INFORMATION AND CONTROL 55, 89–101 (1982)

A Consistent Extension of the Lambda-Calculus as a Base for Functional Programming Languages

KLAUS J. BERKLING

Institut für Informationssystemforschung,
Gesellschaft für Mathematik und Datenverarbeitung, 5205 St. Augustin 1, West Germany

AND

ELFRIEDE FEHR*

Lehrstuhl für Informatik II, RWTH-Aachen, 5100 Aachen, West Germany

Church's lambda-calculus is modified by introducing a new mechanism, the lambda-bar operator $\not\equiv$, which neutralizes the effect of one preceding lambda binding. This operator can be used in such a way that renaming of bound variables in any reduction sequence can be avoided, with the effect that efficient interpreters with comparatively simple machine organization can be designed. It is shown that any semantic model of the pure λ-calculus also serves as a model of this modified reduction calculus, which guarantees smooth semantic theories. The Berkling Reduction Language (BRL) is a new functional programming language based upon this modification.

1. INTRODUCTION

Functional (applicative) programming languages, such as LISP (McCarthy *et al.*, 1965), KRC (Turner, 1981), PCF (Plotkin, 1977) etc., are in general based upon the lambda-calculus (Church, 1941). Although operational and denotational semantics of the lambda-calculus are by now well understood (Gordon, 1975), most of the existing implementations of the lambda-calculus correspond to incomplete versions or inconsistent extensions of the axioms of the lambda-calculus. The reason for this is mainly the fact that the standard reduction of leftmost–outermost β-redexes with preceding tests on variable conflicts and appropriate renaming is highly inefficient, when implemented on or simulated by a machine.

Interpreters of LISP and LISP-like languages as the one given in

* Correspondence should be sent to Dr. Elfriede Fehr.

0019-9958/82 $2.00

(McCarthy *et al.*, 1965) use in general a dynamic binding mechanism, which violates the semantics of the underlying lambda-calculus as shown in (Eick and Fehr, 1983). Another well-known implementation of the lambda-calculus is the SECD-machine (Landin, 1964), which supports the correct scope-rules of the lambda-calculus but as shown in (McGowan, 1970) fails to reduce all expressions having a normal form, because functional arguments cannot be treated appropriately. Similar problems arise with implementation on a cellular computer architecture as introduced in (Magó, 1979), which are suitable only for a restricted class of functional languages as, e.g., FP and FFP (Backus, 1978), because objects of higher functional types cannot be handled.

Other implementations such as the graph-machine introduced in (Wadsworth, 1971) or the combinator reduction introduced in (Turner, 1979) make a radical change in the representation of lambda-expressions. They work on labelled graphs or purely combinatory expressions, with the effect that the original lambda-expressions get lost and intermediate results can hardly be understood by the programmer. All these deficiencies can be overcome by implementing a variant of the lambda-calculus, which maintains the original structure and naming of an expression. This variant is obtained by adding an unbinding mechanism lambda-bar ($\neq$) to the language, which neutralizes the effect of one preceding lambda-binding. For example, the variable x occurs free in the expression $\lambda x. \neq x$ but bound in the expression $\lambda x \cdot \lambda x \cdot \neq x$.

The benefit of this extension is that β-conversion can be performed without renaming of variables by systematically using the lambda-bar mechanism. As a result machine models of languages based upon this extension have an uncomplicated machine structure and run very efficiently.

The corresponding formalism was first introduced in (De Bruijn, 1972). De Bruijn uses an implementation of this mechanism in his AUTOMATH-project (De Bruijn, 1980) and shows that it is very efficient for automatic formula manipulation. In (Berkling, 1976a) the same mechanism was introduced independently. Berkling developed in (Berkling, 1976b) a reduction language BRL which is an extension of the lambda-calculus not only by a certain set of base operations, such as conditionals, arithmetical, Boolean, and list operations, but also by the unbinding mechanism lambda-bar. A machine implementation of BRL was first simulated in *PL/I* (Hommes, 1977) and then a hardware-model was built (Kluge, 1979), which started operating in 1978 and has since shown a satisfactory performance. In particular the machine supports different reduction strategies, such as a finite number of call-by-value reductions followed by a call-by-name reduction. Hence the efficiency of the BRL-machine can again be increased with more efficient strategies.

The semantic effect of the lambda-bar operation on the lambda-calculus

was until now not very clear, since there existed only the syntactical and operational descriptions of it. In this paper we want to give a denotational semantics to it and a proof that it consistently extends the lambda-calculus.

2. A Short Introduction to BRL

As in most functional programming languages there is only one syntactical category, expressions, in the Berkling Reduction Language. In this paper we do not mention every feature of BRL but rather point out the different ways of forming new expressions from given ones.

(i) "simple expressions" in BRL are built up from variables and constants using arithmetical, logical, and list operations, as well as conditionals, e.g.,

(a) $(3.1415 * (\text{radius} * \text{radius}))$,

(b) $4.3 : 3.7$,

(c) *if* $x > 0$ *then* 1 *else if* $x = 0$ *then* 0 *else* -1.

(ii) "abstractions" can be produced from any expression e and identifier x by writing:

sub x *in* e

which is a sugared version of the lambda-expression

$$\lambda x \cdot e$$

and denotes a function with formal parameter x, which substitutes a given argument at each free occurrence of x in e. For example,

(a) *sub* radius *in* $(3.1415 * (\text{radius} * \text{radius}))$,

(b) *sub* r *in* sub h *in* $(((3.1415 * (r * r)) * h)$.

(iii) "combinations" are made of two arbitrary expressions f and g, where one takes the function part and the other one takes the argument part. The corresponding BRL-expression just reads:

apply f *to* g

again a sugared version of the λ-expression $(f\ g)$. For example,

(a) *apply sub* r *in* $(3.1 * (r * r))$ to 24.3.

During an execution, the above expression will be reduced in the first step to $(3.1 * (24.3 * 24.3))$ and in the next step to its value 1830.319. To illustrate the effect of the unbinding mechanism $\not\#$, consider the λ-expression:

$(\lambda x \cdot \lambda y \cdot x\ y)$. Because y occurs free in the argument and bound in the function body, a renaming, say $(\lambda x \cdot \lambda z \cdot x\ y)$, has to be performed before β-reduction can reduce the term to $\lambda z \cdot y$.

The corresponding expression in BRL reads:

apply sub x in sub y in x to y.

After one step of reduction it turns into *sub y in #y*. The formal rule corresponding to β-reduction will be presented in the next section.

(iv) "recursive expressions" are introduced explicitly into BRL although one could use the expression corresponding to the Y-operator of the λ-calculus. For any expression e and variable f we can build the BRL expression:

rec f : *e*

which corresponds to an equation $f = e$ or to the λ-expression $(Y\ \lambda f \cdot e)$. Consider the definition of the factorial as an example:

(a) *rec* fac : *sub n in if* $(n = 1)$ *then* 1

else $(n *$ *apply* fac *to* $(n = 1))$.

(v) Other concepts of BRL are convenient operations for tree manipulations, for pattern matching and some facilities to save function definitions. They will not be presented in this paper, but can be looked at in (Hommes and Schlütter, 1979).

A very nice feature of the BRL-machine is that when working at a terminal, program development runs interactively and long subexpressions are automatically reduced to one symbol, which can be easily expanded using the cursor.

3. Syntax and Conversion Rules of the Reduction-Calculus

In this section we want to give a formal description of the lambda-calculus modification which is the support for BRL. In order to ease an immediate comparison with the λ-notation we shall use pure λ-terms rather than BRL-expressions.

Definition 1 (Syntax). Let χ be a denumerable set of variables. The set $\mathscr{E}$ of reduction terms is given inductively by:

(i) $\#^n x \in \mathscr{E}$ for all $x \in \chi$, $n \in \mathbb{N}$, $n \geqslant 0$,

(ii) $(t_1\ t_2) \in \mathscr{E}$ for all $t_1, t_2 \in \mathscr{E}$,

(iii) $\lambda x \cdot t \in \mathscr{E}$ for all $x \in \chi$, $t \in \mathscr{E}$.

In (i) the symbol $\#^n$ can be read as the n-fold application of $\#$, which reduces to the identity in the case of $n = 0$. As indicated in the Introduction, β-conversion will systematically make use of the unbinding operation $\#$ (lambda-bar). This principle is formalized by the metaconstructor Π (protector), which augments the number of unbindings (protections) of a particular variable by one.

DEFINITION 2 (Π). Let $x \in \chi$, $n \in \mathbb{N}$, and $t \in \mathscr{E}$. The reduction term $\Pi\#^n xt$ is given by induction on the structure of t:

(i) $\Pi\#^n x\#^m y = \#^{m+1} y$ if $x = y$ and $m \geqslant n$,

$= \#^m y$ otherwise;

(ii) $\Pi\#^n x(t_1\ t_2) = (\Pi\#^n xt_1\ \Pi\#^n xt_2)$;

(iii) $\Pi\#^n x\lambda y \cdot t = \lambda y \cdot \Pi\#^{n+1} xt$ if $x = y$,

$= \lambda y \cdot \Pi\#^n xt$ if $x \neq y$.

Now the substitution operator $\$_s^v t$, which substitutes s for v in t, can be defined without using the notions of free and bound occurrences of variables.

DEFINITION 3 ($). Let v be $\#^n x$ for some $n \in \mathbb{N}$, $x \in \chi$ and let $s, t \in \mathscr{E}$. The reduction term $\$_s^v t$ is inductively defined by:

(i) $\$_s^v \#^m y = s$ if $v = \#^m y$, i.e. $m = n$ and $x = y$,

$= \#^{m-1} y$ if $x = y$ and $m > n$,

$= \#^m y$ otherwise;

(ii) $\$_s^v(t_1\ t_2) = (\$_s^v t_1\ \$_s^v t_2)$;

(iii) $\$_s^v \lambda y \cdot t = \lambda y \cdot \$_{\Pi ys}^{\#v} t$ if $x = y$,

$= \lambda y \cdot \$_{\Pi ys}^v t$ if $x \neq y$.

Observe that by 3(i) occurrences of $\#^n x$ with $m > n$ are transformed into $\#^{m-1} x$. This is due to the fact that in the reduction calculus substitution of s for x in t is performed as value of the redex $(\lambda x \cdot t\ s)$, hence one protection of x in t becomes superfluous. The only reduction rule in this calculus corresponds to the β-conversion rule of the λ-calculus, but as clashes of variables cannot occur, all redexes can be reduced without a prior renaming.

DEFINITION 4 (β'). Again, let $x \in \chi$, and let t and s be arbitrary reduction terms. Then the binary relation $\underset{\beta'}{\rightarrow}$ is defined as follows:

$$(\lambda x \cdot t\, s) \underset{\beta'}{\longrightarrow} \$_s^x t.$$

A detailed description of the reduction calculus can be found in (Berkling, 1976a) where another simplification which uses only one variable is presented too.

4. Denotational Semantics and Consistency Proof

In this section we show that the reduction calculus as introduced above has a neat denotational semantics in any model of the λ-calculus, as for example the $P\omega$-model (Scott, 1976), D_∞ (Scott, 1972), or any other. For the rest of this paper let $\mathcal{M}$ be such a model and let φ denote the retraction from $\mathcal{M}$ onto $[\mathcal{M} \to \mathcal{M}]$ with right inverse φ^{-1}, where $[\mathcal{M} \to \mathcal{M}]$ denotes the set of continuous functions from $\mathcal{M}$ to $\mathcal{M}$. As a first step we shall model the effect of the protection operator Π on environments.

DEFINITION 5 ($\Pi_{n,x}$). Let $x \in \chi$ and $n \in \mathbb{N}$. The operator $\Pi_{n,x}$ on an environment ρ, i.e., a mapping from χ to $\mathcal{M}$, is given by

$$\begin{aligned} \Pi_{n,x}(\rho)(\#^k y) &= \rho(\#^{k+1} x) && \text{if } x = y \text{ and } k \geqslant n \\ &= \rho(\#^k y) && \text{otherwise.} \end{aligned}$$

Another useful definition serves to modify an environment such that one protection of a variable x is neglected, and a new value m is given to unprotected occurrences of x.

DEFINITION 6 ($\rho[x \# m]$). For $x \in \chi$, $m \in \mathcal{M}$ and an environment ρ let $\rho[x = m]$ be a new environment given by

$$\begin{aligned} \rho[x \# m](\#^k y) &= \rho(\#^k y) && \text{if } x \neq y, \\ &= m && \text{if } x = y \text{ and } k = 0, \\ &= (\#^{k-1} x) && \text{otherwise.} \end{aligned}$$

The next lemma shows how the environment transforming operators $\Pi_{n,x}$ and $[y \# m]$ commute.

LEMMA 7. *Let $n, k \in \mathbb{N}$, $x, y \in \chi$, $m \in \mathcal{M}$ and ρ be an environment. Then the following holds:*

$$\begin{aligned} \Pi_{n,x}(\rho)[y \# m] &= \Pi_{n+1,x}(\rho[y \# m]) && \text{if } x = y, \\ &= \Pi_{n,x}(\rho[y \# m]) && \text{if } x \neq y. \end{aligned}$$

Proof. Let $z \in \chi$ and $k \in \mathbb{N}$.

Case 1 $(x = y)$.

$$
\begin{aligned}
&\Pi_{n,x}(\rho)[x \# m](\#^k z) \\
&\quad = \Pi_{n,x}(\rho)(\#^k z) && \text{if } x \neq z, \\
&\quad = m && \text{if } x = z \text{ and } k = 0, \\
&\quad = \Pi_{n,x}(\rho)(\#^{k-1} x) && \text{if } x = z \text{ and } k > 0 && \text{by Definition 6,} \\
&\quad = \rho(\#^k z) && \text{if } x \neq z, \\
&\quad = m && \text{if } x = z \text{ and } k = 0, \\
&\quad = \rho(\#^k x) && \text{if } x = z \text{ and } k > 0 \text{ and } k - 1 \geqslant n, \\
&\quad = \rho(\#^{k-1} x) && \text{otherwise} && \text{by Definition 5,} \\
&\Pi_{n+1,x}(\rho[x \# m])(\#^k z) \\
&\quad = \rho[x \# m](\#^{k+1} x) && \text{if } x = z \text{ and } k \geqslant n+1, \\
&\quad = \rho[x \# m](\#^k z) && \text{if } x \neq z \text{ or } k < n+1, && \text{by Definition 5,} \\
&\quad = \rho(\#^k x) && \text{if } x = z \text{ and } k \geqslant n+1, \\
&\quad = \rho(\#^k z) && \text{if } x \neq z, \\
&\quad = m && \text{if } x = z \text{ and } k = 0, \\
&\quad = \rho(\#^{k-1} x) && \text{if } x = z \text{ an } k > 0 \text{ and } k < n+1 \\
& && && \text{by Definition 6.}
\end{aligned}
$$

It is easy to check that the left-hand side and the right-hand side are equal in each subcase.

Case 2 $(x \neq y)$.

$$
\begin{aligned}
&\Pi_{n,x}(\rho)[y \# m](\#^k z) \\
&\quad = \Pi_{n,x}(\rho)(\#^k z) && \text{if } y \neq z, \\
&\quad = m && \text{if } y = z \text{ and } k = 0, \\
&\quad = \Pi_{n,x}(\rho)(\#^{k-1} y) && \text{if } y = z \text{ and } k > 0 && \text{by Definition 6,} \\
&\quad = \rho(\#^{k+1} x) && \text{if } y \neq z \text{ and } k \geqslant n \text{ and } x = z, \\
&\quad = \rho(\#^k z) && \text{if } y \neq z \text{ and } (x \neq z \text{ or } k < n), \\
&\quad = m && \text{if } y = z \text{ and } k = 0, \\
&\quad = \rho(\#^{k-1} y) && \text{if } y = z \text{ and } k > 0 && \text{by Definition 5,}
\end{aligned}
$$

$$
\begin{aligned}
&\Pi_{n,x}(\rho[y \# m])(\#^k z)\\
&\quad = \rho[y \# m](\#^{k+1} z) && \text{if } x = z \text{ and } k \geqslant n,\\
&\quad = \rho[y \# m](\#^k z) && \text{if } x \neq z \text{ or } k < n && \text{by Definition 5,}\\
&\quad = \rho(\#^{k+1} z) && \text{if } x = z \text{ and } k \geqslant n \text{ and } y \neq z,\\
&\quad = \rho(\#^k z) && \text{if } y \neq z \text{ and } (x \neq z \text{ or } k < n),\\
&\quad = m && \text{if } y = z \text{ and } k = 0,\\
&\quad = \rho(\#^{k-1} y) && \text{if } y = z \text{ and } k > 0 && \text{by Definition 6.}
\end{aligned}
$$

Again it is easy to verify that both sides of the equation of the lemma are equal in all subcases. ∎

Now we can elegantly formulate the denotational semantics of the reduction terms.

DEFINITION 8 (Semantics $[\![\ ]\!]$). The semantics of a term t with respect to an environment $\rho : \chi \to \mathscr{M}$ in the model $\mathscr{M}$ is given inductively by:

(i) $[\![v]\!]\rho = \rho(v)$ for each $v = \#^n x$, $n \in \mathbb{N}$, $x \in \chi$,

(ii) $[\![(t_1\ t_2)]\!]\rho = \varphi([\![t_1]\!]\rho)([\![t_2]\!]\rho)$ for $t_1, t_2 \in \mathscr{E}$,

(iii) $[\![\lambda x \cdot t]\!]\rho = \varphi^{-1}(m \mapsto [\![t]\!]\rho[\![x \# m]\!])$ for $x \in \chi$, $t \in \mathscr{E}$.

This definition exhibits the fact that our reduction calculus is semanticly a clean extension of the λ-calculus, because terms without occurrences of $\#$ obtain exactly the usual λ-semantics. Before proving that this semantic definition is compatible with the β'-reduction, we have to show in three lemmas, how he protection operator Π and the substitution operator \$ behave on the semantic level.

LEMMA 9. *Let $n \in \mathbb{N}$, $x \in \chi$, $t \in \mathscr{E}$, and ρ be an environment. The following equation holds:*

$$[\![\Pi \#^n x\ t]\!]\rho = [\![t]\!]\Pi_{n,x}(\rho).$$

Proof. Induction on the structure of t:

$$
\begin{aligned}
\text{(i)}\quad [\![\Pi \#^n x \#^k y]\!]\rho &= [\![\#^{k+1} y]\!]\rho && \text{if } x = y \text{ and } k \geqslant n,\\
&= [\![\#^k y]\!]\rho && \text{otherwise} && \text{by Definition 2,}\\
&= \Pi_{n,x}(\rho)(\#^k y) && && \text{by Definition 8(i) and 5,}\\
&= [\![\#^k y]\!]\Pi_{n,x}(\rho) && && \text{by Definition 8(i).}
\end{aligned}
$$

(ii) By Definition 2(ii) Π distributes to both components of the combination, thus the induction hypothesis is applicable.

(iii) $[\![\Pi \#^n x\, \lambda y \cdot t]\!]\, \rho$

$= [\![\lambda y \cdot \Pi \#^{n+1} x\, t]\!]\, \rho$ if $x = y$,

$= [\![\lambda y \cdot \Pi \#^n x\, t]\!]\, \rho$ if $x \neq y$

by Definition 2(iii)

$= \varphi^{-1}(m \mapsto [\![\Pi \#^{n+1} x\, t]\!]\, \rho[y \# m])$ if $x = y$,

$= \varphi^{-1}(m \mapsto [\![\Pi \#^n x\, t]\!]\, \rho[y \# m])$ if $x \neq y$

by Definition 8(iii)

$= \varphi^{-1}(m \mapsto [\![t]\!]\, \Pi_{n+1,x}(\rho[y \# m]))$ if $x = y$,

$= \varphi^{-1}(m \mapsto [\![t]\!]\, \Pi_{n,x}(\rho[y \# m]))$ if $x \neq y$

by induction hypothesis,

$= \varphi^{-1}(m \mapsto [\![t]\!]\, \Pi_{n,x}(\rho)[y \# m])$ by Lemma 7,

$= [\![\lambda y \cdot t]\!]\, \Pi_{n,x}(\rho)$ by Definition 8(iii). ∎

LEMMA 10. *Let x, m and ρ be as above. The following equation holds:*

$$\Pi_{0,x}(\rho[x \# m]) = \rho.$$

Proof.

$\Pi_{0,x}(\rho[x \# m])(\#^k y) = \rho[x \# m](\#^{k+1} y)$ if $x = y$,

$= \rho[x \# m](\#^k y)$ if $x \neq y$

by Definition 5,

$= \rho(\#^k y)$ by Definition 6. ∎

In our next lemma we shall use a generalization of Definition 6, namely, the change of an environment on a protected variable.

DEFINITION 11 ($\rho[v\#m]$). Let $v = \#^n x$, $n \in \mathbb{N}$, $x \in \chi$, $m \in \mathcal{M}$ and ρ be an environment. For $k \in \mathbb{N}$ and $y \in \chi$ let $\rho[v\#m](\#^k y)$ be given by

$$\begin{aligned} \rho[v\#m](\#^k y) &= m && \text{if } x = y \text{ and } k = n, \\ &= \rho(\#^{k-1} y) && \text{if } x = y \text{ and } k > n, \\ &= \rho(\#^k y) && \text{otherwise.} \end{aligned}$$

Observe that Definition 11 is consistent with Definition 6, i.e., $\rho[\#^0 x \# m] = \rho[x\#m]$.

LEMMA 12. *Again let $v = \#^n x$ for some $n \in \mathbb{N}$, $x \in \chi$ and let s, $t \in \mathscr{E}$, and ρ be an environment. The following equation holds:*

$$[\![\$_s^v t]\!]\rho = [\![t]\!][v\#[\![s]\!]\rho].$$

Proof. Induction on the structure of t:

(i) $$\begin{aligned} [\![\$_s^v \#^m y]\!]\rho &= [\![s]\!]\rho && \text{if } v = \#^m y, \\ &= [\![\#^{m-1} y]\!]\rho && \text{if } x = y \text{ and } m > n, \\ &= [\![\#^m y]\!]\rho && \text{otherwise} && \text{by Definition 3,} \\ &= \rho[v\#[\![s]\!]\rho](\#^m y) && && \text{by Definition 11 and 8(i}) \\ &= [\![\#^m y]\!]\rho[v\#[\![s]\!]\rho] && && \text{by Definition 8(i).} \end{aligned}$$

(ii) immediate by induction hypothesis.

(iii) *Case* 1 ($x = y$).

$$\begin{aligned} &[\![\$_s^v \lambda y \cdot t]\!]\rho \\ &= [\![\$_s^v \lambda x \cdot t]\!]\rho = [\![\lambda x \cdot \$_{\Pi x s}^{\#v} t]\!]\rho && \text{by definition 3(iii),} \\ &= \varphi^{-1}(m \mapsto [\![\$_{\Pi x s}^{\#v} t]\!]\rho[x\#m]) && \text{by Definition 8(iii),} \\ &= \varphi^{-1}(m \mapsto [\![t]\!]\rho[x\#m][\#v \# [\![\Pi x s]\!]\rho[x\#m]]) && \text{by induction hypothesis,} \\ &= \varphi^{-1}(m \mapsto [\![t]\!]\rho[x\#m][\#v \# [\![s]\!]\Pi_{0,x}(\rho[x\#m])]) && \text{by Lemma 9,} \\ &= \varphi^{-1}(m \mapsto [\![t]\!]\rho[x\#m][\#v \# [\![s]\!]\rho]) && \text{by Lemma 10;} \\ &[\![\lambda x \cdot t]\!]\rho[v \# [\![s]\!]\rho] \\ &= \varphi^{-1}(m \mapsto [\![t]\!]\rho[v \# [\![s]\!]\rho][x\#m]) && \text{by Definition 8(iii).} \end{aligned}$$

It remains to show, that the two environments $\rho_1 := \rho[x \# m][\# v \# [\![s]\!]\rho]$ and $\rho_2 := \rho[v \# [\![s]\!]\rho][x \# m]$ are equal.

$$
\begin{aligned}
\rho_1(\#^k z) &= [\![s]\!]\rho && \text{if } x = z \text{ and } k = n+1,\\
&= \rho[x \# m](\#^{k-1} z) && \text{if } x = z \text{ and } k > n+1,\\
&= \rho[x \# m](\#^k z) && \text{otherwise} \quad \text{by Definition 11,}\\
&= [\![s]\!]\rho && \text{if } x = z \text{ and } k = n+1,\\
&= \rho(\#^{k-2} z) && \text{if } x = z \text{ and } k > n+1,\\
&= \rho(\#^k z) && \text{if } x \neq z,\\
&= m && \text{if } x = z \text{ and } k = 0,\\
&= \rho(\#^{k-1} x) && \text{otherwise} \quad \text{by Definition 6,}
\end{aligned}
$$

$$
\begin{aligned}
\rho_2(\#^k z) &= \rho[v \# [\![s]\!]\rho](\#^k z) && \text{if } x \neq z,\\
&= m && \text{if } x = z \text{ and } k = 0,\\
&= \rho[v \# [\![s]\!]\rho](\#^{k-1} x) && \text{otherwise} \quad \text{by Definition 6,}\\
&= \rho(\#^k z) && \text{if } x \neq z,\\
&= m && \text{if } x = z \text{ and } k = 0,\\
&= [\![s]\!]\rho && \text{if } x = z \text{ and } k - 1 = n,\\
&= \rho(\#^{k-2} z) && \text{if } x = z \text{ and } k - 1 = n,\\
&= \rho(\#^{k-1} z) && \text{otherwise.}
\end{aligned}
$$

So ρ_1 and ρ_2 are equal in all subcases.

Case 2 ($x \neq y$).

$$
\begin{aligned}
&[\![\$_s^v \lambda y \cdot t]\!]\rho\\
&\quad = [\![\lambda y \cdot \$_{\Pi y s}^v t]\!]\rho && \text{by Definition 3(iii),}\\
&\quad = \varphi^{-1}(m \mapsto [\![\$_{\Pi y s}^v t]\!]\rho[y \# m]) && \text{by Definition 8(iii),}\\
&\quad = \varphi^{-1}(m \mapsto [\![t]\!]\rho[y \# m][v \# [\![\Pi y s]\!]\rho[y \# m]]) && \text{by induction hypothesis,}\\
&\quad = \varphi^{-1}(m \mapsto [\![t]\!]\rho[y \# m][v \# [\![s]\!]\, \Pi_{0,y}(\rho[y \# m])]) && \text{by Lemma 9,}\\
&\quad = \varphi^{-1}(m \mapsto [\![t]\!]\rho[y \# m][v \# [\![s]\!]\rho]) && \text{by Lemma 10,}\\
&[\![\lambda y \cdot t]\!]\rho[v \# [\![s]\!]\rho]\\
&\quad = \varphi^{-1}(m \mapsto [\![t]\!]\rho[v \# [\![s]\!]\rho][y \# m]) && \text{by Definition 8(iii).}
\end{aligned}
$$

It is easy to verify that the environments $\rho[y \# m][v \# [\![s]\!]\rho]$ and $\rho[v \# [\![s]\!]\rho][y \# m]$ are equal due to the fact that $x \neq y$. ∎

We can now prove our main theorem which ensures the consistency of the β'-conversion with the laws of the λ-calculus.

THEOREM 13. *For all $s, t \in \mathscr{E}$ and $x \in \chi$ the following equation holds:*

$$[\![(\lambda x \cdot t\, s)]\!]\rho = [\![\$_s^x\, t]\!]\rho,$$

i.e., β'-reduction preserves meaning.

Proof.

$[\![(\lambda x \cdot t\, s)]\!]\rho$

$= \varphi([\![\lambda x \cdot t]\!]\rho)([\![s]\!]\rho)$ by definition 8(ii),

$= \varphi(\varphi^{-1}(m \mapsto [\![t]\!]\rho[x \# m]))([\![s]\!]\rho)$ by Definition 8(iii),

$= [\![t]\!]\rho[x \# [\![s]\!]\rho]$ because φ^{-1} is the right inverse of φ,

$= [\![\$_s^x t]\!]\rho$ by Lemma 12. ∎

Now we can make a last observation, which guarantees that β'-conversion is also complete.

THEOREM 14. *Let $t \in \mathscr{E}$. If t has a normal-form t', then $t \overset{*}{\underset{\beta'}{\to}} t'$.*

Sketch of Proof. Consider t as a term of Church's λ-calculus and let u be a normal form of t, which is derived from t by outside–in reductions. Any β-reduction during this sequence has a corresponding β'-reduction in a sequence starting also from t. Furthermore, after each β-reduction in the first sequence and β'-reduction in the second sequence both corresponding terms have the same abstract syntax. Hence, the term corresponding to u is in normal-form and was reached from t by β'-reductions.

This concludes our treatment of the reduction calculus.

CONCLUDING REMARKS

The aim of functional programming is to design a clean system which includes transparent computer architecture, for a neatly defined language, and a profound metatheory to support program-verification. The BRL is a suggestion for such a language, where a transparent computer architecture already exists and this paper makes the full theory of λ-calculus available for a program verification system.

REFERENCES

BACKUS, J. (1978), Can programming be liberated from the von Neumann style? *Comm. ACM* **21**, 613–641.

BERKLING, K. J. (1976a), "A Symmetric Complement to the Lambda-Calculus," Interner Bericht, ISF–76–7, GMD, D–5205, St. Augustin–1.

BERKLING, K. J. (1976b), Reduction Languages for Reduction Machines, Interner Bericht, ISF–76–8, GMD, D–5205, St. Augustin–1.

CHURCH, A. (1941), "The Calculi of Lambda-Conversion," Princeton Univ. Press, Princeton, N.J.

DE BRUIJN, N. G. (1972), Lambda-calculus notation with nameless dummies. A tool for automatic formula manipulation with application to the Church–Rosser theorem, *Indag. Math.* **34**, 381–392.

DE BRUIJN, N. G. (1980), A Survey of the Project AUTOMATH, *in* "To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism" (J. P. Seldin and J. R. Hindley, Eds.), pp. 579–606, Academic Press, London/New York.

EICK, A., AND FEHR, E. (1983), Inconsistencies of Pure LISP, *in* "Theoretical Computer Science, 6th GI-Conference" (A. B. Cremers and H. P. Kriegel, Eds.), pp. 101–110, Lecture Notes in Computer Science, No. 145, Springer-Verlag, Berlin.

GORDON, M. (1975), "Operational Reasoning and Denotational Semantics," Memo AIM–264, Stanford Artificial Intelligence Laboratory, Stanford.

HOMMES, F. (1977), "The Internal Structure of the Reduction Machine," Interner Bericht, ISF–77–3, GMD, D–5205, St. Augustin–1.

HOMMES, F., AND SCHLÜTTER, H. (1979), "Reduction Machine System," User's Guide, GMD–ISF, D–5205, St. Augustin–1.

KLUGE, W. E. (1979), "The Architecture of a Reduction Language Machine Hardware Model," Interner Bericht, ISF–79–3, GMD, St. Augustin–1.

LANDIN, P. J. (1964), The mechanical evaluation of expressions, *Comput. J.* **6**, 308–320.

MCCARTHY, J., *et al.* (1965), "LISP 1.5 Programmer's Manual," MIT Press, Cambridge, Mass.

MCGOWAN, C. L. (1970), "The Correctness of a Modified SECD-machine," *in* "Second ACM-Symposium on Theory of Computing," pp. 149–157, Assoc. Comput. Mach., New York.

MAGÓ, G. (1979), A network of microprocessors to execute reduction languages, *Internat. J. Comput. Inform. Sci.* **8**, 349–385.

PLOTKIN, G. D. (1977), LCF considered as a programming language, *Theoret. Comput. Sci.* **5**, 223–255.

SCOTT, D. (1972), Continuous lattices, *in* "Proceedings, Dalhousie Conference," pp. 97–134, Lecture Notes in Mathematics, Springer-Verlag, New York.

SCOTT, D. (1976), Data types as lattices, *SIAM J. Comput.* **5**, 522–587.

TURNER, D. A. (1979), A new implementation technique for applicative languages, *Software-Practice and Experience* **9**, 31–49.

TURNER, D. A. (1981), The semantic elegance of applicative languages, "Functional Programming Languages and Computer Architecture" (Arvind, J. Dennis, Eds.), pp. 85–92, Assoc. Comput. Mach, New York.

WADSWORTH, C. (1971), "Semantics and Pragmatics of the Lambda-Calculus," Ph.D. thesis, Oxford Univ. Oxford.