

An Invariant Cost Model for the Lambda Calculus

Joint Work with Simone Martini

Ugo Dal Lago

Laboratoire d'Informatique de Paris-Nord
Université Paris XIII

CIE, July 3rd 2006

Outline

Lambda Calculus as a Reasonable Machine
Previous Work

The Difference Cost Model

Embedding Turing Machines

Evaluating with Turing Machines

Closed Values as a Partial Combinatory Algebra

Conclusions

Lambda Calculus as a way to Compute Functions

- ▶ We all know Lambda Calculus is Turing-Complete: it can compute every partial recursive function.
- ▶ From the point of view of **computability**, the Church-Turing thesis holds.
- ▶ And **complexity**? What about the *invariance* thesis?

“Reasonable machines can simulate each other within a polynomially bounded overhead in time and a constant-factor overhead in space”.

- ▶ Unfortunately, there is not a clear answer.
- ▶ What is a unit-cost computation step in the lambda-calculus?

What is the Cost of Normalization?

- ▶ We need an invariant cost model for the lambda calculus.
- ▶ Taking the number of beta-reductions is just too naive. Consider terms like $\underline{n} \underline{2}$ where \underline{m} is the Church Numeral for m (for every natural number m).
- ▶ Different reduction strategies yield different costs.
- ▶ We can design a cost model from an abstract machine, but this complicates things.
 - ▶ We lose the possibility to reason quantitatively directly in the calculus.
- ▶ The cost model we are looking for should be reasonably *parsimonious*, although this is not crucial.

Why?

- ▶ In many situations, evaluating the cost of reducing lambda terms is necessary.
- ▶ In particular, we are interested in the implicit computational complexity of higher-order systems.
 - ▶ In [DalLagoHofmann05], lambda terms are realizers in a quantitative semantical framework.
 - ▶ Invariance is really essential: we want to precisely characterize a complexity class.
- ▶ We do not want to evaluate the efficiency of abstract or concrete implementations of functional programming languages.
 - ▶ *Optimality* is not a crucial property.

Previous Work

- ▶ Previous work mainly comes from the literature on optimal reduction of lambda calculus.
- ▶ Frandsen and Sturivant proposed a cost model based on parallel reduction and redex-families [FrandsenSturivant91].
 - ▶ It is not known whether this cost model is invariant or not.
 - ▶ Mairson and Lawall's [MairsonLawall96] give negative evidences.
- ▶ Mairson and Lawall proposed another cost model based on Lévy's labels.
 - ▶ As described in [MairsonLawall97], it only works with lambda-terms which can be reduced with Lamping's abstract algorithm.

The Calculus

- ▶ Terms are defined as follows:

$$M ::= x \mid \lambda x.M \mid MM$$

Λ denotes the set of all lambda terms.

- ▶ Values are defined as follows:

$$V ::= x \mid \lambda x.M$$

Ξ denotes the set of all closed values.

- ▶ Call-by-value reduction is denoted by \rightarrow and is obtained by closing the rule

$$(\lambda x.M)V \rightarrow M\{V/x\}$$

under all *applicative* contexts. Here M ranges over terms, while V ranges over values.

Properties of the Calculus

Reduction is *almost* deterministic:

Lemma (Diamond Property)

If $M \rightarrow N$ and $M \rightarrow L$ then either $N \equiv L$ or there is P such that $N \rightarrow P$ and $L \rightarrow P$.

Corollary (Strategy Equivalence)

M has a normal form iff M is strongly normalizing.

Theorem

For every term M , there are at most one normal form N and one integer n such that $M \rightarrow^n N$.

Properties of the Calculus

Reduction is *almost* deterministic:

Lemma (Diamond Property)

If $M \rightarrow N$ and $M \rightarrow L$ then either $N \equiv L$ or there is P such that $N \rightarrow P$ and $L \rightarrow P$.

Corollary (Strategy Equivalence)

M has a normal form iff M is strongly normalizing.

Theorem

For every term M , there are at most one normal form N and one integer n such that $M \rightarrow^n N$.

Properties of the Calculus

Reduction is *almost* deterministic:

Lemma (Diamond Property)

If $M \rightarrow N$ and $M \rightarrow L$ then either $N \equiv L$ or there is P such that $N \rightarrow P$ and $L \rightarrow P$.

Corollary (Strategy Equivalence)

M has a normal form iff M is strongly normalizing.

Theorem

For every term M , there are at most one normal form N and one integer n such that $M \rightarrow^n N$.

The Difference Cost Model

- ▶ Concatenation of $\alpha, \beta \in \mathbb{N}^*$ is simply denoted as $\alpha\beta$.
- ▶ \rightarrow will denote a subset of $\Lambda \times \mathbb{N}^* \times \Lambda$. In the following, we will write $M \xrightarrow{\alpha} N$ standing for $(M, \alpha, N) \in \rightarrow$. The definition of \rightarrow (in SOS-style) follows:

$$\frac{M \rightarrow N \quad n = \max\{1, |N| - |M|\}}{M \xrightarrow{(n)} N}$$
$$\frac{}{M \xrightarrow{\varepsilon} M} \quad \frac{M \xrightarrow{\alpha} N \quad N \xrightarrow{\beta} L}{M \xrightarrow{\alpha\beta} L}$$

- ▶ Given $\alpha = (n_1, \dots, n_m) \in \mathbb{N}^*$, define $\|\alpha\| = \sum_{i=1}^m n_i$.

The Difference Cost Model

Lemma (Diamond Property Revisited)

If $M \xrightarrow{(n)} N$ and $M \xrightarrow{(m)} L$, then either $N \equiv L$ or there is P such that $N \xrightarrow{(m)} P$ and $L \xrightarrow{(n)} P$.

Theorem

For every term M , there are at most one normal form N and one integer n such that $M \xrightarrow{\alpha} N$ and $\|\alpha\| = n$.

Definition (Difference cost model)

If $M \xrightarrow{\alpha} N$, where N is a normal form, then $\text{Time}(M)$ is $\|\alpha\| + |M|$.
If M diverges, then $\text{Time}(M)$ is infinite.

The Difference Cost Model

Lemma (Diamond Property Revisited)

If $M \xrightarrow{(n)} N$ and $M \xrightarrow{(m)} L$, then either $N \equiv L$ or there is P such that $N \xrightarrow{(m)} P$ and $L \xrightarrow{(n)} P$.

Theorem

For every term M , there are at most one normal form N and one integer n such that $M \xrightarrow{\alpha} N$ and $\|\alpha\| = n$.

Definition (Difference cost model)

If $M \xrightarrow{\alpha} N$, where N is a normal form, then $\text{Time}(M)$ is $\|\alpha\| + |M|$.
If M diverges, then $\text{Time}(M)$ is infinite.

The Difference Cost Model

Lemma (Diamond Property Revisited)

If $M \xrightarrow{(n)} N$ and $M \xrightarrow{(m)} L$, then either $N \equiv L$ or there is P such that $N \xrightarrow{(m)} P$ and $L \xrightarrow{(n)} P$.

Theorem

For every term M , there are at most one normal form N and one integer n such that $M \xrightarrow{\alpha} N$ and $\|\alpha\| = n$.

Definition (Difference cost model)

If $M \xrightarrow{\alpha} N$, where N is a normal form, then $\mathit{Time}(M)$ is $\|\alpha\| + |M|$.
If M diverges, then $\mathit{Time}(M)$ is infinite.

Finite Sets and Strings

- ▶ Elements of finite sets and strings from a finite alphabet can be encoded as values in the lambda calculus.
- ▶ Elements of any finite set $A = \{a_1, \dots, a_n\}$ can be encoded as follows:

$$\ulcorner a_i \urcorner^A \equiv \lambda x_1 \dots \lambda x_n. x_i$$

- ▶ Let $\Sigma = \{a_1, \dots, a_n\}$ be a finite alphabet. A string $s \in \Sigma^*$ can be represented by a value $\ulcorner s \urcorner^{\Sigma^*}$ as follows, by induction on s :

$$\begin{aligned}\ulcorner \epsilon \urcorner^{\Sigma^*} &\equiv \lambda x_1 \dots \lambda x_n. \lambda y. y \\ \ulcorner a_i u \urcorner^{\Sigma^*} &\equiv \lambda x_1 \dots \lambda x_n. \lambda y. x_i \ulcorner u \urcorner^{\Sigma^*}\end{aligned}$$

- ▶ Tuples can be encoded as follows:

$$\langle V_1, \dots, V_n \rangle = \lambda x. x V_1 \dots V_n$$

A Fixpoint Operator

- ▶ Arbitrary recursion can be represented in the calculus (no types).
- ▶ We denote by H the term MM , where

$$M \equiv \lambda x. \lambda f. f(\lambda z. xxfz).$$

- ▶ H is a call-by-value fixed-point operator: for every N , there is α such that

$$HN \xrightarrow{\alpha} N(\lambda z. HNz)$$

$$\|\alpha\| = O(|N|)$$

The Theorem

- ▶ Given a Turing Machine \mathcal{M} , we can easily build lambda terms efficiently performing the following operations:
 - ▶ Building the initial configuration of \mathcal{M} from a given string.
 - ▶ Computing one transition step of \mathcal{M} .
 - ▶ Extracting a string content from a final configuration of \mathcal{M} .
- ▶ As a consequence, we have:

Theorem

If $f : \Delta^ \rightarrow \Delta^*$ is computed by a Turing machine \mathcal{M} in time g , then there is a term $U(\mathcal{M}, \Delta)$ such that for every $u \in \Delta^*$ there is α with $U(\mathcal{M}, \Delta) \ulcorner u \urcorner^{\Delta^*} \xrightarrow{\alpha} \ulcorner f(u) \urcorner^{\Delta^*}$ and $\|\alpha\| = O(g(|u|))$*

The Theorem

- ▶ Given a Turing Machine \mathcal{M} , we can easily build lambda terms efficiently performing the following operations:
 - ▶ Building the initial configuration of \mathcal{M} from a given string.
 - ▶ Computing one transition step of \mathcal{M} .
 - ▶ Extracting a string content from a final configuration of \mathcal{M} .
- ▶ As a consequence, we have:

Theorem

If $f : \Delta^ \rightarrow \Delta^*$ is computed by a Turing machine \mathcal{M} in time g , then there is a term $U(\mathcal{M}, \Delta)$ such that for every $u \in \Delta^*$ there is α with $U(\mathcal{M}, \Delta) \ulcorner u \urcorner^{\Delta^*} \xrightarrow{\alpha} \ulcorner f(u) \urcorner^{\Delta^*}$ and $\|\alpha\| = O(g(|u|))$*

DeBruijn's Representation

- ▶ To each lambda term M we can associate a string $M^\# \in \{\lambda, @, 0, 1, \blacktriangleright\}^+$ in the standard deBruijn way, writing @ for (prefix) application.
- ▶ For example, if $M \equiv (\lambda x.xy)(\lambda x.\lambda y.\lambda z.x)$, then $M^\#$ is

$@\lambda@\blacktriangleright 0\blacktriangleright \lambda\lambda\lambda\blacktriangleright 10$

- ▶ In other words, free occurrences of variables are translated into \blacktriangleright , while bounded occurrences of variables are translated into $\blacktriangleright s$, where s is the binary representation of the deBruijn index for that occurrence.

The Theorem

- ▶ Summing up, $M^\#$ is a string in a fixed, finite alphabet for any lambda term M .
- ▶ As a consequence, we can manipulate lambda terms with Turing Machines.

Lemma

If $M \rightarrow^n N$, then $n \leq \text{Time}(M)$ and $|N| \leq \text{Time}(M)$.

Theorem

There is a 7-tapes Turing Machine \mathcal{R} computing the normal form of any lambda-term M in $O((\text{Time}(M))^4)$ steps.

The Theorem

- ▶ Summing up, $M^\#$ is a string in a fixed, finite alphabet for any lambda term M .
- ▶ As a consequence, we can manipulate lambda terms with Turing Machines.

Lemma

If $M \rightarrow^n N$, then $n \leq \text{Time}(M)$ and $|N| \leq \text{Time}(M)$.

Theorem

There is a 7-tapes Turing Machine \mathcal{R} computing the normal form of any lambda-term M in $O((\text{Time}(M))^4)$ steps.

The Theorem

- ▶ Summing up, $M^\#$ is a string in a fixed, finite alphabet for any lambda term M .
- ▶ As a consequence, we can manipulate lambda terms with Turing Machines.

Lemma

If $M \rightarrow^n N$, then $n \leq \text{Time}(M)$ and $|N| \leq \text{Time}(M)$.

Theorem

There is a 7-tapes Turing Machine \mathcal{R} computing the normal form of any lambda-term M in $O((\text{Time}(M))^4)$ steps.

Closed Values as a Partial Combinatory Algebra

- ▶ If U and V are closed values and UV has a normal form W (which must be a closed value), then we will denote W by $\{U\}(V)$.
- ▶ In this way, we can give Ξ the status of a partial applicative structure, which turns out to be a partial combinatory algebra.
- ▶ $Time(\{U\}(V))$ is simply $Time(UV)$ (if it exists). This induces a finer structure on Ξ .
- ▶ This has been exploited in [DallagoHofmann05], where lambda terms are realizers in a quantitative semantic frameworks.

Some Interesting Properties

- ▶ Notice there are terms M_{swap} , M_{cont} , M_{eval} and M_{curry} such that:
 - ▶ $\{M_{swap}\}(\langle V, U \rangle) = \langle U, V \rangle$ and $Time(\{M_{swap}\}(\langle V, U \rangle))$ is constant.
 - ▶ $\{M_{cont}\}(V) = \langle V, V \rangle$ and $Time(\{M_{cont}\}(\langle V \rangle))$ is linear (in $|V|$).
 - ▶ $\{M_{eval}\}(\langle V, U \rangle) = \{V\}(U)$ and $Time(\{M_{eval}\}(\langle V, U \rangle)) = Time(\{V\}(U)) + c$, where c is a constant.
 - ▶ $\{M_{curry}\}(\langle V \rangle) = U$, U is the curried version of V and $Time(\{M_{curry}\}(V))$ is constant. Moreover, the time complexity of U is “the same” as that of V .
- ▶ More generally, $M \leq |M|$ whenever V is a linear term.

Conclusions and Further Work

- ▶ The difference cost model is invariant and reasonably parsimonious.
- ▶ We have not treated space.
- ▶ Further work:
 - ▶ If we introduce some form of sharing, our counterexample fails. What happens in presence of sharing? (some results are on the way).
 - ▶ Is it possible to generalize the cost model to call-by-name (with a strategy)?
 - ▶ Go toward categorical semantics.