Verified Compilation of Weak Call-by-Value \( \lambda \)-Calculus into Combinators and Closures

Bachelor’s Talk

Fabian Kunze

Advisor: Prof. Dr. Gert Smolka

Programming Systems Lab

11.12.2015
L: Weak Call-by-Value $\lambda$-Calculus

$s, t, u ::= x \mid \lambda x.s \mid st \quad (x \in \mathbb{N})$

$x_u^x := u \quad (st)_u^x := s_u^x t_u^x$

$y_u^x := y \quad (\lambda y.s)_u^x := \lambda y.(s_u^x)$

- Turing complete
- Data can be represented as procedure (closed $\lambda$-abstraction) using Church encoding.
SK Combinatory Logic

\[ X, Y, Z ::= x \mid S \mid K \mid XY \quad (x \in \mathbb{N}) \]

\[
\begin{align*}
KXY & \Rightarrow X \\
SXYZ & \Rightarrow (XZ)(YZ) \\
XY & \Rightarrow X'Y \\
Y & \Rightarrow Y'
\end{align*}
\]

- also called SKI, but combinator I can be defined:

\[
I ::= SKK
\]

\[
IX = SKKX \Rightarrow KX(KX) \Rightarrow X
\]
SK can "simulate" substitution:

Example

\[ \lambda x. (xy) \sim SI(Ky): \]

\[ (\lambda x. (xy))z \triangleright zy \]

\[ SI(Ky)z \triangleright (Iz)(Kyz) \triangleright^* zy \]

- **S**: 'push' argument down in application
- **K**: discard 'pushed' argument
- **I**: take 'pushed' argument

λ-calculus can be embedded in SK (but altered SK-equivalence).

We will embed L into a call-by-value version of SK!
Content

1 Introduction
   - L: Weak Call-by-Value $\lambda$-Calculus
   - SK Combinatory Logic

2 SKv: Call-by-Value Combinatory Logic
   - Pseudo-Abstraction

3 Compiling L into SKv
   - Soundness
   - Left-Invertibility of Compilation
   - Completeness on normalizing terms
   - Completeness on arbitrary terms

4 LC: L with closures

5 Related Work
SKv: Call-by-Value Combinatory Logic

\[
\begin{align*}
X, Y, Z & ::= x \mid K \mid S \mid XY \quad (x \in \mathbb{N}) \\
Val \ni X, Y & ::= x \mid K \mid KX \mid S \mid SX \mid SXY \quad (x \in \mathbb{N})
\end{align*}
\]

\[
\begin{align*}
X \sim_{SK} X' & \quad Y \sim_{SK} Y' & \quad X, Y \in \text{Val} \\
XY \sim_{SK} X'Y & \quad XY \sim_{SK} XY' & \quad KXY \sim_{SK} X \\
SXYZ \sim_{SK} XZ(YZ)
\end{align*}
\]

- If \( X_1 \sim_{sk}^{k_1} Y_1 \) and \( X_2 \sim_{sk}^{k_2} Y_2 \), then \( X_1X_2 \sim_{sk}^{k_1+k_2} Y_1Y_2 \).
- Values are irreducible, and closed irreducible terms are values.
- \( I ::= SKK \) yield \( IX \sim_{SK}^2 X \)
Uniform Confluence

Uniform Confluence

If \( y_1 \leftarrow x \rightarrow y_2 \), then either
\( y_1 = y_2 \) or \( \exists z, y_1 \rightarrow z \leftarrow y_2 \).

\[ \Rightarrow: \text{Take } k_1 = k_2 = 1. \]

\[ \Leftarrow: \text{Induction on } k_1 \text{ and } k_2. \]

Call-by-value systems (like L and SKv) have the uniform diamond:
Redexes are not nested, so the two reductions either contract the same redex (\( y_1 = y_2 \)).
Or they contract disjoint redexes; contracting both joins \( y_1 \) and \( y_2 \).
Substitution in SKv

\[
\begin{align*}
x_Z^x &:= u & K_Z^x &:= K & (XY)_Z^x &:= X_Z^x Y_Z^x \\
y_Z^y &:= y & S_Z^x &:= S
\end{align*}
\]

Substitutivity:

- If \( Y \) is closed and \( x \neq z \), then \( z \in \text{FV}(X) \) iff \( z \in \text{FV}(X_Y^x) \).

No similar lemma for values, e.g. \( xK \not\in \text{Val} \), but \( (xK)_K^x = KK \in \text{Val} \).

\( X \) is a *maximal value* iff \( X \in \text{Val} \), but \( XY \not\in \text{Val} \).

So maximal values are values of form \( x, KX \) and \( SXY \).

- If \( Y \) is a maximal value, then \( X \in \text{Val} \) iff \( X_Y^x \in \text{Val} \).
Pseudo-Abstraction

\[
[x].x := I \\
[x].X := KX \\
[x].(XY) := S([x].X)([x].Y)
\]

if \(x \not\in \text{FV}(X) \land X \in \text{Val}\)

otherwise

Similarities to L-abstractions:

- \([x].X\) is maximal value.
- \(Y\) value \(\implies ([x].X)Y \succeq^+_{SK} X^x_Y\)
- \(\text{FV}([x].X) = \text{FV}(X) \setminus \{x\}\)

Commutes with Substitution:

- \(Y\) maximal value \(\land z \not\in \text{FV}(Y) \land z \neq x \implies ([z].X)^x_Y = [z].(X^x_Y)\)

Proof by Induction on \(X\).

Crucial: \(z \not\in \text{FV}(X) \land X \in \text{Val} \iff z \not\in \text{FV}(X^x_Y) \land X^x_Y \in \text{Val}\)
Compiling L into SKv

\[ C \times := x \]
\[ C (st) := (C s)(C t) \]
\[ C (\lambda x. s) := [x].(C s) \]

For readability: \( X := C X \)

- If \( s \) is closed, then \( s \) is abstraction iff \( C s \) is a (maximal) value.
  - Thus a closed \( s \) is \( L \)-redex iff \( s \) is \( SKv \)-redex
- If \( t \) is a procedure, then \( s^\times_{\overline{t}} = \overline{s^\times_t} \)

**Soundness**

If \( s \) is closed and \( s \overset{L}{\Rightarrow} t \), then \( C s \overset{+}{\Rightarrow}_{SK} C t \).

Implies (for closed \( s \)):

\[ s \overset{*}{\Rightarrow}_{L} t \Rightarrow C s \overset{*}{\Rightarrow}_{SK} C t \quad \text{and} \quad s \downarrow t \Rightarrow C s \downarrow C t \]
Left-Invertibility of Compilation

\[ [x]^{-1}(SKK) := x \]
\[ [x]^{-1}(SXY) := ([x]^{-1}.X)([x]^{-1}.Y) \]
\[ [x]^{-1}(KX) := X \]

\[ [x]^{-1}([x].X) = X \]

\[ C^{-1} x := x \]
\[ C^{-1} X := \lambda x.(C^{-1} ([x]^{-1}.X)) \quad \text{if } X \in \text{Val} \]
\[ C^{-1} (XY) := (C^{-1} X)(C^{-1} Y) \]

\[ C^{-1} (C \ s) = s \]

So \( C \) is injective (modulo \( \alpha \)-conversion).
Completeness

\[ C \ s \succ^*_S \ C \ t \implies s \succ^*_L t \] does not hold:

Example

For a (reasonable) procedure \( u \):
\[
(\lambda x. xx)u = \text{SII}u \succ^*_S (Iu) (Iu) = ((\lambda x. x)u) ((\lambda x. x)u),
\]
but \( (\lambda x. xx)u \not\succ^*_L ((\lambda x. x)u) ((\lambda x. x)u) \)

SKv-reductions can be extended to correspond to L-reductions:

\[
\text{closed} \land C \ s \succ^*_S X \implies \exists t, X \succ^*_S C \ t \land s \succ_L t
\]
Completeness on normalizing terms

SKv-reductions can be extended corresponding to L-reductions:

\[
s \text{ closed} \land C \ s \succ_{SK} X \implies \exists t, X \succ^*_{SK} C t \land s \succ_L t
\]

Proof

Induction on \(s \succ_{SK} X\):

- \(s\) is SKv-redex
  \[\implies s = (\lambda x.s')u,\text{ where }u\text{ procedure}\]
  - successor of \(s\) unique: \(X\)
  - \(s = (\lambda x.s')u \succ^+_{SK} (s')^X_u\)
  \[\implies t := s'^X_u\text{ has claimed properties}\]

- \(s\) not SKv-redex
  \[\implies s = s_1 s_2,\text{ redex contained in }s_1\text{ or }s_2\]
  \[\implies \text{claim holds by inductive hypothesis}\]
Completeness on normalizing terms (2)

\[ s \text{ closed} \land C \; s \succ_{SK} X \implies \exists t, X \succ^*_{SK} C \; t \land s \succ_L t \]

Generalizes to reduction chains:

\[ s \text{ closed} \land C \; s \succ^*_{SK} X \implies \exists t, X \succ^*_{SK} C \; t \land s \succ^*_{L} t \]

**Proof**

Induction on length of \( s \succ^K_{SK} X \):

- \( k = 0 \): trivial
- \( k = 1 + k' \): extend, uniform confluence and inductive hypothesis
Completeness on normalizing terms (3)

\[ s \text{ closed} \land C s \succ^{*}_{SK} X \implies \exists t, X \succ^{*}_{SK} C t \land s \succ^{*}_{L} t \]

For normalizing, closed \( s \):

\[ C s \Downarrow X \implies s \Downarrow C^{-1} X \]

Combined with soundness:

\[ C s \Downarrow C t \iff s \Downarrow t \]

This is satisfying for a term having a normal form, but we can do better!
Completeness on arbitrary terms

We want: $\mathcal{C} s \triangleright_{\text{SK}}^* \mathcal{C} t \Rightarrow s \equiv_L t$.

We study the $\mathcal{C}$-image of (closed) $\beta$-redexes, depending on the body:

Assume an abstraction $s$ and $(\lambda x. s)t \triangleright_{\text{SK}}^* u$. Then $(\lambda x. s)t \equiv_L u$

Proof

We have $(\lambda x. s)t \Downarrow s^x_t$.
We use completeness on normalizing terms and confluence of SKv.

For other bodies:

Assume a non-abstraction $s$ and $(\lambda x. s)t \triangleright_{\text{SK}} Y$.
Then there is a closed $s'$ with $s' = Y$ and $(\lambda x. s)t \equiv_L s'$

Proof

By exhausting case distinction on $s$ (variable and closed or non-closed application).
Completeness on arbitrary terms (2)

\[ s \text{ closed } \land \exists \mathcal{C} \ s \overset{k}{\preceq}_{SK} \mathcal{C} \ t \implies s \equiv_L t \]

Idea: Decompose \( X_1 X_2 \overset{k}{\preceq}_{SK} Y_1 Y_2 \):

\[
X_i \overset{k_i}{\preceq}_{SK} Y_i \lor X_i \overset{k}{\preceq}_{SK} Z_i \land \left( Z_1 Z_2 \overset{k'}{\preceq}_{SK} Y \land Y \overset{k'}{\preceq}_{SK} Y_1 Y_2 \right)_{\text{redex}}
\]

Proof by Lexicographic induction on \((k, s)\)

\( s = \lambda x.s' \): \( s = t \Rightarrow s = t \) by injectivity.

\( t = \lambda x.t' \): completeness on normalizing terms.

Decompose \( s = s_1 s_2 \overset{k}{\preceq}_{SK} t_1 t_2 = t \):

- \( s_i \overset{k_i}{\preceq}_{SK} t_i \) with \( k_i \leq k \): inductive hypothesis \( \Rightarrow s_i \equiv_L u_i \)

- \( s_i \overset{k_i}{\preceq}_{SK} u_i; u_1 u_2 \overset{k}{\preceq}_{SK} Y; Y \overset{k'}{\preceq}_{SK} t \); \( k_1 + k_2 + 1 + k' = k \); \( u_i \) procedures.

By inductive hypothesis: \( s_i \equiv_L u_i \).

- Body of \( u_1 \) abstraction: \( u_1 u_2 \equiv_L t \). So \( s_1 s_2 \equiv_L u_1 u_2 \equiv_L t \)

- Body of \( u_1 \) non-abstraction: \( \exists s' \) such that \( Y = s' \) and \( u_1 u_2 \equiv_L s' \).

So \( s_1 s_2 \equiv_L u_1 u_2 \equiv_L s' \equiv_L t \).
Summary

\( C \) compiles \( L \) into \( SKv \) and is fully compatible with term equivalence, evaluation and normality on closed terms \( s, t \):

\[
\begin{align*}
s \equiv_L t & \iff C \ s \equiv_{SK} C \ t \\
s \downarrow t & \iff C \ s \downarrow C \ t \\
s \text{normal} & \iff C \ s \text{ normal}
\end{align*}
\]

So the whole semantic structure of \( L \) can be embedded in \( SKv \) and also be pulled back to \( L \) using \( C^{-1} \).
**LC: L with closures**

\[ p, q, r ::= x \mid s[\sigma] \mid p \cdot q \quad (x \in \mathbb{N}, s \in L, \sigma : \mathbb{N} \rightarrow \text{LC}) \]

Intuition: Carry out substitution as deep as needed

\[
\begin{align*}
    x[\sigma] & \succcurlyeq_{\text{LC}} \sigma(x) \\
    (\lambda x.s)[\sigma] \cdot (\lambda y.t)[\tau] & \succcurlyeq_{\text{LC}} s[x \mapsto (\lambda y.t)[\tau], \sigma] \sigma \\
    st[\sigma] & \succcurlyeq_{\text{LC}} s[\sigma] \cdot t[\sigma] \\
    p & \succcurlyeq_{\text{LC}} p' \\
    p \cdot q & \succcurlyeq_{\text{LC}} p' \cdot q \\
    q & \succcurlyeq_{\text{LC}} q'
\end{align*}
\]

- **call-by-value** ⇒ uniformly confluent
- **admissible terms**: derivable from closed L-terms in empty context.
- **⌈·⌉ : L\text{C} \rightarrow L** substitutes using the environments
- **Simulation Lemma**: \( \downarrow t \iff \exists q, p \downarrow q \land t = q \).
- **We also have a complete interpreter for LC.**
Completeness on arbitrary terms

J. Roger Hindley and Jonathan P. Seldin. 

Yannick Forster
*A Formal and Constructive Theory of Computation*  
Bachelor thesis, Saarland University, 2014

Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy.
*Explicit substitutions*  

Jean-Jacques Lévy and Luc Maranget.  
*Explicit substitutions and programming languages*  
19th FSTTCS, p. 181–200, 1999