \sigma, n+1, \theta 1, \eta, q \cdot (1-a)\rangle}\;(\text{prob2})$$

$$\frac{\varphi(\sigma) = \mathsf{true}}{\langle \texttt{while}(\varphi)\{C\}, \sigma, n, \theta, \eta, q\rangle \vdash \langle C \fatsemi \texttt{while}(\varphi)\{C\}, \sigma, n+1, \theta, \eta, q\rangle}\;(\text{while1})$$

$$\frac{\varphi(\sigma) = \mathsf{false}}{\langle \texttt{while}(\varphi)\{C\}, \sigma, n, \theta, \eta, q\rangle \vdash \langle \downarrow, \sigma, n+1, \theta, \eta, q\rangle}\;(\text{while2})$$



A fragment of pGCL: ReDiP

- No divergence statement.
- All arithmetic expressions are linear/affine $a_0 + a_1 x_1 + a_2 x_2 \cdots a_n x_n$ where $a_i \in \mathbb{Z} \cup \text{Param}$ and $x_i \in \text{Var}$.
- All comparison expressions are rectangular $\langle expr \rangle \ op \ n$ where $op \in \{=, \neq, >, < . \leq, \geq\}$ and $n \in \mathbb{Z} \cup \text{Param}$.
- Special support for $x \bmod 2 = 0$.
- Special support for IID sampling.
- Sampling from a user-defined PGF.

Still quite expressive :).

Contents in this section

Notations

- vectors $\mathbf{x} = (x_1, x_2, \dots x_k)$ and $\sigma : \{1, \dots, k\} \mapsto \mathbb{N}$
- monomials $\mathbf{x}^\sigma = x_1^{\sigma(1)} x_2^{\sigma(2)} \dots x_k^{\sigma(k)} = \prod_i x_i^{\sigma(i)}$
- polynomial rings $\mathbb{R}[\![X]\!]$, $\mathbb{R}[\![X, Y]\!]$, and $\mathbb{R}[\![\mathbf{X}]\!]$

GF: sequences as formal power series

$$\sum_{n=0}^{\infty} a_n x^n \qquad \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} b_{n,m} x^n y^m \qquad \sum_{\sigma \in \mathbb{N}^k}^{\infty} f(\sigma) \mathbf{x}^\sigma$$

finite representation of infinite objects: closed-forms

GF $g(x)$	seq $[x^n]g$	parameters
$(1-ax)^k$	$\binom{k}{n} a^n$	$a, k \in \mathbb{R}^*$
$(1-x)^{-2}$	$n+1$	
x^k	$[n = k]$	$k \in \mathbb{N}^+$
e^{kx}	$k^n / n!$	$k \in \mathbb{R}^*$
$\ln(1-x)$	$\frac{-1}{n}$	$n \geq 1$

Generatingfuctionology[12] II

sequence manipulation as algebraic operations

sequence manipulation	gf algebra
a_{n-m}	$x^m f$
$a_n + b_n$	$f + g$
αa_n	αf
$\sum_k a_k b_{n-k}$	fg
na_n	$x\partial_x f$
a_{n-1}/n	$\int f \mathrm{d}x$

wikipedia/generating function/application/example 3

$$\begin{cases} U_n = 2V_{n-1} + U_{n-2} \\ V_n = U_{n-1} + V_{n-2} \\ U_0 = U_1 = V_0 = V_1 = 1 \end{cases} \implies \begin{cases} U(z) = 1 + 2zV(z) + z^2 U(z) \\ V(z) = zU(z) + z^2 U(z) = \frac{z}{1-z^2} U(z) \end{cases}$$

$$U(z) = \frac{1-z^2}{1-4z^2+z^4} = \frac{1}{3-\sqrt{3}} \cdot \frac{1}{1-\left(2+\sqrt{3}\right)z^2} + \frac{1}{3+\sqrt{3}} \cdot \frac{1}{1-\left(2-\sqrt{3}\right)z^2}$$

$$U_{2n+1} = 0 \qquad U_{2n} = \left\lceil \frac{(2+\sqrt{3})^n}{3-\sqrt{3}} \right\rceil$$

Probability Generating Functions I

Z-transform of PMF. $\mathbb{E}(s^X) = \sum_n p(n)s^n$ and $\mathbb{E}(s^X t^Y) = \sum_{n,m} p(n,m)s^n t^m$.

use of PGF

- represent distributions using generating functions
- operating random variable is manipulating PGFs
- working in finite closed-form

Random stopping sum

A $X_1, X_2, \ldots$ a sequence of iid variables with PGF $g_X(\cdot)$. Another independent random variable N with PGF $g_N(\cdot)$.

$$\begin{aligned}
\mathbb{E}_{N,\mathbf{X}}\left\{t^{\sum_{i=1}^{N} X_i}\right\} &= \mathbb{E}_N\left\{\mathbb{E}_{\mathbf{X}|N}\left[t^{\sum_{i=1}^{N} X_i} \mid N\right]\right\} = \mathbb{E}_N\left\{\mathbb{E}_{\mathbf{X}|N}\left[\prod_{i=1}^{N} t^{X_i} \mid N\right]\right\} \\
&= \mathbb{E}_N\left\{\prod_{i=1}^{N} \mathbb{E}_{\mathbf{X}|N}\left[t^{X_i} \mid N\right]\right\} = \mathbb{E}_N\left\{\prod_{i=1}^{N} \mathbb{E}_{\mathbf{X}}\left[t^{X_i}\right]\right\} = \mathbb{E}_N\left\{(g_X(t))^N\right\} \\
&= \sum_{n=0}^{\infty} (G_X(t))^n \Pr(N=n) = g_N(g_X(t))
\end{aligned}$$

PGF transformer semantics I

A denotational semantics

- Distribution of program states represented as PGFs.
- Program executions transforms PGFs.
- A program is a PGF transformer $[\![P]\!] : \mathrm{PGF} \to \mathrm{PGF}$

Suppose that the program state is $g = \mathbb{E}(s^X t^Y)$ where $X, Y \in \mathbb{N}$:

- `x := n` assignment: $g \mapsto g[s/1]s^n$

$$\mathbb{E}(s^X t^Y) \to s^n \mathbb{E}(1^X t^Y) = s^n \mathbb{E}(t^Y) = \mathbb{E}(s^n t^Y)$$

- `x := x+n` cumulation: $g \mapsto gs^n$

$$\mathbb{E}(s^X t^Y) \to s^n \mathbb{E}(s^X t^Y) = \mathbb{E}(s^{X+n} t^Y)$$

- `x := x-1` self-decrement: $g \mapsto (g - g[s/0])s^{-1} + g[s/0]$

$$\mathbb{E}(s^X t^Y) \to s^{-1} \mathbb{E}(s^X t^Y) = \mathbb{E}(s^{X-1} t^Y)$$

- `x := x+y` addition: $g \mapsto g[t/st]$

$$\mathbb{E}(s^X t^Y) \to \mathbb{E}(s^X (st)^Y) = \mathbb{E}(s^{X+Y} t^Y)$$

PGF transformer semantics II

- `x := D` samples from distribution: $g \mapsto g[s/1] \cdot [D](s)$ where the PGF of D is $[D](r)$

$$\mathbb{E}(s^X t^Y) \to \mathbb{E}(s^D)\mathbb{E}(1^X t^Y) = \mathbb{E}(s^D t^Y)$$

- `x := iid(D,y)` iid sampling: $g \mapsto g[s/1][t/t[D](s)]$

$$\mathbb{E}(s^X t^Y) \to \mathbb{E}(1^X t^Y ([D](s))^Y) = \sum_{0 \le m} ([D](s))^m t^m$$

- `if(x<n) {P} else {Q}` conditional branching: $g \mapsto [\![P]\!](g_{x<n}) + [\![Q]\!](g - g_{x<n})$ where

$$g_{x<n} = \sum_{x<n} \sum_{y} p(x,y) s^x t^y = \sum_{i<n} \frac{s^i}{i!} \left(\frac{\partial^i}{\partial s^i} g \right) [s/0]$$

- `P;Q` sequential composition: $g \mapsto [\![Q]\!]([\![P]\!](g))$

PGF transformer semantics III

make it sweet: syntactic sugar

`{P} [r] {Q}`	$g \mapsto r[\![P]\!](g) + (1-r)[\![Q]\!](g)$
`loop(n) {P}` `x := x-n`	$[\![P]\!]([\![P]\!] \cdots [\![P]\!](g))$ `loop(n) {x := x-1}`

`if(`$p \wedge q$`) {P} else {Q}`
`if(`$p \vee q$`) {P} else {Q}`
`if(`$\neg p$`) {P} else {Q}`
	`if(`p`){ if(`q`){P}else{Q} } else{Q}`
	`if(`p`){P} else { if(`q`){P} else{Q} }`
	`if(`p`) {Q} else {P}`

`if(`$x \leq n$`) P else Q`
`if(`$x > n$`) P else Q`
`if(`$x \geq n$`) P else Q`
`if(`$x = n$`) P else Q`
`if(`$x \neq n$`) P else Q`
	`if(`$x < n+1$`) P else Q`
	`if(`$x \leq n$`) Q else P`
	`if(`$x < n$`) Q else P`
	`if(`$x \leq n \wedge x \geq n$`) P else Q`
	`if(`$\neg(x = n)$`) P else Q`

$x \bmod 2 = 0$ $x \bmod 2 = 1$	$g \mapsto \frac{1}{2}(g[s/-s] + g)$ $g \mapsto \frac{1}{2}(g - g[s/-s])$

1. A while loop $\text{while}(\phi)\{P\}$
2. A infinitely nested braching tree

```
if(cond){
    body;
    if(cond){
        body;
        if(cond){
            body;
            ...
             ...
        }
    }
}
```

3. A least fixed point $\mu X.\text{if}(\phi)\{P; X\}$

rigorous and complicated theoretical derivation

Not quite correct.

- The PRODIGY verifier only checks invariant instead of automatically infer one.
- Defining a ω-complete partial order $(\leq, \mathsf{PGF})$
- For Almost Surely Termination (AST) programs: $[\![\text{if}(\phi)\{P; X\}]\!] \leq [\![X]\!]$
- For Universally AST (UAST) programs: $[\![\text{if}(\phi)\{P; X\}]\!] = [\![X]\!]$

Linearity

For a well-formed ReDiP program P, $[\![P]\!]$ is a linear transform

- `x := x+n` transformer $g \mapsto s^n g$

$$(af + bg) \mapsto (af + bg)s^n = a(s^n f) + b(s^n g)$$

- `x := x+y` transformer $g \mapsto g[t/st]$

$$(af + bg) \mapsto (af + bg)[t/st] = a(f[t/st]) + b(g[t/st])$$

- `x := 0` transformer $g \mapsto g[s/1]$

$$(af + bg)[t/1] = af[t/1] + bg[t/1]$$

- Other simple ReDiP program statements.
- By induction `P;Q` transformer $g \mapsto [\![Q]\!]([\![P]\!](g))$.

$$(af + bg) \mapsto [\![Q]\!](a[\![P]\!](f) + b[\![Q]\!](g)) = a[\![Q]\!]([\![P]\!](g)) + b[\![Q]\!]([\![P]\!](g))$$

- Other compositional statements `if(cond){P}else{Q}` and `while(cond){P}`.

Rational closed-form preservation

Rational closed-forms are amenable to computer algebra systems (e.g., sympy and GiNaC).

$$\frac{f}{g} = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} a_{ij} x^i y^j}{\sum_{i=0}^{n} \sum_{j=0}^{m} b_{ij} x^i y^j}$$

Rational closed-forms are closed under ReDiP transforms.

- `x := x+n` transformer $g \mapsto s^n g$

$$\frac{f}{h} \mapsto \frac{s^n f}{h}$$

- `x := x+y` transformer $g \mapsto g[t/st]$

$$\frac{f}{h} \mapsto \frac{f[t/st]}{h[t/st]}$$

- `x := 0` transformer $g \mapsto g[s/1]$

$$\frac{f}{h} \mapsto \frac{f[s/1]}{h[s/1]}$$

- Other simple ReDiP program statements.
- By induction `P;Q` transformer $g \mapsto [\![Q]\!]([\![P]\!](g))$.

$$f/h \mapsto [\![Q]\!]([\![P]\!](f/h)) = [\![Q]\!](f_1/h_1) = f_2/h_2$$

- Other compositional statements `if(cond){P}else{Q}` and `while(cond){P}`.

Equivalence verification

For two ReDiP programs P_1, P_2 whose state is $g(X_1, X_2, \ldots X_n) \in \mathrm{PGF}(\mathbb{N}^n)$

- Equivalence checking: for all valid generating function of distributions over $\mathbb{N}^n$.
  $\forall g \in \mathrm{PGF}(\mathbb{N}^n) \quad [\![P_1]\!](g) = [\![P_2]\!](g)$
- By linearity: for all point-mass distributions over $\mathbb{N}^n$
  $\forall (y_1, y_2, \ldots y_n) \in \mathbb{N}^n \quad [\![P_1]\!](X_1^{y_1} X_2^{y_2} \cdots X_n^{y_n}) = [\![P_2]\!](X_1^{y_1} X_2^{y_2} \cdots X_n^{y_n})$
- Ebbed all n-dim point-mass PGFs into a 2n-dim PGF $\delta(\mathbf{X}; \mathbf{U})$
  $[\![P_1]\!](\delta) = [\![P_2]\!](\delta)$ where

$$\delta = \sum_{y_1, y_2, \ldots y_n} (U_1 X_1)^{y_1} (U_2 X_2)^{y_2} \cdots (U_n X_n)^{y_n} = \prod_{i=1}^{n} \left(\sum_j (X_i U_i)^j \right) = \prod_{i=1}^{n} (1 - X_i U_i)^{-1}$$

Summary

1. Restricting syntax of pGCL to obtain ReDiP.
2. Recursively defining the PGF transformer semantics of ReDiP.
3. Handling unbounded loops with fixed-point induction.
4. Equivalence checking leveraging linearity

Contents in this section

Benchmark setup

Implementation

- About 500 nlocs of Python implementation.
- pGCL analysis using the probably package[9].
- Algebraic computation with sympy[7].

Benchmark

15 pairs of equivalent programs from PRODIGY[8].

Results I

Table: Results on the 15 test cases

test case	time
dep_bern.pgcl	6.81s
dueling_cowboys_parameter.pgcl	4.61s
geometric.pgcl	2.62s
geometric_observe.pgcl	2.72s
geometric_observe_parameter.pgcl	4.90s
geometric_parameter.pgcl	4.45s
geometric_shifted.pgcl	6.18s
ky_die.pgcl	18.5s
ky_die_2.pgcl	3.96s
ky_die_parameter.pgcl	26.0s
n_geometric.pgcl	2.37s
negative_binomial_parameter.pgcl	4.07s
random_walk.pgcl	4.46s
running_paper_example.pgcl	2.36s
trivial_iid.pgcl	1.85s

Results II

Table: scalability test: finite loop handling

iterations	time
1	2.75s
2	10.90s
3	19.36s
4	30.84s
5	43.52s
6	58.54s
7	72.68s
8	90.30s
9	105.35s
10	122.72s

- Check result: equivalent. 7.340888474005624 seconds.
- Variables $\{c \mapsto u_0 x_0, s \mapsto u_1 x_1, tmp \mapsto u_2 x_2\}$ parameters []
- $[\![P_1]\!](\delta)$: `(-u1*(u2*x2 - 1)*(u1*x0*sqrt(1 - x0**2) - u1*x0 + x0**2 - x0*(u1*sqrt(1 - x0**2) - u1 + x0) + 2*sqrt(1 - x0**2) - 2)/2 + (u2 - 1)*(u1*sqrt(1 - x0**2) - u1 + x0))/((u2 - 1)*(u0*x0 - 1)*(u2*x2 - 1)*(u1*sqrt(1 - x0**2) - u1 + x0))`
- $[\![P_2]\!](\delta)$: `-u1*(sqrt(1 - x0**2) - 1)/(u0*u1*u2*x0*sqrt(1 - x0**2) - u0*u1*u2*x0 - u0*u1*x0*sqrt(1 - x0**2) + u0*u1*x0 + u0*u2*x0**2 - u0*x0**2 - u1*u2*sqrt(1 - x0**2) + u1*u2 + u1*sqrt(1 - x0**2) - u1 - u2*x0 + x0) + 1/((u0*x0 - 1)*(u2*x2 - 1))`
- Comparison $[\![P_1]\!](\delta) - [\![P_2]\!](\delta) = 0$

```
nat s; nat c; nat tmp;

while(s > 0){
    {s := s+1} [1/2] {s := s-1}
    c := c+1
    tmp := 0
}
```

```
nat s; nat c; nat tmp;

if(s > 0){
    tmp := iid( ((1-(1-(c*c))^(1/2))/c), s
    c := c + tmp
    tmp := 0; s := 0
} else { skip }
```

Contents in this section

Conclusion

Our contributions:

- We define ReDiP a simple yet expressive subset of pGCL.
- We present the PGF transformers semantics of ReDiP.
- We explain the properties of PGF transformers.
- We designed an algorithm for ReDiP equivalence checking based on PGF transformer semantics.
- We tested the effectiveness and efficiency of such tool.

Limitations and future works I

conditional equivalence checking

From $[\![P_1]\!] = [\![P_2]\!]$ to $[\![P_1|\phi]\!] = [\![P_2|\phi]\!]$

```
-- prog: a program that handles input x such that P(x) is true
-- cond: the P(x) predicate
restrict :: (a -> Bool) -> (a -> b) -> (a -> Maybe b)
restrict cond prog input = if cond input
                           then Just (prog input)
                           else Nothing
-- check equivalence: (restrict p f) (restrict p g)
```

Obstacles:

- Support for conditional reasoning (`observe(..)` in pGCL).
- Expressing conditions besides rectangular ones, e.g., $x^2 + y^2 < 1$.

incorporating real-valued variables using MGF/CF

Discrete random variables likes Poisson(λ) and Binomial(n, p) are not enough. How to add support for $\mathcal{N}(\mu, \sigma^2)$ and $\beta(\alpha, \beta)$?
A natural generalization would be:

$$\mathbb{E}(e^{sX+tY}) \to e^{cs}\mathbb{E}(e^{0X+tY}) = \mathbb{E}(e^{sc+tY})$$
$$\mathbb{E}(e^{sX+tY}) \to \mathbb{E}(e^{sX+(t+s)Y}) = \mathbb{E}(e^{s(X+Y)tY})$$

Obstacles:

- Rational closed-form preservation no longer holds.
- Rectangular bound $x < c$ hard to express for MGF/CF

$$\mathbb{E}(t^{XI(X<n)}) = \sum_{x<n} p(x = n)t^n$$
$$\mathbb{E}(e^{tXI(X<n)}) = \int_{-\infty}^{c} f(x)e^{tx}\mathrm{d}x$$

Limitations and future works III

reactive programs

- The current approach works for UAST programs.
- Reactive programs are naturally non-terminating.
- Possible solution: bisimulation based equivalence checking.

```
OnInputSymbol(() => {
    alpha1, alpha2 = iid(std_normal, 2)
    x[n] = input();
    output(alpha1 * (x[n-1] - mu)
          + alpha2 * (x[n-2] - mu)^2);
    mu = (mu * n + x[n]) / (n + 1);
    n += 1;
})
```

```
OnInputSymbol(() => {
    alpha1, alpha2 = iid(std_normal, 2)
    output(alpha1 * (x[n-1] - mu)
          + alpha2 * (x[n-2]^2
             - 2*x[n]*mu + mu^2));
    x[n] = input();
    n += 1
    mu = (mu * (n-1) + x[n]) / n
})
```

references I

[1] Chakarov, Aleksandar and Sankaranarayanan, Sriram. "Probabilistic program analysis with martingales". In: Computer Aided Verification: 25th International Conference. Springer. 2013, pp. 511–526.

[2] Mingshuai Chen et al. Does a Program Yield the Right Distribution? Verifying Probabilistic Programs via Generating 2022. arXiv: 2205.01449 [cs.LO].

[3] He Jifeng, K. Seidel, and A. McIver. "Probabilistic models for the guarded command language". In: Science of Computer Programming 28.2 (1997). Formal Specifications: Foundations, Methods, Tools and Applications, pp. 171–192. ISSN: 0167-6423. DOI: https://doi.org/10.1016/S0167-6423(96)00019-6. URL: https://www.sciencedirect.com/science/article/pii/S0167642396000196.

[4] Lutz Klinkenberg et al. "Exact Bayesian Inference for Loopy Probabilistic Programs using Generating Functions". In: Proc. ACM Program. Lang. 8.OOPSLA1 (2024).

[5] Lutz Klinkenberg et al. "Exact Probabilistic Inference Using Generating Functions". In: LAFI. [Extended Abstract]. 2023.

[6] Dexter Kozen. "Semantics of probabilistic programs". In: 20th Annual Symposium on Foundations of Computer Science (sfcs 1979). IEEE. 1979, pp. 101–114.

[7] Aaron Meurer et al. "SymPy: symbolic computing in Python". In: PeerJ Computer Science 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. URL: https://doi.org/10.7717/peerj-cs.103.

[8] Chen Mingshuai et al. "Does a program yield the right distribution? Verifying probabilistic programs via generating functions". In: International Conference on Computer Aided Verification. Springer. 2022, pp. 79–101.

[9] Philipp Schröer, Lutz Klinkenberg, and Leo Mommers. Probably. https://github.com/Philipp15b/probably. 2021.

[10] Di Wang, Jan Hoffmann, and Thomas Reps. "Central moment analysis for cost accumulators in probabilistic programs". In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language 2021, pp. 559–573.

[11] Jinyi Wang et al. "Quantitative analysis of assertion violations in probabilistic programs". In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language PLDI 2021. Virtual, Canada: Association for Computing Machinery, 2021, pp. 1171–1186. ISBN: 9781450383912. DOI: 10.1145/3453483.3454102.

[12] Herbert S Wilf. generatingfunctionology. CRC press, 2005.

Q and A

Questions are welcomed.