Research Report

KSTS/RR-85/010
14 June 1985

Machine Language for the Reduction Machine Reducing Recursive Programs

by

Akira Aiba
Masakazu Nakanishi

Akira Aiba
Masakazu Nakanishi

Department of Mathematics
Faculty of Science and Technology
Keio University

Hiyoshi 3-14-1, Kohoku-ku
Yokohama, 223 Japan

Dept. of Math., Fac. of Sci. & Tech., Keio Univ.
Hiyoshi 3-14-1, Kohoku-ku, Yokohama, 223 Japan

KSTS/RR-85/010
June 14, 1985

Machine Language for the Reduction Machine Reducing Recursive Programs

by

A. Aiba and M. Nakanishi

Department of Mathematics

Faculty of Science and Technology

KEIO University

ABSTRACT: By experiments, the cost of the reduction of lambda-terms which represent applicative programs are compared to the cost of the reduction of combinatory expressions which represent the same programs. These programs belong to the restricted linear recursion schema. Lambda-terms and two kinds of combinatory expressions obtained by Turner's methods are compared, all using the normal order reduction strategy. The main result states that the method of the reduction by which sub-expressions are shared is more efficient compared to the method by which sub-expressions are not shared. The reduction by the lambda-calculus is more efficient than that by the combinatory logic when they reduce expressions representing programs belong to the restricted linear recursion schema.

1. Introduction

After the reduction machine for the lambda-calculus and that for the combinatory-logic were proposed, the relative efficiency of those machines has been discussed. In this paper, the efficiency of the lambda-calculus is compared to that of combionatory-logic when they reduce terms representing recursive programs. The complexities of terms obtained from recursive programs, especially those of combinatory expressions are affected by the number of parameters in the recursive programs.

Comparisons are made on the following grounds.

i) lambda-terms, and combinatory expressions are extended by adding constants and base-operations on constants. These extended reduction systems should be Church-Rosser. The Church-Rosser property of such kind of reduction systems are studied in [Klop-80].

ii) Terms in the extended lambda-calculus and those in the extended combinatory-logic are reduced by the normal order reduction strategy.

iii) A combinatory expression is obtained from a lambda-term by Turner's two bracket abstraction algorithms [Turner-78, 79].

In section 2, recursive programs to be measured, terms in each reduction system are defined, and measured items are presented. In section 3, the results of the measurement are presented, and they are compared in section 4.

2. Basis for the comparison

The class of recursive program schema to be measured is linear [Strong-71].

Following schema has in common B, the set of base-operation symbols; P the set of predicate symbols; X the set of argument symbols; and F the set of function symbols, one of which must be chosen to represent the particular function being presented.

Definition-1(Branched recursion equation):
A branched recursion equation is an equation of the form

$$f(\mathbf{x}) = \begin{cases} t_1 & \text{if } p_1(\mathbf{x}) \\ t_2 & \text{if not } p_1(\mathbf{x}), \text{ and } p_2(\mathbf{x}) \\ t_3 & \text{if not } p_1(\mathbf{x}), \text{ and not } p_2(\mathbf{x}), \text{ and } p_3(\mathbf{x}) \\ \vdots & \vdots \\ t_n & \text{if not } p_1(\mathbf{x}), \text{ and not } p_2(\mathbf{x}), \text{ and not } p_3(\mathbf{x}), \\ & \text{and } \ldots \text{ not } p_{n-1}(\mathbf{x}), \text{ and } p_n(\mathbf{x}) \end{cases}$$

where f is a function symbol, $p_1, \ldots, p_n$ are predicate symbols, $t_1, \ldots, t_n$ are any expressions consisting of argument symbols, base-operation symbols, predicate symbols, and function symbols. And **x** denotes a sequence of argument symbols.

A branched recursion equation schema is a system of branched recursion equations with distinct left sides. Let $\langle E, f \rangle$ be a recursive presentation, where f is a function symbol occuring on the left side of an equation in a branched recursion schema E.

Computations of $\langle E, f \rangle$ can be viewed as derivations in a

tree-grammer in which function symbols play the role of non-terminal; and argument symbols, base-operation symbols, and predicate symbols play the role of terminal symbols.

Each line of the branched equation such that

$$f(\mathbf{x}) = t_i \quad \text{if not } p_1(\mathbf{x}), \text{ and not } p_2(\mathbf{x}), \text{ and } \ldots, \text{ and not } p_{i-1}(\mathbf{x}), \text{ and } p_i(\mathbf{x})$$

is viewed as a rule as follows:

$$f(\mathbf{x}) \rightarrow t_i \quad \text{if not } p_1(\mathbf{x}), \text{ and not } p_2(\mathbf{x}), \text{ and } \ldots, \text{ not } p_{i-1}(\mathbf{x}), \text{ and } p_i(\mathbf{x}).$$

Definition-2(Linear):

A rule

$$f(\mathbf{x}) \rightarrow t_i \quad \text{if not } p_1(\mathbf{x}), \text{ and not } p_2(\mathbf{x}), \text{ and } \ldots, \text{ not } p_{i-1}(\mathbf{x}), \text{ and } p_i(\mathbf{x})$$

is linear if t_i has at most one non-terminal.

Definition-3(Linear branched recursion schema):

Branched recursion schema is linear if each line of that rule is linear.

Programs which belong to the following form of linear recursion schema is measured.

$$f(x_1, \ldots, x_n) = q(x_1, \ldots, x_n) \text{ if } p(x_1, \ldots, x_n)$$
$$h(x_1, \ldots, x_n,$$
$$f(k(x_1), \ldots, k(x_n)))$$
$$\text{if not } p(x_1, \ldots, x_n) \quad \ldots\ldots(1)$$

where p, q, h, and k are base-operation symbols.

The schema (1) can be written as the following lambda-term.

$$f = \lambda x_1 . \lambda x_2 . \ldots \lambda x_n . (\mathrm{cond}\ (p\ x_1 \ldots x_n)\ (q\ x_1 \ldots x_n)\ (h\ x_1 \ldots x_n)\ (f\ (k\ x_1) \ldots (k\ x_n)))) \ldots\ldots\ldots (2)$$

where p, q, h, k, and cond are base-operation symbols.

There are following alternatives of the reduction machine on which recursive programs in the form of (1) are measured.

i) Machine code: There are two kinds of machine codes for the machine. One consists of lambda-terms, and the other consists of combinatory expressions. Of course, both reduction systems should be extended by introducing constants and base-operation symbols which are described later.

ii) Sharing mechanism: On both lambda-terms and combinatory expressions, there is a choice: One is to share sub-expressions and the other is to not shared them. In the lambda-calculus, sub-expressions which are substituted to occurences of the bound variable can be shared. Of course, sometimes an extra procedure called copy operation should be done [Wadsworth-71]. In the combinatory logic, the plural occurences of sub-expressions in reduction rules, such as the third argument of the combinator **S**, can be shared as in Turner's reduction machine [Turner-79].

There is further alternative to the reduction machine for combinators. The alternative concerns combinators which are used to represent the machine code. One uses combinators **S**, **K**, **I**, **B**, and **C**, and the other uses combinators **S**, **K**, **I**, **B**, **C**, **S'**, **B'**, and

C'. By the above alternatives, there are two versions of reduction machines for lambda-terms, and there are four versions of reduction machines for combinators. The versions of reduction machines on which efficiencies are measured are as follows.

1) **Type-I reduction machine**: Reduction machine for the lambda-calculus with no sharing mechanism,
2) **Type-II reduction machine**: One for lambda-terms which has sharing mechanism,
3) **Type-III reduction machine**: one for combinators **S**, **K**, **I**, **B**, and **C**, which has no sharing mechanism,
4) **Type-IV reduction machine**: one for combinators **S**, **K**, **I**, **B**, and **C**, which has the sharing mechanism on the reduction of the combinator **S**, and
5) **Type-V reduction machine**: one for combinators **S**, **K**, **I**, **B**, **C**, **S'**, **B'**, and **C'** which has the sharing mechanism on the reduction of combinators **S** and **S'**.

We are mainly concerned with the comparison of the relative efficiency of reduction machines for lambda-terms and combinators, which corresponds to the comparison of type-I or type-II, with type-III, IV, or type-V.

To mentain the Church-Rosser property, extra facilities for lambda-terms and combinators should be restricted as described in [Klop-80].

In the experiments for the linear recursion in the form of

(1), the followings are used.

1) $p(x_1, \ldots, x_n)$ as $onep(x_1, \ldots, x_n)$ which returns **true** if all $x_1, \ldots, x_n$ are equal to 1.

2) $q(x_1, \ldots, x_n)$ as $plus(x_1, \ldots, x_n, 1)$ which returns the summation of all arguments $x_1, \ldots, x_n$, and 1.

3) $h(x_1, \ldots, x_n, y)$ as $plus(x_1, \ldots, x_n, y)$.

4) $k(x_i)$ as $pred(x_i)$ which is the predecessor function.

"cond" is introduced into the reduction systems to represent the conditional branching scheme. It should always have three arguments.

Consider the linear recursion in the form of (1) with two arguments as an example.

$$f(x_1, x_2) = \begin{cases} plus(x_1, x_2, 1) & \text{if } onep(x_1, x_2) \\ plus(x_1, x_2, f(pred(x_1), pred(x_2))) & \text{if not } onep(x_1, x_2) \end{cases} \quad \ldots\ldots(2)$$

(2) is translated into the following lambda-term.

```
f = λ x1. λ x2.(cond (onep x1 x2) (plus x1 x2 1)
                    (plus x1 x2 (f (pred x1) (pred x2))))
                                                  ..... (3)
```

This lambda-term (3) is translated into the following expression of combinators by Turner's method [Turner-79].

```
f = S(B S(S (B S(B(B cond) onep)) (C(B C plus) 1)))
      (S(B S plus) (C(B B(B f pred)) pred)) ......(4)
```

Where as "another bracket abstraction algorithm" [Turner-78] translates (3) into the following expression of combinators.

```
f = (S' S(S' S(B' B cond onep) (C' C plus 1))
        (S' S plus (C' B(B f pred) pred))) ......(5)
```

Measurements are made on the following items.

i) Calling f: The number of the times of extracting the expression from a variable as its name. That is, the number of times of the reduction of f.

ii) Base-operations: The number of times of base-operation symbols "onep", "plus", and "pred" are applied during the reduction.

iii) Conditional: The number of times of the base-operation symbol "cond" are applied during the reduction.

iv) Proper reductions: In the results for the reduction machine for lambda-terms, this shows the number of beta-reductions during the reduction. And in the reduction machine for combinators, this shows the total number of combinator reduction.

3. Results

In this section, the results of the measurement are presented. In the following tables, "n" denotes the number of times of "pred" is applied before the value of the base operation "onep" turns out **true**.

3.1 Reduction machine type-I

When the machine code representing the linear recursion is reduced in the reduction machine type-I, each measured items are as in the table-1. The rightmost column shows the general case.

table-1 (a): Reduction of one parameter in type-I

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	6	12	20	n**2+3n+2
Conditionals	1	2	3	4	n+1
β-reductions	1	2	3	4	n+1

table-1 (b): Reduction of two parameters in type-I

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	8	18	32	2n**2+4n+2
Conditionals	1	2	3	4	n+1
β-reductions	2	4	6	8	2n+2

table-1 (c): Reduction of three parameters in type-I

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	10	24	44	3n**2+5n+2
Conditionals	1	2	3	4	n+1
β-reductions	3	6	9	12	3n+3

table-1 (d): Reduction of four parameters in type-I

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	12	30	56	4n**2+6n+2
Conditionals	1	2	3	4	n+1
β-reductions	4	8	12	16	4n+4

3.2 Reduction machine type-II

When the machine code representing the linear recursion is reduced in the reduction machine type-II, each items are as in the table-2. The rightmost column shows the general case.

table-2 (a): Reduction of one parameter in type-II

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	5	8	11	3n+2
Conditionals	1	2	3	4	n+1
β-reductions	1	2	3	4	n+1

table-2 (b): Reduction of two parameters in type-II

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	6	10	14	4n+2
Conditionals	1	2	3	4	n+1
β-reductions	2	4	6	8	2n+2

table-2 (c): Reduction of three parameters in type-II

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	7	12	17	5n+2
Conditionals	1	2	3	4	n+1
β-reductions	3	6	9	12	3n+3

table-2 (d): Reduction of four parameters in type-II

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	8	14	20	6n+2
Conditionals	1	2	3	4	n+1
β-reductions	4	8	12	16	4n+4

3.3 Reduction machine type-III

When the machine code representing the linear recursion is reduced in the reduction machine type-III, each measured item is as in the table-3.

table-3 (a): Reduction of one parameter in type-III

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	6	12	20	n**2+3n+2
Conditionals	1	2	3	4	n+1
Combinators	4	9	14	19	5n+4
S	2	5	8	11	3n+2
B	1	3	5	7	2n+1
C	1	1	1	1	1

table-3 (b): Reduction of two parameters in type-III

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	8	18	32	2n**2+4n+2
Conditionals	1	2	3	4	n+1
Combinators	11	26	41	56	15n+11
S	4	10	16	22	6n+4
B	5	13	21	29	8n+5
C	2	3	4	5	n+2

table-3 (c): Reduction of three parameters in type-III

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	10	24	44	3n**2+5n+2
Conditionals	1	2	3	4	n+1
Combinators	21	52	83	114	31n+21
S	6	15	24	33	9n+6
B	12	31	50	69	19n+12
C	3	6	9	12	3n+3

table-3 (d): Reduction of four parameters in type-III

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	12	30	56	4n**2+6n+2
Conditionals	1	2	3	4	n+1
Combinators	34	88	142	196	54n+34
S	8	20	32	44	12n+8
B	22	58	94	130	36n+22
C	4	10	16	22	6n+4

3.4 Reduction machine type-IV

When the machine code representing the linear recursion is reduced in the reduction machine type-IV, items to be measured are as in the table-4.

table-4 (a): Reduction of one parameter in type-VI

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	5	8	11	3n+2
Conditionals	1	2	3	4	n+1
Combinators	4	9	14	19	5n+4
S	2	5	8	11	3n+2
B	1	3	5	7	2n+1
C	1	1	1	1	1

table-4 (b): Reduction of two parameters in type-VI

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	6	10	14	4n+2
Conditionals	1	2	3	4	n+1
Combinators	11	26	41	56	15n+11
S	4	10	16	22	6n+4
B	5	13	21	29	8n+5
C	2	3	4	5	n+2

table-4 (c): Reduction of three parameters in type-VI

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	7	12	17	5n+2
Conditionals	1	2	3	4	n+1
Combinators	21	52	83	114	31n+21
S	6	15	24	33	9n+6
B	12	31	50	69	19n+12
C	3	6	9	12	3n+3

3.5 Reduction machine type-V

When the machine code representing the linear recursion is reduced in the reduction machine type-V, measured items are as in the table-5.

table-5 (a): Reduction of two parameters in type-V

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	6	10	14	4n+2
Conditionals	1	2	3	4	n+1
Combinators	8	19	30	41	11n+8
S	2	5	8	11	3n+2
B	1	4	7	10	3n+1
C	1	1	1	1	1
S'	2	5	8	11	3n+2
B'	1	2	3	4	n+1
C'	1	2	3	4	n+1

table-5 (b): Reduction of three parameters in type-V

	0	1	2	3	n
Calling f	1	2	3	4	n+1
Base operations	2	7	12	17	5n+2
Conditionals	1	2	3	4	n+1
Combinators	12	28	44	60	16n+12
S	2	5	8	11	3n+2
B	1	5	9	13	4n+1
C	1	1	1	1	1
S'	4	10	16	22	6n+4
B'	2	4	6	8	2n+2
C'	2	4	6	8	2n+2

4. Conclusion

In this paper we presented the relative efficiency of the reduction machine for the lambda-terms, combinators **S**, **K**, **I**, **B**, and **C**, and combinators **S**, **K**, **I**, **B**, **C**, **S'**, **B'**, and **C'**, when they represent a kind of the linear recursion. The result of the experiments can be summarized as follows.

4.1 General case

In the following, "n" denotes the depth of the recursion, which is the same as in the previous section, and "m" denotes the number of parameters.

table-6: Summaries of experiments

	type-I	type-II	type-III	type-VI	type-V
Calling f	n+1	n+1	n+1	n+1	n+1
Base Operations	m·n**2 +(m+2)n+2	(m+2)n+2	m*2**2 +(m+2)n+2	(m+2)n+2	(m+2)n+2
Conditionals	n+1	n+1	n+1	n+1	n+1
β-reductions	m(n+1)	m(n+1)			
Combinators			(3m**2+m+1)n +1/2(11m**2-9m+8)	(3m**2+m+1)n +1/2(3m**2+5m)	(6m-1)n +(4m-1)

By observing the table-6, it is clear that the reduction machines which share their sub-expressions, or those that do not requir the same number of steps of "calling f", "Base operations", and "conditionals", independent of their reduction systems. Thus, the difference between the reduction machine for the lambda-calculus and that for the combinatory-logic is in the number of their proper reductions, beta-reductions and reductions for combinators. However, these number can not be compared directly, because beta-reduction and combinatory reduction may need different amount of time or space to perform the reduction.

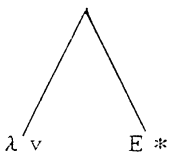
4.2 Comparison

Suppose that lambda-terms and expressions of combinators are represented as binary trees in each reduction machine as in the fig.-1, the cost of each reduction can be compared by following two cliteria.

1) The number of pointers changed within the tree representing a term to perform the reduction.

2) The number of cells consumed to perform the reduction.

(a) v → v (where v is a variable, a constant, or a base operation symbol)

(b) λ v. E → 

(where E* is the tree representation of E)

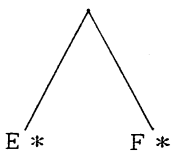
(c) (E F) →  (where E* and F* are tree representations of E and F)

Fig.-1: Tree representations

By introducing the above two cliteria, the beta-reduction can be compared to the combinator reduction.

In the reduction machine type-I, the lambda-term representing the linear recursion with m-parameters has (5m+6) symbols in its body, and the number of terminal nodes of that tree is (6m+6), and the number of non-terminal nodes of that tree is (6m+5). In the reduction machine type-II, with the lambda-term representing the linear recursion with m-parameters, four occurences of each parameter are shared in a terminal node. When subsequent m beta-reductions are performed, non-terminal nodes in the body and terminal nodes representing a parameter are copied in the following argument. Consequently, (6m+11) cells are consumed and (10m+20) pointers are changed to perform subsequent m beta-reductions.

The number of cells and the number of changed pointers to perform a reduction are dependent on the combinator which are used in the reduction. In table-7, the number of consumed cells and the number of changed pointers are listed.

By these arguments, the total number of consumed cells and the total number of changed pointers to obtain the normal form of expression which belong to the two way branched linear recursion in each reduction machine can be summarized as in the table-8.

table-7: Cells and pointers of Combinators

	Number of cells	Number of pointers
S	2	6
B	1	4
C	1	4
S '	3	8
B '	2	6
C '	2	5

table-8: Number of cells and pointers

	type-I	type-II	type-III	type-VI	type-V
Number of cells	9mn+9m +5n+5	7mn+7m +10n+10	3nm**2+7m**2 +4mn-2m+n+4	3nm**2+3/2·m**2 +4mn+9/2·m+7n	14mn+10m-6n-9
Number of pointers	10mn+10m +10n+10	10mn+10m +10n+10	12nm**2+25m**2 +10mn-13m+4n+16	12nm**2+6m**2 +10mn+14m+22n	39mn+27m-13n-11

By table-8, the reduction by which sub-expressions are shared is the efficient method to obtain the normal form than the reduction by which sub-expressions are not shared, essentially. And the reduction by the lambda-calculus is also more efficient than that by the combinatory logic when they reduce expressions representing functions which belong to the linear recursion schema.

table-9: Number of cells

	m=1	m=2	m=3	m=4
type-I	14n+14	23n+23	32n+32	41n+41
type-II	17n+17	24n+24	31n+31	38n+38
type-V	8n+1	22n+11	36n+21	50n+31

table-10: Number of pointers

	m=1	m=2	m=3	m=4
type-I	20n+20	30n+30	40n+40	50n+50
type-II	20n+20	30n+30	40n+40	50n+50
type-V	26n+16	65n+43	104n+68	143n+97

When "m" is given, expressions in the table-9 and 10 can be obtained. To conclude, the reduction in the lambda-calculus is more efficient than that in the combinatyory-logic if "m" and "n" are sufficiently large, when they reduce terms representing programs which belong to the restricted linear recursion schema.

REFERENCES

[Barendregt-81] Barendregt, H. P., 1981, The Lambda Calculus: Its Syntax and Semantics, North-Holland, Amsterdam.

[Ida-84] Ida, T., and Konagaya, A., 1984, Comparison of closure reduction and combinatory reduction schemes, preprint for LA in RIMS.

[Jones-82] Jones, S. L. P., 1982, An Investigation of the Relative Efficiencies of Combinators and Lambda Expressions, Conference Record of the 1982 ACM Symposium on Lisp and Functional Programming, pp.150-158.

[Klop-80] Klop, J. W., 1981, Combinatory Reduction System, Theses for the Ph. D., University of Utrecht.

[Strong-71] Strong Jr., H. R., 1971, Translating Recursion Equation into Flow Charts, J. Comput. Syst. Sci., 5, pp.254-285.

[Turner-78] Turner, D. A., 1978, Another Algorithm for Bracket Abstraction, J. Symbolic Logic, 44, pp.267-270.

[Turner-79] Turner, D. A., 1979, A New Implementation Technique for Applicative Languages, Softw. Pract. Exper., 9, pp.31-49.