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
Are the following two programs contextually equivalent?

\[ P_1 \overset{\text{def}}{=} \lambda x . (0 \sqcap 1) \]
\[ P_2 \overset{\text{def}}{=} (\lambda x . 0) \sqcap (\lambda x . 1). \]

\( \sqcap \) is the internal choice (like in CSP).
Are the following two programs contextually equivalent?

\[ P_1 \overset{\text{def}}{=} \lambda x . (0 \sqcap 1) \]
\[ P_2 \overset{\text{def}}{=} (\lambda x . 0) \sqcap (\lambda x . 1). \]

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

Answer: NO!

- The following program can distinguish them:

\[
\begin{align*}
\text{bind } f & = [\_] \text{ in bind } x = f(0) \text{ in bind } y = f(0) \text{ in } (x = y)
\end{align*}
\]

- 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:

  \[
  \lambda \eta . \lambda m_0 . \lambda m_1 . Enc(\eta, m_0, pk) \sim_c \lambda \eta . \lambda m_0 . \lambda m_1 . Enc(\eta, m_1, pk)
  \]

  It is sufficient to prove that the two programs are equivalent in linear contexts [Goldreich’04].
Proof techniques for contextual equivalence:

- Logical relations [Plotkin’80, Pitts’97, MS’92, GLN’02, ...]
- Simulation relations [Abramsky’90, Bierman’00, Jeffrey’99, ...]

Howe’s approach [Howe’96]

None of these techniques 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 [Abramsky’90, Bierman’00, Jeffrey’99, ...]
    Howe’s approach [Howe’96]
  - None of these techniques 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:

\[ \tau \land \tau' \mid \tau \otimes \tau' \mid \tau \rightarrow \tau' \mid \tau \rightsquigarrow \tau' \mid T\tau \mid \ldots \]

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

- Expressions:

\begin{align*}
& \lambda x . e \mid e e' \\
& \langle e_1, e_2 \rangle \mid \text{proj}_i(e) \\
& e_1 \otimes e_2 \mid \text{let } x \otimes y = e \text{ in } e' \\
& \text{fix}_\tau \\
& \ldots \\
& \text{val}(e) \\
& \text{bind } x = e \text{ in } e' \\
& e \sqcap e' \\
\end{align*}

Abstractions and applications
Products and projections
Tensor products and projections
Fix-point recursions
Trivial computation
Sequential composition
Non-deterministic choice
Typing Rules

\[ \Gamma; \Delta \vdash e : \tau \]

\(\Gamma\): non-linear resources, \(\Delta\): linear resources.

- **Tensor products:**

  \[ \Gamma; \Delta_i \vdash e_i : \tau_1 \ (i = 1, 2) \]
  \[ \Gamma; \Delta_1, \Delta_2 \vdash e_1 \otimes e_2 : \tau_1 \otimes \tau_2 \]

  \[ \Gamma; \Delta, \Delta' \vdash \text{let } x \otimes y = e' \text{ in } e : \tau \]

- **Linear functions:**

  \[ \Gamma; \Delta, x : \tau \vdash e : \tau' \]
  \[ \Gamma; \Delta \vdash \lambda x. e : \tau \rightarrow \tau' \]

  \[ \Gamma; \Delta \vdash e : \tau' \rightarrow \tau \]
  \[ \Gamma; \Delta' \vdash e' : \tau' \]

  \[ \Gamma; \Delta, \Delta' \vdash ee' : \tau' \]

- **Non-determinism:**

  \[ \Gamma; \Delta \vdash e_1 : T\tau_1 \]
  \[ \Gamma; \Delta', x : \tau_1 \vdash e_2 : T\tau_2 \]

  \[ \Gamma; \Delta, \Delta' \vdash \text{bind } x = e_1 \text{ in } e_2 : T\tau_2 \]

  \[ \Gamma; \Delta \vdash e_i : T\tau \ (i = 1, 2) \]
  \[ \Gamma; \Delta \vdash e_1 \sqcap e_2 : T\tau \]
Operational Semantics

- Call-by-name semantics:
  - Reductions:
    
    $$(\lambda x.e)e' \leadsto e[e'/x]$$  
    
    let $x \otimes y = e_1 \otimes e_2$ in $e$ $\leadsto e[e_1/x, e_2/y]$
    
    bind $x = \text{val}(e')$ in $e$ $\leadsto e[e'/x]$
    
    $e_1 \sqcap e_2 \leadsto e_i \ (i = 1, 2),$
    
    ...  

- Evaluation contexts:
  
  $\mathcal{E} ::= \mathcal{E} e \mid \text{proj}_i(\mathcal{E}) \mid \text{let } x \otimes y = \mathcal{E} \text{ in } e \mid \text{bind } x = \mathcal{E} \text{ in } e \mid \text{val}(\mathcal{E}) \mid \ldots$
  
  - 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 $C_{x:\tau}$ is a program with a single linear variable $x$ and no non-linear variables, i.e.,

$$\emptyset; x : \tau \vdash C_{x:\tau} : \sigma$$

- Linear contextual equivalence (Morris-style):
  - $e$ may converge (written as $e \downarrow$) if there exists a value $v$ such that $e \Rightarrow^* v \not\Rightarrow$.
  - Linear contextual preorder: $e_1 \sqsubseteq_\tau e_2$ if $C[e_1/x] \downarrow$ implies $C[e_2/x] \downarrow$ for all linear context $C_{x:\tau}$.
  - 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])

\[ c \in \{ \text{true, false, 0, 1, 2, \ldots} \} \]
\[ c \xrightarrow{c} \Omega \]
\[ \Gamma; \Delta \vdash \lambda x . e : \tau \quad \emptyset; \emptyset \vdash e' : \tau' \]
\[ \tau \equiv \tau' \circ \tau'' \text{ or } \tau' \rightarrow \tau'' \]
\[ \lambda x . e \xrightarrow{\oplus e'} e'[e'/x] \]
\[ \Gamma; \Delta \vdash e_1 \otimes e_2 : \tau_1 \otimes \tau_2 \quad \emptyset; x : \tau_1, y : \tau_2 \vdash e : \tau \]
\[ e_1 \otimes e_2 \xrightarrow{\otimes e} e[e_1/x, e_2/y] \]
\[ \Gamma; \Delta \vdash \text{val}(e) : T\tau \]
\[ \text{val}(e) \xrightarrow{T} e \]

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

\[
P_1 \equiv \text{val}(\lambda x.\text{val}(0) \sqcap \text{val}(1))
\]

\[
P_2 \equiv \text{val}(\lambda x.\text{val}(0)) \sqcap \text{val}(\lambda x.\text{val}(1))
\]

Both programs have traces \(\langle T, @e, T, 1 \rangle, \langle T, @e, T, 0 \rangle\):

\[
P_1 \xrightarrow{T} \lambda x.\text{val}(0) \sqcap \text{val}(1)
\]

\[
P_1 \xrightarrow{@e} (\text{val}(0) \sqcap \text{val}(1))[e/x]
\equiv \text{val}(0) \sqcap \text{val}(1)
\]

\[
P_1 \xrightarrow{\sim} \text{val}(1)
\]

\[
P_1 \xrightarrow{T} 1
\]

\[
P_1 \xrightarrow{1} \Omega,
\]

\[
P_2 \xrightarrow{T} \lambda x.\text{val}(1)
\]

\[
P_2 \xrightarrow{@e} \text{val}(1)[e/x]
\equiv \text{val}(1)
\]

\[
P_2 \xrightarrow{T} 1
\]

\[
P_2 \xrightarrow{1} \Omega.
\]
Linear context transitions describes how contexts can interact with programs in the hole (consume information that hole programs leak):

\[
\begin{align*}
C[\text{proj}_i(x)/y] & \xrightarrow{\text{proj}_i} C_y (i = 1, 2) \\
C[x e/y] & \xrightarrow{\@e} C_y \\
C[\text{let } z_1 \otimes z_2 = x \text{ in } e/y] & \xrightarrow{\otimes e} C_y \\
C[\text{bind } z = x \text{ in } e/y] & \xrightarrow{T} C[(\lambda z.e)x'/y]
\end{align*}
\]

A linear context transition often transforms the free variable into another one (of a different type).
Proving Linear Contextual Equivalence
A reduction of $C[e/x]$ ($C_{x:\tau}$ be a linear context) is a linear context reduction if it is in one of the following forms:

- $C[e/x] \rightarrow C'[e/x]$, if $C \rightarrow C'$;
- $C[e/x] \rightarrow C[e'/x]$, if $C$ is an evaluation context, and $e \rightarrow e'$;
- $C[e/x] \rightarrow C'[e'/y]$, if $C$ is an evaluation context, $e \not\rightarrow$, and $C \circ \alpha \rightarrow C'$, $e \xrightarrow{\alpha} e'$ for some external action $\alpha$. 
Lemma. For every linear context $C_{x:	au}$ and LPCF program $e$, if $C[e/x]$ is reducible, then $C[e/x] \rightsquigarrow$ 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 $C[e_1/x] \sqsubseteq C[e_2/x]$.
  
  - Standard induction over traces (of $C[e_i/x]$) 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 $C[e/x]$.
  
  - This allows for proving precongruence by induction on traces of $C[e/x]$.
  
  - The proof technique also works for deterministic languages.
Soundness

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

\[
\begin{align*}
  P_1 & \equiv \text{val}(\lambda x.\text{val}(0) \sqcap \text{val}(1)) \\
  P_2 & \equiv \text{val}(\lambda x.\text{val}(0)) \sqcap \text{val}(\lambda x.\text{val}(1))
\end{align*}
\]

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

Completeness

- **Completeness theorem.** \( \sim^C \subseteq \sim^T \) in NLPCF.
  
  - Induction over traces does not work for non-deterministic languages.
  - We construct *trace-specific 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-deterministic languages.

- Future work
  - Probabilistic languages (for application in cryptography).
  - Denotational models.