Program Equivalence in Linear Contexts

Yu Zhang
Institute of Software, Chinese Academy of Sciences

Joint work with Yuxing Deng (BASICS, SJTU)

Beijing November 5, 2013

An Example

- Are the following two programs contextually equivalent?

$$\begin{array}{lcl} P_1 & \stackrel{\text{def}}{=} & \lambda x\,.\,(0 \sqcap 1) \\ P_2 & \stackrel{\text{def}}{=} & (\lambda x\,.\,0) \sqcap (\lambda x\,.\,1). \end{array}$$

$\sqcap$ is the internal choice (like in CSP).

An Example

- Are the following two programs contextually equivalent?

$$\begin{aligned} P_1 &\stackrel{\text{def}}{=} \lambda x\,.\,(0 \sqcap 1) \\ P_2 &\stackrel{\text{def}}{=} (\lambda x\,.\,0) \sqcap (\lambda x\,.\,1). \end{aligned}$$

$\sqcap$ is the internal choice (like in CSP).

- Answer: NO!
  - The following program can distinguish them:

$$\texttt{bind}\ f = [\_]\ \texttt{in}\ \texttt{bind}\ x = f(0)\ \texttt{in}\ \texttt{bind}\ y = f(0)\ \texttt{in}\ (x = y)$$

  - But it requires evaluating the target program twice.
  - What if the target program is only allowed to be used linearly (only once)?

Motivation

- We noticed this problem when using our CSLR logic to prove security of cryptographic constructions.
- CSLR logic [Zhang'09, NZ'10, NZ'13]:
  - A functional language with a type system that characterizes probabilistic polynomial-time computations (PPT class).
  - An equational proof system that helps to justify computational indistinguishability between programs.
- Semantic security:

$$\lambda\eta\,.\,\lambda m_0\,.\,\lambda m_1\,.\,\boldsymbol{Enc}(\eta, m_0, pk) \simeq_C \lambda\eta\,.\,\lambda m_0\,.\,\lambda m_1\,.\,\boldsymbol{Enc}(\eta, m_1, pk)$$

It is sufficient to prove that the two programs are equivalent in linear contexts [Goldreich'04].

Main Result

- Proof techniques for contextual equivalence:
  - Logical relations [Plotkin'80, Pitts'97, MS'92, GLN'02, ...]
  - Simulation relations [Abrasmsky'90, Bierman'00, Jeffrey'99, ...]
    Howe's approach [Howe'96]
  - None of these technique can help us to prove the equivalence in the example.

Main Result

- Proof techniques for contextual equivalence:
  - Logical relations [Plotkin’80, Pitts’97, MS’92, GLN’02, ...]
  - Simulation relations [Abrasmsky’90, Bierman’00, Jeffrey’99, ...]
    Howe’s approach [Howe’96]
  - None of these technique can help us to prove the equivalence in the example.
- Our result: linear contextual equivalence is trace equivalence!
  - Sound and complete.
  - Valid in both deterministic and non-deterministic languages.

The Non-deterministic Linear PCF

The Language

- Types:

$$\tau \,\&\, \tau' \mid \tau \otimes \tau' \mid \tau \to \tau' \mid \tau \multimap \tau' \mid \mathsf{T}\tau \mid \ldots$$

Non-linear function types are primitive and no exponential constructor.

- Expressions:

$\lambda x \,.\, e \mid e\, e'$	Abstractions and applications
$\langle e_1, e_2 \rangle \mid \mathtt{proj}_i(e)$	Products and projections
$e_1 \otimes e_2 \mid \mathtt{let}\ x \otimes y = e\ \mathtt{in}\ e'$	Tensor products and projections
$\mathtt{fix}_\tau$	Fix-point recursions
$\ldots$	$\ldots$
$\mathtt{val}(e)$	Trivial computation
$\mathtt{bind}\ x = e\ \mathtt{in}\ e'$	Sequential composition
$e \sqcap e'$	Non-deterministic choice

Typing Rules

$$\Gamma; \Delta \vdash e : \tau$$

Γ: non-linear resources, Δ: linear resources.

- Tensor products:

$$\frac{\Gamma; \Delta_i \vdash e_i : \tau_1 \ (i = 1, 2)}{\Gamma; \Delta_1, \Delta_2 \vdash e_1 \otimes e_2 : \tau_1 \otimes \tau_2} \qquad \frac{\Gamma; \Delta, x : \tau_1, y : \tau_2 \vdash e : \tau \quad \Gamma; \Delta' \vdash e' : \tau_1 \otimes \tau_2}{\Gamma; \Delta, \Delta' \vdash \mathtt{let}\ x \otimes y = e'\ \mathtt{in}\ e : \tau}$$

- Linear functions:

$$\frac{\Gamma; \Delta, x : \tau \vdash e : \tau'}{\Gamma; \Delta \vdash \lambda x \,.\, e : \tau \multimap \tau'} \qquad \frac{\Gamma; \Delta \vdash e : \tau' \multimap \tau \quad \Gamma; \Delta' \vdash e' : \tau'}{\Gamma; \Delta, \Delta' \vdash e\, e' : \tau'}$$

- Non-determinism:

$$\frac{\Gamma; \Delta \vdash e_1 : \mathsf{T}\tau_1 \quad \Gamma; \Delta', x : \tau_1 \vdash e_2 : \mathsf{T}\tau_2}{\Gamma; \Delta, \Delta' \vdash \mathtt{bind}\ x = e_1\ \mathtt{in}\ e_2 : \mathsf{T}\tau_2} \qquad \frac{\Gamma; \Delta \vdash e_i : \mathsf{T}\tau \ (i = 1, 2)}{\Gamma; \Delta \vdash e_1 \sqcap e_2 : \mathsf{T}\tau}$$

Operational Semantics

- Call-by-name semantics:
  - Reductions:

$$
\begin{array}{l}
(\lambda x.e)e' \leadsto e[e'/x] \\
\mathtt{let}\ x \otimes y = e_1 \otimes e_2\ \mathtt{in}\ e \leadsto e[e_1/x, e_2/y] \\
\mathtt{bind}\ x = \mathtt{val}(e')\ \mathtt{in}\ e \leadsto e[e'/x] \\
e_1 \sqcap e_2 \leadsto e_i\ (i = 1, 2), \\
\ldots \quad \ldots
\end{array}
$$

- Evaluation contexts:

$$\mathcal{E} ::= \mathcal{E}\, e \mid \mathtt{proj}_i(\mathcal{E}) \mid \mathtt{let}\ x \otimes y = \mathcal{E}\ \mathtt{in}\ e \mid \mathtt{bind}\ x = \mathcal{E}\ \mathtt{in}\ e \mid \mathtt{val}(\mathcal{E}) \mid \ldots$$

  - Linear resources can be computed (reduced) only once during evaluation.
  - Not evaluation contexts: $\langle \mathcal{E}, e \rangle,\ \langle e, \mathcal{E} \rangle, \mathcal{E} \sqcap e, e \sqcap \mathcal{E}, \ldots$

Linear Contextual Equivalence

- A linear context $\mathcal{C}_{x:\tau}$ is a program with a single linear variable x and no non-linear variables, i.e.,

$$\emptyset; x : \tau \vdash \mathcal{C}_{x:\tau} : \sigma$$

- Linear contextual equivalence (Morris-style):
  - e may converge (written as $e \Downarrow$) if there exists a value v such that $e \rightsquigarrow^* v \not\rightsquigarrow$;
  - *Linear contextual preorder*: $e_1 \sqsubseteq_\tau e_2$ if $\mathcal{C}[e_1/x] \Downarrow$ implies $\mathcal{C}[e_2/x] \Downarrow$ for all linear context $\mathcal{C}_{x:\tau}$.
  - *Linear contextual equivalence* $\simeq$: $e_1 \simeq_\tau e_2$ iff $e_1 \sqsubseteq_\tau e_2$ and $e_2 \sqsubseteq_\tau e_1$.

Trace Model

Program Transitions

A labeled transition system (based on [Gordon'95])

$$\frac{c \in \{\texttt{true}, \texttt{false}, 0, 1, 2, \ldots\}}{c \xrightarrow{c} \mathbf{\Omega}} \qquad \frac{\Gamma; \Delta \vdash \langle e_1, e_2 \rangle : \tau_1 \,\&\, \tau_2}{\langle e_1, e_2 \rangle \xrightarrow{\texttt{proj}_i} e_i}$$

$$\frac{\Gamma; \Delta \vdash \lambda x \,.\, e : \tau \quad \emptyset; \emptyset \vdash e' : \tau' \quad \tau \equiv \tau' \multimap \tau'' \text{ or } \tau' \to \tau''}{\lambda x \,.\, e \xrightarrow{@e'} e[e'/x]}$$

$$\frac{\Gamma; \Delta \vdash e_1 \otimes e_2 : \tau_1 \otimes \tau_2 \qquad \emptyset; x : \tau_1, y : \tau_2 \vdash e : \tau}{e_1 \otimes e_2 \xrightarrow{\otimes e} e[e_1/x, e_2/y]}$$

$$\frac{\Gamma; \Delta \vdash \texttt{val}(e) : \mathsf{T}\tau}{\texttt{val}(e) \xrightarrow{\mathsf{T}} e}$$

- Program transitions describes how programs can interact with contexts (leak information to contexts).

Example of Program Traces

$$
\begin{aligned}
P_1 &\equiv \mathtt{val}(\lambda x.\mathtt{val}(0) \sqcap \mathtt{val}(1)) \\
P_2 &\equiv \mathtt{val}(\lambda x.\mathtt{val}(0)) \sqcap \mathtt{val}(\lambda x.\mathtt{val}(1))
\end{aligned}
$$

Both programs have traces $\langle \mathsf{T}, @e, \mathsf{T}, 1\rangle, \langle \mathsf{T}, @e, \mathsf{T}, 0\rangle$:

$$
\begin{aligned}
P_1 &\xrightarrow{\mathsf{T}} \lambda x.\mathtt{val}(0) \sqcap \mathtt{val}(1) \\
&\xrightarrow{@e} (\mathtt{val}(0) \sqcap \mathtt{val}(1))[e/x] \\
&\equiv \mathtt{val}(0) \sqcap \mathtt{val}(1) \\
&\leadsto \mathtt{val}(1) \\
&\xrightarrow{\mathsf{T}} 1 \\
&\xrightarrow{1} \mathbf{\Omega},
\end{aligned}
\qquad
\begin{aligned}
P_2 &\leadsto \mathtt{val}(\lambda x.\mathtt{val}(1)) \\
&\xrightarrow{\mathsf{T}} \lambda x.\mathtt{val}(1) \\
&\xrightarrow{@e} \mathtt{val}(1)[e/x] \\
&\equiv \mathtt{val}(1) \\
&\xrightarrow{\mathsf{T}} 1 \\
&\xrightarrow{1} \mathbf{\Omega}.
\end{aligned}
$$

Context Transitions

- Linear context transitions describes how contexts can interact with programs in the hole (consume information that hole programs leak):

$$\begin{array}{rcl}
\mathcal{C}[\mathtt{proj}_i(x)/y] & \overset{\mathtt{proj}_i}{\multimap\!\!\longrightarrow} & \mathcal{C}_y \ (i = 1, 2) \\
\mathcal{C}[x\, e/y] & \overset{@e}{\multimap\!\!\longrightarrow} & \mathcal{C}_y \\
\mathcal{C}[\mathtt{let}\ z_1 \otimes z_2 = x\ \mathtt{in}\ e/y] & \overset{\otimes e}{\multimap\!\!\longrightarrow} & \mathcal{C}_y \\
\mathcal{C}[\mathtt{bind}\ z = x\ \mathtt{in}\ e/y] & \overset{\mathsf{T}}{\multimap\!\!\longrightarrow} & \mathcal{C}[(\lambda z.e)x'/y]
\end{array}$$

- A linear context transition often transforms the free variable into another one (of a different type).

Proving Linear Contextual Equivalence

Linear Context Reduction

- A reduction of $\mathcal{C}[e/x]$ ($\mathcal{C}_{x:\tau}$ be a linear context) is a linear context reduction if it is in one of the following forms:
  - $\mathcal{C}[e/x] \rightsquigarrow \mathcal{C}'[e/x]$, if $\mathcal{C} \rightsquigarrow \mathcal{C}'$;
  - $\mathcal{C}[e/x] \rightsquigarrow \mathcal{C}[e'/x]$, if $\mathcal{C}$ is an evaluation context, and $e \rightsquigarrow e'$;
  - $\mathcal{C}[e/x] \rightsquigarrow \mathcal{C}'[e'/y]$, if $\mathcal{C}$ is an evaluation context, $e \not\rightsquigarrow$, and $\mathcal{C} \circ\!\!\xrightarrow{\alpha} \mathcal{C}'$, $e \xrightarrow{\alpha} e'$ for some external action α.

Linear Context Reduction Lemma

Lemma. For every linear context $\mathcal{C}_{x:\tau}$ and LPCF program e, if $\mathcal{C}[e/x]$ is reducible, then $\mathcal{C}[e/x] \rightsquigarrow$ must be a linear context reduction.

- Proof by structural induction on the linear context.
- Not true for non-linear contexts: we do not necessarily have the second and the third form if the context contains multiple copies of the target program.
- The core lemma for proving precongruence of trace equivalence w.r.t. linear contextual equivalence.

Soundness of Trace Equivalence

- Trace preorder $\sqsubseteq^T$: $e_1 \sqsubseteq^T e_2$ if all traces of e_1 are traces of e_2.
- **Theorem.** Trace preorder $\sqsubseteq$ is a precongruence with respect to linear contexts, i.e., $e_1 \sqsubseteq e_2$ implies that $\mathcal{C}[e_1/x] \sqsubseteq \mathcal{C}[e_2/x]$.
  - Standard induction over traces (of $\mathcal{C}[e_i/x]$) works for deterministic languages, but not for non-determinism: trace preorder does not conform to induction in general.
  - Proof by inductively constructing a relation between traces of e and those of $\mathcal{C}[e/x]$.
  - This allows for proving precongruence by induction on traces of $\mathcal{C}[e/x]$.
  - The proof technique also works for deterministic langauges.

Soundness

- **Soundness theorem.** In NLPCF, $\simeq^T \subseteq \simeq^C$.
  - This allows us to prove the equivalence of the two programs in linear contexts:

$$
\begin{array}{rcl}
P_1 & \equiv & \mathtt{val}(\lambda x.\mathtt{val}(0) \sqcap \mathtt{val}(1)) \\
P_2 & \equiv & \mathtt{val}(\lambda x.\mathtt{val}(0)) \sqcap \mathtt{val}(\lambda x.\mathtt{val}(1))
\end{array}
$$

Both have traces $\langle \mathsf{T}, @e, \mathsf{T}, 1\rangle, \langle \mathsf{T}, @e, \mathsf{T}, 0\rangle$.

Completeness

- **Completeness theorem.** $\simeq^C \subseteq \simeq^T$ in NLPCF.
  - Induction over traces does not work for non-deterministic languages.
  - We construct *trace-sepcific contexts* to recognize given traces — the context will perform the exact sequence of interactions with target programs as specified by the trace.
  - We show that a program can take a trace s if and only if the corresponding s-specific context (filled with the program) may converge.
  - Proof also works in deterministic languages.

Conclusion

- Proving program equivalence in linear contexts:
  - Characterizing linear contextual equivalence by trace equivalence.
  - The proof is both sound and complete.
  - Proof techniques work in both deterministic and non-deteministic languages.
- Future work
  - Probabilistic languages (for application in cryptography).
  - Denotational models.