PPCheck: Verifying the Equivalence of Probabilistic Programs

NUS National University of Singapore

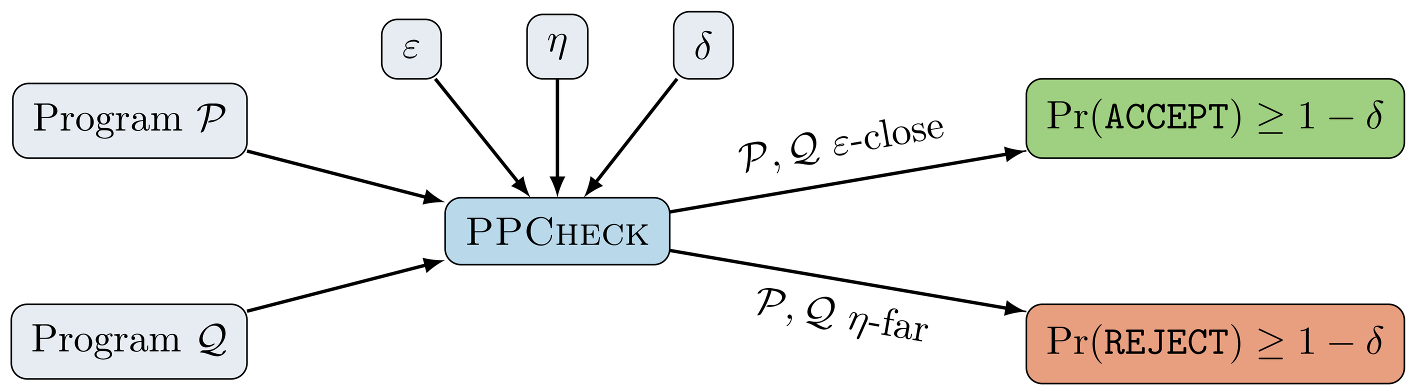
School of Computing

Alexandru Dinu[1] Sourav Chakraborty[2] Kuldeep Meel[1]

[1]National University of Singapore [2]Indian Statistical Institute

Our Contribution

- **Goal:** Test the equivalence of probabilistic programs.
- **Current setup:** Test the **approximate** equivalence of programs specifying **discrete** probability distributions.
- Offer guarantees in terms of *closeness* (tolerance parameter ε) and *farness* (intolerance parameter η), with error parameter δ.



Introducing PPCheck: Reducing Approximate Probabilistic Program Equivalence to a Distribution Testing problem.

Given two probabilistic programs $\mathcal{P}$ and $\mathcal{Q}$, with high probability ($\geq 1-\delta$), PPCheck will:

- `ACCEPT` $\mathcal{P}$ and $\mathcal{Q}$ as approximately equivalent if their underlying distributions $\mathbf{p}$ and $\mathbf{q}$ are ε-close.
- `REJECT` $\mathcal{P}$ and $\mathcal{Q}$ as approximately equivalent if their underlying distributions $\mathbf{p}$ and $\mathbf{q}$ are η-far.

Definition 1 (ε**-closeness**)**.** *Probabilistic programs $\mathcal{P}$ and $\mathcal{Q}$ (with distributions $\mathbf{p}$ and $\mathbf{q}$) are ε-close in multiplicative sense in ℓ_∞ distance if:*

$$(\forall)\ i \in [n],\ 1-\varepsilon \leq \frac{\mathbf{p}(i)}{\mathbf{q}(i)} \leq 1+\varepsilon$$

Definition 2 (η**-farness**)**.** *Probabilistic programs $\mathcal{P}$ and $\mathcal{Q}$ (with distributions $\mathbf{p}$ and $\mathbf{q}$) are η-far in ℓ_1 distance if:*

$$\sum_{i \in [n]} |\mathbf{p}(i) - \mathbf{q}(i)| \geq \eta$$

Introducing a preliminary **benchmark suite** for assessing the sanity of testing algorithms: collection of probabilistic programs written in WebPPL (Goodman-Stuhlmüller-14).

Example

```
// program_P.wppl
var 𝒫 = function () {
    var x = sample(RandomInteger({n: upper_bound}));
    var y = count_set_bits(x);
    return y;
}
var p = Infer({model: 𝒫, method: 'enumerate'})
```

```
// program_Q.wppl
var 𝒬 = function () {
    var x = sample(Gaussian({mu: mu, sigma: sigma}));
    var y = round(x);
    return y;
}
var q = Infer({model: 𝒬, method: 'MCMC'})
```

```
# tester.py
import pplib  # wrappers over WebPPL programs
import decide # decision algorithms

# can call sample(...) and condition(...) on p and q
n = 10
p = pplib.Distribution("program_P.wppl", upper_bound=2**n)
q = pplib.Distribution("program_Q.wppl", mu=n/2, sigma=sqrt(n)/2)

result = decide.ppcheck(p, q, eps=0.3, eta=0.85, delta=0.2)
```

```
# result
{
  "value": True, # ACCEPT
  "report": {
    "samples": 53_073_788,
    "log10": 7.724,
    "time": 87.532,
    "iterations": 38,
  }
}
```

Future Work

- Current limitation: PPCheck can only test discrete probability distributions and cannot handle non-termination.
- Explore **program-aware** testing – make PPCheck aware of program structure and properties.
- Incorporate the verification of inference engines as part of the benchmarks.

Central Idea

Algorithm 1 PPCheck($\mathcal{P}, \mathcal{Q}, \varepsilon, \eta, \delta$)

$t \leftarrow f(\varepsilon, \eta, \delta)$ // number of samples to draw from both programs (derived from the guarantees)
$S_1 \leftarrow t$ samples from $\mathcal{P}$
$S_2 \leftarrow t$ samples from $\mathcal{Q}$
for $(\sigma_1, \sigma_2) \in$ **zip**(S_1, S_2) **do**
  $\alpha_1 \leftarrow$ estimate $\dfrac{\mathbf{p}(\sigma_1)}{\mathbf{p}(\sigma_2)}$ by conditioning $\mathcal{P}$ on (σ_1, σ_2)
  $\alpha_2 \leftarrow$ estimate $\dfrac{\mathbf{q}(\sigma_1)}{\mathbf{q}(\sigma_2)}$ by conditioning $\mathcal{Q}$ on (σ_1, σ_2)
  if α_1 and α_2 are far from each other **then** // reject if $\frac{\alpha_1}{\alpha_2}$ or $\frac{\alpha_2}{\alpha_1}$ exceed a threshold value (derived from the guarantees)
    return `REJECT`
return `ACCEPT`

- If $\mathcal{P}$ and $\mathcal{Q}$ are η-far, we hope to find a witness pair of samples (σ_1, σ_2) on which the ratios $\dfrac{\mathbf{p}(\sigma_1)}{\mathbf{p}(\sigma_2)}$ and $\dfrac{\mathbf{q}(\sigma_1)}{\mathbf{q}(\sigma_2)}$ differ significantly.
- If $\mathcal{P}$ and $\mathcal{Q}$ are ε-close, then such witness does not exist, expecting the algorithm to accept all iterations.
- The estimate α for $\dfrac{\mathbf{p}(\sigma_1)}{\mathbf{p}(\sigma_2)}$ is computed as $\alpha = \dfrac{bias}{1-bias}$, where $bias = \dfrac{\mathbf{p}(\sigma_1)}{\mathbf{p}(\sigma_1)+\mathbf{p}(\sigma_2)}$ is obtained by leveraging **conditional sampling**.
- First, we compute a multiplicative estimate for *bias*, and if $bias > \dfrac{1}{2}$, an additive re-estimation is performed, to ensure that $1 - bias$ is a good estimate for $\dfrac{\mathbf{p}(\sigma_2)}{\mathbf{p}(\sigma_1)+\mathbf{p}(\sigma_2)}$.
- Conditioning a program on (σ_1, σ_2) essentially turns it into a (possibly) biased coin.

Experimental Results

Benchmark name	Support size	PPCheck		BFRSW	
		Result	Samples ($\log_{10}$)	Result	Samples ($\log_{10}$)
`discrete_normal_close_4`	5	ACCEPT	7.046	ACCEPT	6.834
`discrete_normal_close_6`	7	ACCEPT	7.655	ACCEPT	6.938
`discrete_normal_close_8`	9	ACCEPT	8.349	ACCEPT	7.015
`discrete_normal_close_10`	11	ACCEPT	7.676	ACCEPT	7.077
`uniform_eps_close_12`	2^{12}	ACCEPT	7.573	REJECT	8.887
`uniform_eps_close_14`	2^{14}	ACCEPT	7.583	REJECT	9.308
`uniform_eps_close_16`	2^{16}	ACCEPT	7.577	timeout	
`uniform_eps_close_18`	2^{18}	ACCEPT	7.585	timeout	
`discrete_normal_far_4`	5	REJECT	5.802	REJECT	5.976
`discrete_normal_far_6`	7	REJECT	5.755	REJECT	6.117
`discrete_normal_far_8`	9	REJECT	5.963	REJECT	6.219
`discrete_normal_far_10`	11	REJECT	5.794	REJECT	6.300
`uniform_eta_far_12`	2^{12}	REJECT	6.200	REJECT	8.887
`uniform_eta_far_14`	2^{14}	REJECT	6.172	REJECT	9.308
`uniform_eta_far_16`	2^{16}	REJECT	6.184	timeout	
`uniform_eta_far_18`	2^{18}	REJECT	6.114	timeout	

Table 1: Parameters: $\varepsilon = 0.3, \eta = 0.85, \delta = 0.2$, `timeout` $= 10^9$ samples / call.