Simulating call by value in Combinatory Logic

Paolo Parisen Toldin

University of Bologna

10 June 2010
Motivations:

- What is the dynamic significance of the encoding of $\lambda$-calculus into Combinatory Logic?
  - Which notion of reduction in $\Lambda$ can be simulated in $\text{CL}$?
- How much does it cost to do a step of reduction in $\lambda$-calculus with cbv?
  - We can assume the cost of a reduction step to be unitary. [Moran04, DallagoMartini09]
  - Let’s translate in $\text{CL}$ and simulate it. We will reduce to $\lambda$ calculus weak.
- What are the relations between the complexity of $\lambda$ steps and a $\text{CL}$ step?
- We shall consider the $\lambda$-calculus weak.
Related works in bibliography:

- *Call-by-Value Combinatory Logic and the Lambda-Value Calculus* - J.Gateley & B.F.Duba
- *A New Implementation Technique for Applicative Languages* - D.A.Turner
- *On Constructor Rewrite Systems and Lambda Calculus* - U.Dal lago & S.Martini
The Curry translation:

\[
\begin{align*}
[x]_\eta &= x \\
[MN]_\eta &= [M]_\eta[N]_\eta \\
[\lambda x.M]_\eta &= [x]_\mu[M]_\eta
\end{align*}
\]

where

\[
\begin{align*}
[x]_\mu.M &= KM & x \notin FV(M) \\
[x]_\mu.x &= I \\
[x]_\mu.C &= KC \\
[x]_\mu.MN &= S([x]_\mu.M)([x]_\mu.N)
\end{align*}
\]

with \(C\) combinator
Bad Properties

- It does not map normal forms to normal forms.

**Example**

\[ \lambda x. \Delta \Delta \text{ is NF in the weak } \lambda\text{-calculus.} \]

\[ [\lambda x. \Delta \Delta]_\eta = K((SII)(SII)) \text{ is not NF in CL} \]

- In general, strong reduction cannot be simulated.

**Example**

\[ \lambda x. (\lambda y. y)x \to \lambda x. x \]

\[ [\lambda x. (\lambda y. y)x] = S(KI)I \]

- Weak reduction cannot be simulated exactly: \( \lambda \) weak is not confluent while \( CL \) is confluent.
Another abstraction algorithm

Do we need to choose another abstraction algorithm? **NO**
Consider the follow abstraction algorithm:

\[
\begin{align*}
[x]_\nu.y &= Ky & x \neq y \\
[x]_\nu.x &= I \\
[x]_\nu.C &= KC \\
[x]_\nu.MN &= S([x]_\nu.M)([x]_\nu.N)
\end{align*}
\]

with \( C \) combinator

This algoritm differs from the previus for the first rule.
It maps NF to NF, but it cannot be used for simulating call by value: it does not preserve the substitution.
From Combinatory Logic to $\lambda$ calculus, instead of using the standard translation

\[
\begin{align*}
[x]_\lambda &= x \\
[K]_\lambda &= \lambda xy.x \\
[XY]_\lambda &= [X]_\lambda [Y]_\lambda \\
[I]_\lambda &= \lambda x.x \\
[S]_\lambda &= \lambda xyz.xz(yz)
\end{align*}
\]

we’ll use the followed one:

\[
\begin{align*}
[x]_\lambda &= x \\
[KM]_\lambda &= \lambda y.[M]_\lambda \\
[XY]_\lambda &= [X]_\lambda [Y]_\lambda \\
[I]_\lambda &= \lambda x.x \\
[SMN]_\lambda &= \lambda z.[M]_\lambda z([N]_\lambda z)
\end{align*}
\]

we’ll see why...
Call-by-Value in CL

Values:

\[ V_2 = \{S\} \]
\[ V_1 = \{K\} \cup \{MN | M \in V_2 \land N \in CL\} \]
\[ V_0 = \{I\} \cup \{MN | M \in V_1 \land N \in CL\} \]
\[ V = \{M | M \in V_0 \lor M \in V_1 \lor M \in V_2 \lor M = x\} \]

\[
\frac{M \in V}{IM \triangleright_{wCBV} M}
\]
\[
\frac{N \in V}{KMN \triangleright_{wCBV} M}
\]
\[
\frac{P \in V}{SMNP \triangleright_{wCBV} (MP)(NP)}
\]
\[
\frac{M \triangleright_{wCBV} N}{ML \triangleright_{wCBV} NL}
\]
\[
\frac{M \triangleright_{wCBV} N}{LM \triangleright_{wCBV} LN}
\]
Let’s reformulate the rules in the following way. Let’s consider $S$, $K$, $I$ as functions, respectively, 2-ary, 1-ary, 0-ary. Here, simply, the set of values is defined as:

$V = \{M | M = I \lor M = K(P) \lor M = S(P, Q) \lor M = x, \text{ for some } P, Q\}$

\[
\begin{align*}
M \in V & \quad \Rightarrow \quad IM \triangleright_{wCBV} M \\
N \in V & \quad \Rightarrow \quad K(M)N \triangleright_{wCBV} M \\
P \in V & \quad \Rightarrow \quad S(M, N)P \triangleright_{wCBV} (MP)(NP) \\
M \triangleright_{wCBV} N & \quad \Rightarrow \quad ML \triangleright_{wCBV} NL
\end{align*}
\]

The $[.]_\mu$ abstraction produces always a term in $V$!
Results:

Theorem (lambda to combinatory)

\[ M \rightarrow_{\lambda w CBV} M' \implies [M]_{\eta} \triangleright_{\eta \Sigma 2}^{2 \times |M| - 1} [M']_{\eta} \]

Theorem (combinatory to lambda)

\[ M \triangleright_{\lambda CBV} M' \implies [M]_{\lambda} \rightarrow_{\lambda w}^* [M']_{\lambda} \]

Theorem (combinatory to lambda)

\[ [\cdot]_{\lambda} \text{ maps NFs to NFs.} \]
The reduction steps in CL may duplicate sub-terms.

We can implement it by using term graph rewriting as explained in [Turner79] by sharing these subgraphs.
General Overview

\[ \lambda C \quad M \quad M' \]

\[ CL \quad [M]_\eta \quad [M']_\eta \]

graphs
What about the call by name?

We have several ideas, still a working in progress. The main problem lies in the fact that doing only leftmost step is not enough.

\[
(\lambda z.\lambda x.xz)a \rightarrow \lambda x.xa
\]

\[
(S \, (K\,(SI))) \, ((S\,(KK))\,I))\]

\[
\rightarrow [K\,(SI)\,a][((S\,(KK))\,I)\,a] \ldots
\]
With Marco Gaboardi, I’m developing a general framework to deal with languages based on Combinatory Logic.

CoLoBo - Combinatory Logic in Bologna (http://colobo.sourceforge.net/)

The idea is to develop a tool to perform evaluation of quantitative properties (e.g. number of steps). It also works as a generic interpreter.
Questions?