Program Equivalence in a Typed Probabilistic Call-by-Need Functional Language

David Sabel and Manfred Schmidt-Schauß

WPTE 2022
31 July 2022, Haifa, Israel

Motivation and Goals

Probabilistic Programming + **Functional Programming** + **Call-by-Need Evaluation**

Probabilistic Programming

- programs express probabilistic models
- evaluation results in (multi-)distributions
- apply correct program transformations

Functional Programming

- declarative, high-level and generic programming
- clean (mathematical) definition
- equational reasoning

Call-by-Need Evaluation

- declarative: only needed bindings are evaluated
- efficient implementation of lazy evaluation
- in the probabilistic setting: different from call-by-name and call-by-value

A lot of related work on probabilistic lambda calculi with call-by-name or call-by-value evaluation
(see Ugo Dal Lago: *On Probabilistic Lambda-Calculi*, 2020)

→ **Investigate the semantics of a probabilistic call-by-need functional language**

Previous Work and This Work

Sabel, Schmidt-Schauß & Maio
PPDP'22 (to appear)

- analysis of an **untyped** call-by-need lambda calculus with probabilistic choice and recursive `let`
- contextual equivalence **observes the expected termination** in all contexts
- several proof techniques to show equivalences
- extension to data types and case-expressions

Sabel & Schmidt-Schauß
WPTE'22

- program equivalence in a **typed** probabilistic **PCF-like** language with call-by-need evaluation
- built-in **natural numbers**
- contextual equivalence observes expected termination **in contexts of type** nat only
- distribution-equivalence as other (more natural) notion of equality
- **main goal**: simpler characterisation of contextual equivalence (work in progress)

$ProbPCF^{need}$: Syntax

Syntax of Expressions and Types

Expressions: $s, t, r \in \mathit{Exp} ::= x \mid \lambda x.s \mid (s\ t) \mid \mathtt{fix}\ s \mid \mathtt{let}\ x = s\ \mathtt{in}\ t \mid (s \oplus t)$
$\mid \mathtt{if}\ r\ \mathtt{then}\ s\ \mathtt{else}\ t \mid \mathtt{pred}\ s \mid \mathtt{succ}\ s \mid n$ where $n \in \mathbb{N}_0$

Values: $v ::= n \mid \lambda x.s$

WHNFs: $LR[v]$ where $LR ::= [\cdot] \mid \mathtt{let}\ x = s\ \mathtt{in}\ LR$

Types: $\tau, \rho, \sigma \in \mathit{Typ} ::= \mathsf{nat} \mid \tau \to \rho$

Probabilistic choice $(s \oplus t)$ randomly evaluates to s or t (both with probability 0.5)

Type checking: standard monomorphic type system, $e \in \mathit{Exp}$ is well-typed iff $e : \tau$

$$\frac{s : \tau \to \rho, t : \tau}{(s\ t) : \rho} \qquad \frac{t : \tau, s : \rho, \rho = \Gamma(x)}{(\mathtt{let}\ x = s\ \mathtt{in}\ t) : \tau} \qquad \frac{s : \rho, t : \rho}{(s \oplus t) : \rho} \qquad \frac{s : \mathsf{nat}}{(\mathtt{succ}\ s) : \mathsf{nat}} \qquad \dots$$

Examples

$(1 \oplus 2) \oplus (3 \oplus 4)$

- evaluates to 1,2,3,4, each with probability 0.25
- represents the distribution $\{(0.25, 1), (0.25, 2), (0.25, 3), (0.25, 4)\}$

Examples

$(1 \oplus 2) \oplus (3 \oplus 4)$

- evaluates to 1,2,3,4, each with probability 0.25
- represents the distribution $\{(0.25, 1), (0.25, 2), (0.25, 3), (0.25, 4)\}$

$(v_1 \oplus v_2) \oplus (v_3 \oplus v_4)$

- represents the **multi**-distribution $\{(0.25, v_1), (0.25, v_2), (0.25, v_3), (0.25, v_4)\}$
- the corresponding distribution depends on
  - whether $v_i = v_j$ and
  - on the interpretation =

Examples

$(1 \oplus 2) \oplus (3 \oplus 4)$

- evaluates to 1,2,3,4, each with probability 0.25
- represents the distribution $\{(0.25, 1), (0.25, 2), (0.25, 3), (0.25, 4)\}$

$(v_1 \oplus v_2) \oplus (v_3 \oplus v_4)$

- represents the **multi**-distribution $\{(0.25, v_1), (0.25, v_2), (0.25, v_3), (0.25, v_4)\}$
- the corresponding distribution depends on
  - whether $v_i = v_j$ and
  - on the interpretation =

$\texttt{fix}\ (\lambda u.(0 \oplus \texttt{succ}\ u))$

- evaluates to 0 or recursively proceeds with the successor
- generates the distribution

$$\left\{\left(\frac{1}{2}, 0\right), \left(\frac{1}{4}, 1\right), \left(\frac{1}{8}, 2\right), \left(\frac{1}{16}, 3\right), \ldots\right\} = \left\{\left(\frac{1}{2^{i+1}}, i\right) \mid i \in \mathbb{N}_0\right\}$$

Call-by-Name, Call-by-Value, Call-By-Need

possible evaluation results

	call-by-name	**call-by-need**	**call-by-value**
$(\lambda y.1)\ \bot$	1	1	diverges
$(\lambda\ x.x + x)\ (1\ \oplus\ 2)$	2,3,4	2 and 4	2 and 4

$ProbPCF^{need}$: Operational Semantics

(sr,lbeta) $R[(\lambda x.s)\ t] \xrightarrow{sr} R[\mathtt{let}\ x = t\ \mathtt{in}\ s]$

(sr,cp) $\mathrm{LR}[\mathtt{let}\ x = v\ \mathtt{in}\ R[x]] \xrightarrow{sr} \mathrm{LR}[\mathtt{let}\ x = v\ \mathtt{in}\ R[v]]$

(sr,probl) $R[s \oplus t] \xrightarrow{sr} R[s]$

(sr,probr) $R[s \oplus t] \xrightarrow{sr} R[t]$

"prob-reductions"

(sr,succ) $R[\mathtt{succ}\ n] \xrightarrow{sr} R[n+1]$

... ...

where **reduction contexts** R are

$$\begin{aligned}
R &::= \mathrm{LR}[A] \mid \mathrm{LR}[\mathtt{let}\ x = A\ \mathtt{in}\ R[x]] \\
A &::= [\cdot] \mid (A\ s) \mid \mathtt{if}\ A\ \mathtt{then}\ s\ \mathtt{else}\ t \mid \mathtt{pred}\ A \mid \mathtt{succ}\ A \mid \mathtt{fix}\ A \\
\mathrm{LR} &::= [\cdot] \mid \mathtt{let}\ x = s\ \mathtt{in}\ \mathrm{LR}
\end{aligned}$$

- for $\xrightarrow{sr}$, redexes are unique and $\xrightarrow{sr}$ is only non-deterministic for prob-reductions
- type safety (progress and type preservation)

Tracking Probabilities

Weighted expression (p, s) with rational number $p \in (0, 1]$ and expression s

Weighted standard reduction step $\xrightarrow{wsr}$

$$(p, s) \xrightarrow{wsr,a} \begin{cases} (p, t) & \text{iff } s \xrightarrow{sr,a} t \text{ and } a \notin \{probl, probr\} \\ \left(\frac{p}{2}, t\right) & \text{iff } s \xrightarrow{sr,a} t \text{ and } a \in \{probl, probr\} \end{cases}$$

$\xrightarrow{wsr,*}$ denotes the reflexive-transitive closure of $\xrightarrow{wsr}$

Evaluation

An **evaluation** of (p, s) is a sequence $(p, s) \xrightarrow{wsr,*} (q, t)$ where t is a WHNF.
Eval(p, s) = set of all evaluations starting with (p, s)
Notation: $(p, s) \lightning_L (q, t) \in$ *Eval*(p, s) where L = sequence of labels of prob-reductions

Expected Convergence

Expected convergence

$$\mathrm{ExCv}(s) = \sum_{(1,s)\Downarrow_L (q,t)\,\in\, \mathit{Eval}(1,s)} q.$$

"= probability that evaluation of s ends with a WHNF"

Expected value convergence

$$\mathrm{ExVCv}(s,n) = \sum_{(1,s)\Downarrow_L (q,\mathrm{LR}[n])\,\in\, \mathit{Eval}(1,s)} q,$$

"= probability that evaluations of s ends with number n"

Lemma

For all expressions s : nat: $\mathrm{ExCv}(s) = \sum_{i=0}^{\infty} \mathrm{ExVCv}(s,i)$

Contextual Equivalence

Contextual Preorder and Equivalence

For equally typed expressions $s, t : \sigma$:

- **contextual preorder** $s \leq_c t$ iff $\forall C[\cdot_\sigma]$: nat: $\mathrm{ExCv}(C[s]) \leq \mathrm{ExCv}(C[t])$

  "in any context: t converges at least as often as s"

- **contextual equivalence** $s \sim_c t$ iff $s \leq_c t \wedge t \leq_c s$

Refuting equivalences requires one context acting as counter-example

Example: $(2 \oplus (3 \oplus 4)) \not\sim_c ((2 \oplus 3) \oplus 4)$:

- $C = \texttt{if pred (pred} [\cdot_{\mathsf{nat}}]) \texttt{ then } 0 \texttt{ else } \bot$ (where $\bot = \texttt{fix } \lambda x.x$)
- $\mathrm{ExCv}(C[(2 \oplus (3 \oplus 4))]) = 0.5$ but $\mathrm{ExCv}(C[((2 \oplus 3) \oplus 4)]) = 0.25$

Proving equivalences is **harder** due to the quantification over all contexts.

Expected Convergence with Bounded Number of Prob-Reductions

Expected convergence of s with bound k = number prob-reductions

$$\mathrm{ExCv}(s,k) = \sum_{\substack{(1,s)\lightning_L(q,t) \in \mathit{Eval}(1,s), \\ |L| \leq k}} q$$

→ allows inductive proofs and constructions on the number k, and in the limit, differences in k do not matter:

Lemma

Let $s, t : \tau$ such that $\forall k \geq 0 : \exists d : \mathrm{ExCv}(s,k) \leq \mathrm{ExCv}(t, k+d)$.
Then $\mathrm{ExCv}(s) \leq \mathrm{ExCv}(t)$.

Context Lemma

Context Lemma

Let $N \geq 0$, for $1 \leq i \leq N$: $s_i, t_i : \sigma$, such that $\forall k \geq 0$, $\forall$reduction contexts $R[\cdot_\sigma]$: nat there exists $d \geq 0 : \mathrm{ExCv}(R[s_i], k) \leq \mathrm{ExCv}(R[t_i], k + d)$.
Let $C[\cdot_{1,\sigma}, \ldots, \cdot_{N,\sigma}]$: nat be a multicontext with N holes of type σ.
Then the inequation $\mathrm{ExCv}(C[s_1, \ldots, s_N]) \leq \mathrm{ExCv}(C[t_1, \ldots, t_N])$ holds.

- Instantiation for $N = 1$:

If $\forall k \geq 0$, $R[\cdot_\sigma]$: nat, $\exists : d \geq 0 : \mathrm{ExCv}(R[s], k) \leq \mathrm{ExCv}(R[t], k + d)$, then $s \leq_c t$.

- Valuable proof tool to show contextual equivalences

Program Transformations

A program transformation T is a binary relation of equally typed expressions.

T is **correct** iff $\xrightarrow{T} \subseteq \sim_c$

Some Correct Program Transformations

(fix)	$\texttt{fix}\ \lambda x.s \to (\lambda x.s)\ (\texttt{fix}\ \lambda x.s)$	*(llet)*	$\texttt{let}\ x = (\texttt{let}\ y = s\ \texttt{in}\ t)\ \texttt{in}\ r$
(lbeta)	$((\lambda x.s)\ t) \to \texttt{let}\ x = t\ \texttt{in}\ s$		$\to \texttt{let}\ y = s, x = t\ \texttt{in}\ r$
(succ)	$(\texttt{succ}\ n) \to n+1$	*(cp)*	$\texttt{let}\ x = v\ \texttt{in}\ C[x]$
(pred)	$(\texttt{pred}\ n) \to \max(0, n-1)$		$\to \texttt{let}\ x = v\ \texttt{in}\ C[v]$
(if-then)	$\texttt{if}\ 0\ \texttt{then}\ s\ \texttt{else}\ t \to s$	*(gc)*	$\texttt{let}\ x = s\ \texttt{in}\ t \to t$ if $x \notin FV(t)$
(if-else)	$\texttt{if}\ n\ \texttt{then}\ s\ \texttt{else}\ t \to t$ if $n \neq 0$	*(⊕-id)*	$(s \oplus s) \to s$
(lflata)	$A^1[(\texttt{let}\ x = s\ \texttt{in}\ t)]$	*(⊕-comm)*	$(s \oplus t) \to (t \oplus s)$
	$\to \texttt{let}\ x = s\ \texttt{in}\ A^1[t]$	*(⊕-distr)*	$(r \oplus (s \oplus t)) \to ((r \oplus s) \oplus (r \oplus t))$

- green transformations can be shown correct by the context lemma.
- red transformations require other techniques (e.g. the diagram method).

Distribution-Equivalence

Distribution-Equivalence

Let $s, t :$ nat be two closed expressions. Then s and t are **distribution-equivalent**, $s \sim_d t$, iff for all $n \in \mathbb{N}_0$: $\textsc{ExVCv}(s,n) = \textsc{ExVCv}(t,n)$.

Example:

- $(0 \oplus 1) + 2 * (0 \oplus 1)$
  "tossing two coins, one for each digit of a binary number of length 2"
- $(0 \oplus 1) \oplus (2 \oplus 3)$
  "throwing a fair 4-sided dice"
- both expressions produce the same distribution
  $\{(0.25, 0), (0.25, 1), (0.25, 2), (0.25, 3)\}$

Further Examples

$\texttt{fix}\ (\lambda u.(0 \oplus \texttt{succ}\ u))$ generates the distribution

$$\left\{\left(\frac{1}{2},0\right),\left(\frac{1}{4},1\right),\left(\frac{1}{8},2\right),\left(\frac{1}{16},3\right),\ldots\right\} = \left\{\left(\frac{1}{2^{i+1}},i\right) \mid i \in \mathbb{N}_0\right\}$$

$(\texttt{fix}\ (\lambda f.\lambda u.u \oplus (f\ (\texttt{succ}\ u))))\ 0$ generates the same distribution

$$\left\{\left(\frac{1}{2},0\right),\left(\frac{1}{4},1\right),\left(\frac{1}{8},2\right),\left(\frac{1}{16},3\right),\ldots\right\} = \left\{\left(\frac{1}{2^{i+1}},i\right) \mid i \in \mathbb{N}_0\right\}$$

$(\texttt{fix}\ (\lambda f.\lambda u.u \oplus (f\ (u+2))))\ (0 \oplus 1)$ generates a different distribution

$$\left\{\left(\frac{1}{4},0\right),\left(\frac{1}{4},1\right),\left(\frac{1}{8},2\right),\left(\frac{1}{8},3\right),\left(\frac{1}{16},4\right),\left(\frac{1}{16},5\right),\ldots\right\}$$

Distribution-Equivalence vs. Contextual Equivalence

Contextual equivalence implies distribution-equivalence:

Theorem

Let $s, t : \sigma$ be two typed expressions with $s \sim_c t$.
Then for any context $C[\cdot_\sigma] : \mathsf{nat}$, $C[s] \sim_d C[t]$.

Reverse direction:

Conjecture

If the distribution of closed expressions $s, t : \mathsf{nat}$ in the empty context is the same (i.e. $s \sim_d t$), then s, t are contextually equivalent.

Proof: work in progress (maybe by applicative bisimulation)

Conclusion & Future Work

Conclusions

- Analysis of a typed call-by-need functional language with fair probabilistic choice
- Two program equivalences:
  - Contextual Equivalence observes expected convergence in all contexts
  - Distribution-equivalence: evaluation leads to the same probability distribution

Future work

- Work out proofs
- Proof of the conjecture
- Practical examples
- Extensions of the language: data constructors, case, ...

Thank You!