The lazy call-by-value $\lambda$-calculus

Lavinia Egidi*, Furio Honsell**, Simona Ronchi della Rocca*

* Dipartimento di Informatica, Università degli Studi di Torino (Italy)
** Dipartimento di Matematica e Informatica, Università degli Studi di Udine (Italy)

Introduction.

In the implementation of many functional programming languages, parameters are evaluated before being passed (call-by-value evaluation) and function bodies are evaluated only when parameters are supplied (lazy evaluation). These two features first appeared in the implementation of the language ISWIM given by Landin using the SECD machine [5]. The ISWIM–SECD system can be seen as the paradigm implementation of the functional kernel of several programming languages such as ML.

The call-by-value $\lambda$-calculus ($\lambda^v$) was introduced by Plotkin [8], in order to reason about equivalence and termination of ISWIM programs interpreted by the SECD machine. The $\lambda^v$-calculus is obtained from the $\lambda$-calculus by restricting the $\beta$-rule to redexes whose operand is a value (i.e. a constant, a variable or a function). Plotkin showed that given a $\lambda^v$-term $M$ the SECD machine evaluates $M$ to an output value $v$ if and only if $M \rightarrow^v v$ according to the lazy-leftmost reduction strategy. This strategy being the one which reduces the leftmost redex not inside a lambda abstraction.

Once we have a sequential interpreter $I$ for a programming language, the most natural behavioural equivalence between programs is the one determined by termination observations of the computation processes. This relation, termed $I$-operational equivalence, is defined as:

$$M \equiv_I N \iff \text{for all closing context } C. (C[M] \text{ terminates } \iff C[N] \text{ terminates}).$$

The $\lambda^v$-calculus provides a sound (albeit incomplete) formal system for establishing SECD-operational equalities between $\lambda^v$-terms ($\equiv_{SECD}$).

The model theory of the $\lambda^v$-calculus is particularly interesting. In fact, contrary to ordinary pure $\lambda$-calculus, pure $\lambda^v$-calculus has a canonical denotational model in any of the usual categories for denotational semantics. Namely the initial solution of the equation:

$$D \equiv^v D \iff \text{for all term vectors } P_1, \ldots, P_n. (M P_1 \ldots P_n \text{ terminates } \iff N P_1 \ldots P_n \text{ terminates}).$$

In this paper we study some topics in the model theory of the $\lambda^v$-calculus in view of a deeper understanding of the operational semantics of the SECD machine. First we deal with the general principles. Then we study the theory induced by the model $D$ above. This theory however is too weak, since it is strictly included in $\equiv_{SECD}$. A phenomenon similar to those described in [3] and [9] arises in this context. The SECD machine is sequential and hence not all strict continuous functions are definable by an ISWIM term. In order to find a fully abstract model we have to take into account also models where not all the strict continuous functions are representable. We build then a fully abstract model by means of a notion of call-by-value applicative bisimulation. Similar constructions were carried out by Plotkin [10], Abramsky [1], Mulmuley [6] and Ong [7] for different calculi. This construction is interesting in itself since it provides an alternative observational characterization of the operational equivalence $\equiv_{SECD}$ in terms of a restricted class of "computational experiments", i.e. applicative contexts. Formally

Formally

$$M \equiv^v N \iff \text{for all term vectors } P_1, \ldots, P_n. (M P_1 \ldots P_n \text{ terminates } \iff N P_1 \ldots P_n \text{ terminates}).$$

The fully abstract model produced is very thin, it amounts to a term model equipped with a richer structure. However this result is remarkable since it is not immediate that $\equiv^v$ is a congruence relation.

---

1 Work partially supported by MURST 40% and 60% grants and EEC "Project Stimulation ST2J/0374/C(EDB): Lambda Calculus Type".
Finally we give a logic for reasoning about lazy call-by-value programs, which takes the form of a type assignment system for $\lambda$-terms, and can be viewed as the call-by-value analogue of the intersection type assignment system of Coppo-Dezani [2]. This is inspired by a "concrete" logical presentation of the canonical model. 

*Added in proof.* Since this paper was submitted to MFCS 91, two papers related to this topic have appeared [12] and [13].

1. The ISWIM–SECD system.

In this section we recall basic definitions and facts concerning the language ISWIM, the SECD machine [5] and the lazy–call-by-value $\lambda$–calculus $\lambda \beta_\nu$ [8]. We illustrate also the use of $\lambda \beta_\nu$ for describing the operational semantics of terms induced by the SECD machine.

1.1. Pure ISWIM.

As remarked in the introduction, ISWIM is the paradigm of many functional programming languages. For simplicity, here we deal with the pure version of it, i.e. ISWIM without constants. Thus ISWIM programs turn out to be terms of the pure $\lambda$–calculus.

The language of pure ISWIM is defined by the pair $(\Lambda, Val)$, where $\Lambda$ is the set of terms of pure $\lambda$–calculus over a set of variables $\text{Var}$, and $Val = \text{Var} \cup \{\lambda x. M \mid M \in \Lambda\}$ is the set of values. Free and bound variables are also defined as usual and, for any term $M$, $\text{FV}(M)$ is the set of its free variables. $\Lambda^* \subseteq \Lambda$ is the set of closed terms. We also adopt usual conventions about parentheses, and usual notations for contexts.

A complete definition of a programming language is given once we specify an evaluation function describing the way programs are executed. Landin defined the evaluation function $\text{eval} : \Lambda \rightarrow \Lambda$ for ISWIM, by means of an abstract transition machine, the SECD machine [5]. Plotkin showed that the SECD machine can be described through a reduction strategy on $\Lambda$, using a suitable notion of reduction.

**Definition 1.** The rules for the call–by–value reduction are:

\[(\alpha) (\lambda x. M) \rightarrow_{\nu} (\lambda y. M[y/x])\]

\[(\beta_\nu) (\lambda x. M) N \rightarrow_{\nu} M[x/N] \quad \text{if } N \in Val.\]

The contextual, reflexive and transitive closure of $\rightarrow_{\nu}$ is denoted by $\rightarrow_{\nu}^\ast$; the equivalence induced by the reduction is denoted by $=_{\nu}$. We shall write $M \rightarrow_{\nu} Val$ to indicate that $M$ reduces to a value; we shall call such a term valuable.

**Definition 2.** The SECD reduction strategy ($\rightarrow_{\text{SECD}}$) is defined as follows:

let $M = M'[\lambda x. P]Q$, where $(\lambda x. P)Q$ is the leftmost $\beta_\nu$–redex which is not in the scope of a $\lambda$, then $M \rightarrow_{\text{SECD}} M'[P[x/Q]]$.

The reflexive and transitive closure of $\rightarrow_{\text{SECD}}$ is denoted by $\rightarrow_{\text{SECD}}$; the equivalence induced by $\rightarrow_{\text{SECD}}$ is denoted by $=_{\text{SECD}}$. We shall write $M \rightarrow_{\text{SECD}} Val$ to indicate that $M$ SECD–reduces to a value.

The SECD reduction strategy is a faithful simulation of the behaviour of the SECD machine in the following sense (see [8]):

the SECD machine, given an input $M$, "halts on an output value $V" \iff M \rightarrow_{\text{SECD}} V."

Therefore the reduction strategy SECD can safely be taken as defining the evaluation function of ISWIM.

In order to study the properties of pure ISWIM abstractly, we proceed as Plotkin [8] and introduce the $\lambda \beta_\nu$–calculus as ISWIM equipped with the $\rightarrow_{\nu}$ reduction defined above.

The calculus $\lambda \beta_\nu$ is lazy since all abstractions are considered to be values regardless of their body; it is call–by–value since a reduction is performed only if the argument to which the abstraction is applied is a value.