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

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

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

- 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

- 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

- 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)
**Syntax of Expressions and Types**

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

**Values:**
\[ v ::= n \mid \lambda x.s \]

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

**Types:**
\[ \tau, \rho, \sigma \in \text{Typ} ::= \text{nat} \mid \tau \rightarrow \rho \]

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

**Type checking:** standard monomorphic type system, \(e \in \text{Exp}\) is well-typed iff \(e : \tau\)
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

\[(1 ⊕ 2) ⊕ (3 ⊕ 4)\]

- 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))\}

\[(v_1 ⊕ v_2) ⊕ (v_3 ⊕ v_4)\]

- 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 = \text{fix} \(\lambda u. (0 \oplus \text{succ} \ u)\)
- evaluates to 0 or recursively proceeds with the successor
- generates the distribution

\[
\left\{\left(\frac{1}{2}, 0\right), \left(\frac{1}{4}, 1\right), \left(\frac{1}{8}, 2\right), \left(\frac{1}{16}, 3\right), \ldots\right\} = \left\{\left(\frac{1}{2i+1}, i\right) \mid i \in \mathbb{N}_0\right\}
\]
### Call-by-Name, Call-by-Value, Call-By-Need

<table>
<thead>
<tr>
<th></th>
<th>Call-by-name</th>
<th>Call-by-need</th>
<th>Call-by-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\lambda y.1) \perp$</td>
<td>$1$</td>
<td>$1$</td>
<td>diverges</td>
</tr>
<tr>
<td>$(\lambda x.x + x) (1 \oplus 2)$</td>
<td>$2,3,4$</td>
<td>$2$ and $4$</td>
<td>$2$ and $4$</td>
</tr>
</tbody>
</table>

**Possible Evaluation Results**
**ProbPCF** \(_{need}\): Operational Semantics

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

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

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

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

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

\[ \ldots \ldots \]

where reduction contexts \( R \) are

\[ R ::= LR[A] | LR[\text{let } x = A \text{ in } R[x]] \]

\[ A ::= [] | (A s) | \text{if } A \text{ then } s \text{ else } t | \text{pred } A | \text{succ } A | \text{fix } A \]

\[ LR ::= [] | \text{let } x = s \text{ in } LR \]

- 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 \(\xrightarrow{wsr}\)

\[
(p, s) \xrightarrow{wsr,a} \begin{cases} 
(p, t) & \text{iff } s \xrightarrow{sr,a} t \text{ and } a \notin \{probl, probr\} \\
(p/2, t) & \text{iff } s \xrightarrow{sr,a} t \text{ and } a \in \{probl, probr\}
\end{cases}
\]

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

Evaluation

An evaluation of \((p, s)\) is a sequence \((p, s) \xrightarrow{wsr,*} (q, t)\) where \(t\) is a WHNF.

\(\text{Eval}(p, s) = \text{set of all evaluations starting with } (p, s)\)

Notation: \((p, s)\xrightarrow{\downarrow L}(q, t) \in \text{Eval}(p, s)\) where \(L = \text{sequence of labels of prob-reductions}\)
**Expected Convergence**

**Expected convergence**

\[ \text{ExCv}(s) = \sum_{(1,s) \downarrow_L (q,t) \in \text{Eval}(1,s)} q. \]

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

**Expected value convergence**

\[ \text{ExVCv}(s,n) = \sum_{(1,s) \downarrow_L (q, LR[n]) \in \text{Eval}(1,s)} q, \]

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

**Lemma**

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

**Contextual Preorder and Equivalence**

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

- **contextual preorder** $s \leq_c t$ iff $\forall C[\cdot : \sigma] : \text{nat}: \text{ExCv}(C[s]) \leq \text{ExCv}(C[t])$
  
  "in any context: $t$ converges at least as often as $s"$

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

**Refuting equivalences** requires one context acting as counter-example

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

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

**Proving equivalences** is **harder** due to the quantification over all contexts.
Expected convergence of $s$ with bound $k = \text{number prob-reductions}$

$$\text{ExCv}(s, k) = \sum_{(1, s) \downarrow_L (q, t) \in \text{Eval}(1, s), \ |L| \leq k} q$$

→ 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 $\forall k \geq 0 : \exists d : \text{ExCv}(s, k) \leq \text{ExCv}(t, k + d)$. Then $\text{ExCv}(s) \leq \text{ExCv}(t)$. 
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 $R[\cdot : \sigma] : \text{nat}$ there exists $d \geq 0$: $\text{ExCv}(R[s_i], k) \leq \text{ExCv}(R[t_i], k + d)$.

Let $C[\cdot_1, \sigma, \ldots, \cdot_N, \sigma] : \text{nat}$ be a multicontext with $N$ holes of type $\sigma$.

Then the inequation $\text{ExCv}(C[s_1, \ldots, s_N]) \leq \text{ExCv}(C[t_1, \ldots, t_N])$ holds.

- Instantiation for $N = 1$:
  
  If $\forall k \geq 0$, $R[\cdot : \sigma] : \text{nat}$, $\exists d \geq 0$: $\text{ExCv}(R[s], k) \leq \text{ExCv}(R[t], k + d)$, then $s \leq_c t$.

- Valuable proof tool to show contextual equivalences
A program transformation \( T \) is a binary relation of equally typed expressions. \( T \) is correct iff \( T \subseteq \sim_c \).

**Some Correct Program Transformations**

- **(fix)** \( \text{fix} \lambda x.s \to (\lambda x.s) (\text{fix} \lambda x.s) \)
- **(llet)** \( \text{let } x = (\text{let } y = s \text{ in } t) \text{ in } r \)
- **(lbeta)** \( ((\lambda x.s) t) \to \text{let } x = t \text{ in } s \)
- **(succ)** \( \text{succ } n \to n + 1 \)
- **(pred)** \( \text{pred } n \to \max(0, n - 1) \)
- **(if-then)** \( \text{if } 0 \text{ then } s \text{ else } t \to s \)
- **(if-else)** \( \text{if } n \text{ then } s \text{ else } t \to t \text{ if } n \neq 0 \)
- **(lflata)** \( A^1[(\text{let } x = s \text{ in } t)] \to \text{let } x = s \text{ in } A^1[t] \)
- **(llet)** \( \text{let } y = s, x = t \text{ in } r \)

- **(cp)** \( \text{let } x = v \text{ in } C[x] \)
- **(gc)** \( \text{let } x = s \text{ in } t \to t \text{ if } x \notin FV(t) \)
- **(⊕-id)** \( (s \oplus s) \to s \)
- **(⊕-comm)** \( (s \oplus t) \to (t \oplus s) \)
- **(⊕-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

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

Example:

- $(0 \oplus 1) + 2 \ast (0 \oplus 1)$
  “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

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

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

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

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

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

$$\left\{ \left( \frac{1}{4}, 0 \right), \left( \frac{1}{4}, 1 \right), \left( \frac{1}{8}, 2 \right), \left( \frac{1}{8}, 3 \right), \left( \frac{1}{16}, 4 \right), \left( \frac{1}{16}, 5 \right), \ldots \right\}$$
Contextual equivalence implies distribution-equivalence:

**Theorem**

Let $s, t : \sigma$ be two typed expressions with $s \sim_c t$.

Then for any context $C[\cdot_\sigma] : \text{nat}$, $C[s] \sim_d C[t]$.

Reverse direction:

**Conjecture**

If the distribution of closed expressions $s, t : \text{nat}$ 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!