On the Average Size of Turner's Translation to Combinator Programs

TERUO HIKITA*

Turner proposed in 1979 an interesting method of implementation for functional programs by first translating them to combinator expressions without variables, and then reducing the graphs they represent. One of the points of concern for this method was the expansion of the sizes of expressions resulting from the translation. Kennaway has recently shown that the worst case of the size of this translation is of order n^2 where n is the size of an original program. In this paper we choose a theoretical definition of an average size of the translation, and show that the order of the average is at most $n^{3/2}$. Partial results on lower bounds of the average are also shown in the case of programs with one distinct variable. Finally, numerical results for the average size are exhibited.

1. Introduction

Turner [6] showed in 1979 that use of combinators can be a neat and efficient method of implementation for functional programs. In his implementation of the applicative language SASL, first a user's program is translated, by syntax-desugaring and abstracting variables, into an expression composed of constants, predefined functions and several special kinds of combinators. Then, when supplied with input parameters, the graph which the translated expression represents is transformed (reduced) to a result value. Among these special combinators some are classical in combinatory logic [2], and the others devised by Turner, the latter playing an essential role in preventing the exponential growth of the size of the translated expression [7]. One of the advantages of this method of implementation is a coherent and efficient treatment of high-order functions. Several implementations both by software and hardware have since appeared [5].

Still, in general, the sizes of the translated expressions expand, and the analysis of this expansion has fundamental importance since the size directly relates with the time and space of the translation. Kennaway [3] has recently shown that in the worst case, the size of the translated expression is of order n^2 where n is the size of an original program. Burton [1] has shown that a certain pre-transformation of original programs can reduce the size expansion to a linear one, provided the programs do not contain global variables.

In this paper we choose a definition of average size of Turner's translation and theoretically study the average behavior of the translation. In particular we will show that the order of average size of translated expressions is at most $n^{3/2}$. We will also give some partial results on the lower bounds of the average size in the case of programs with one distinct variable. Finally we will exhibit the numerical results of the average size obtained by computer calculation.

2. Turner's Translation Scheme

We will not explain here the basic facts of λ-calculus and combinatory logic, but the reader is referred to an excellent exposition in [6] for necessary materials. We will, however, show a small example of translation of a function *pick* which selects and returns the n-th element of a given sequence s.

In Example 1, first the syntax-sugaring of program (1) is removed and a tree-like program (2) is obtained. Here, and throughout this paper, association of application is to the left, which is usual in combinatory logic; thus abc means $(ab)c$. Next the variable s is eliminated by "abstraction" in the righthand-side expression of (2) using the translation scheme explained later, obtaining program (3). Finally the variable n is abstracted in (3) and program (4) is obtained, which is the object program for graph transformation. In this example the size of the original program (2) is 12, counting the number of constant values, variables and constant functions, while that of the translated program (4) is 16, counting constants and combinators. Thus the expansion rate of this example is $16/12 \fallingdotseq 1.33$.

In general, an original expression consists of constants and variables composed by the operation of application. The translation proceeds by abstracting each variable successively from the expression, and the final result consists of constants and combinators. Although Turner [6], [7] describes the method of translation in detail for functional programs to combinator expressions, the description there does not seem to be clear enough to

Example 1 Translation through repeated abstractions.

(1) *def pick n s = if n = 1 then hd s else pick (n − 1) (tl s)*
(2) *def pick n s = cond (eq n 1) (hd s) (pick (minus n 1) (tl s))*
(3) *def pick n = S (B′ cond (eq n 1) hd) (B′ pick (minus n 1) tl)*
(4) *def pick = S′ S (C′ (B′ cond) (C eq 1) hd) (C′ (B′ pick) (C minus 1) tl)*

*Department of Mathematics, Tokyo Metropolitan University, Fukazawa, Setagaya, Tokyo 158, Japan.

Translation scheme T

1.	a	constant, combinator, variable $\neq x$	$\boldsymbol{K}a$
		variable $=x$	$\boldsymbol{I}$
2.	E	E is an application, $x \notin E$	$\boldsymbol{K}E$
3.	EF	E is not an application, or $E=E_1E_2$ and E_1 contains variables	
		$x \notin E, x=F$	E
		$x \notin E, x \in F, x \neq F$	$\boldsymbol{B}E\boldsymbol{F}^*$
		$x \in E, x \notin F$	$\boldsymbol{C}E^*F$
		$x \in E, x \in F$	$\boldsymbol{S}E^*F^*$
4.	$(EF)G$	E consists of constants and combinators	
		$x=F, x \notin G$	$\boldsymbol{C}E\boldsymbol{G}$
		$x=F, x \in G$	$\boldsymbol{S}E\boldsymbol{G}^*$
		$x \notin F, x=G$	EF
		$x \notin F, x \in G, x \neq G$	$\boldsymbol{B}'E\boldsymbol{F}G^*$
		$x \in F, x \neq F, x \notin G$	$\boldsymbol{C}'EF^*G$
		$x \in F, x \neq F, x \in G$	$\boldsymbol{S}'EF^*G^*$

Translation scheme T′

1.	a	constant, combinator, variable $\neq x$	$\boldsymbol{K}a$
		variable $=x$	$\boldsymbol{I}$
2.	E	E is an application, $x \notin E$	$\boldsymbol{K}E$
3.	EF	E is not an application, or $E=E_1E_2$ and E_1 contains constants or variables	
		$x \notin E, x \in F$	$\boldsymbol{B}E\boldsymbol{F}^*$
		$x \in E, x \notin F$	$\boldsymbol{C}E^*F$
		$x \in E, x \in F$	$\boldsymbol{S}E^*F^*$
4.	$(EF)G$	E consists of combinators	
		$x \notin F, x \in G$	$\boldsymbol{B}'E\boldsymbol{F}G^*$
		$x \in F, x \notin G$	$\boldsymbol{C}'EF^*G$
		$x \in F, x \in G$	$\boldsymbol{S}'EF^*G^*$

be stated without ambiguity. Kennaway [3] thus describes the translation scheme in a more decisive way. We here restate his scheme T in a slightly more readable form.

In this scheme, the variable of the current abstraction is denoted by x, and abstractions for subexpressions E, F and G are denoted by E^*, F^* and G^*, respectively. The definitions of the combinators used are as follows:

$\boldsymbol{K} \equiv \lambda xy.\ x$, $\boldsymbol{I} \equiv \lambda x.\ x$,
$\boldsymbol{B} \equiv \lambda xyz.\ x(yz)$, $\boldsymbol{C} \equiv \lambda xyz.\ xzy$, $\boldsymbol{S} \equiv \lambda xyz.\ xz(yz)$,
$\boldsymbol{B}' \equiv \lambda wxyz.\ wx(yz)$, $\boldsymbol{C}' \equiv \lambda wxyz.\ w(xz)y$,
$\boldsymbol{S}' \equiv \lambda wxyz.\ w(xz)(yz)$.

Note that all the cases in the scheme are exhaustive and mutually exclusive. The abstraction by a variable proceeds by recursively applying the scheme to subexpressions of an original expression.

In the scheme T the extensionality rule (η-rule)

$$\lambda x.\ Mx = M \qquad (x \text{ does not occur in } M)$$

is incorporated in several places. This rule is entirely removed in our second scheme T′, which corresponds to Kennaway's T′ and U, and this scheme will be useful in the analysis of the size as an intermediate step. Again all the cases in the scheme T′ are both exhaustive and mutually exclusive.

There is another difference between the schemes T and T′, which is found in the conditions of cases 3 and 4. This difference is not a very important one for reducing the translation size, at least when the size of an original program is small.

A final remark is that in both schemes T and T′, the combinator $\boldsymbol{K}$ appears in the expression obtained by abstraction of a variable if and only if an original expression does not contain that variable.

3. Results by Kennaway

In this section we will briefly review the results by Kennaway [3], since it seems these are not yet in the public literature and some of them will be used in the following sections. Throughout the paper we often regard an expression as a binary tree where each interior node stands for application and each leaf node is labelled by a constant, variable or combinator name. Let $s(E, x)$ denote the minimal spanning subtree in the tree representing an expression E such that the subtree shares its root with the tree and it contains all occurrences of a variable x at the leaf nodes. We denote by $|E|$ the size (length) of the expression, and by $|s(E, x)|$ the number of interior nodes of the subtree $s(E, x)$. Kennaway has shown the following key lemma for the translation T′.

Lemma 3.1. Let V be the set of variables occurring in an expression E. Then the size $t'(E)$ of the translated expression of E by abstracting the variables in V from E using the scheme T′ is

$$t'(E) = |E| + \sum_{x \in V} |s(E, x)|.$$

Using this lemma Kennaway has shown the following results. Let $\tilde{E}_{n,k}$ denote the set of expressions of size n with k distinct variables. Then, for an expression E in $\tilde{E}_{n,k}$,

$$t'(E) \leqq (k+1)n - (k^2 - k + 2)/2.$$

Next, let $t(E)$ be the size of the translated expression of an expression E by using the scheme T. Then,

$$t(E) \leqq (k+1)n - (k^2 - k + 6)/2$$

for E in $\tilde{E}_{n,k}$ and $k \geqq 3$. Finally, let $\tilde{E}_n = \bigcup_{k=1}^{n} \tilde{E}_{n,k}$ be the set of expressions of size n. Then

$$t(E) \leqq (n^2 + 3n - 6)/2,$$

for E in $\tilde{E}_n$, $n \geqq 3$. The equality in the above is achieved by an expression

$$x_1(x_2(\ldots(x_{n-1}x_n)\ldots)).$$

Lemma 3.1 tells us that in the case of the scheme T′ the order of abstractions of the distinct variables in an expression does not affect the final size of the translated expression. However, in the case of the scheme T, the order among variables crucially affects the size of the resulting expression, as will be seen in a later example.

4. An Upper Bound of Average Size

First we state the definition of average size which we have chosen. Certainly there should be ramifications of definition of average size, and a brief discussion on our definition will be in the final section.

The average of sizes of translated expressions is taken over all distinct original expressions of fixed n and k, where n is the size of the expressions and k is the number of distinct variables in the expressions. Here we require that each variable must actually appear in an expression, that is, we do not permit a variable to be implicit. The order of abstraction is fixed among the variables. We distinguish between variables but not between constants. Finally, we assume that each of these distinct expressions occurs in equal probability.

For example, when $n=2$ and $k=1$ we consider three expressions ax, xa and xx, while when $n=2$ and $k=2$ there are two: xy and yx. When $n=3$ and $k=2$ there are 24 distinct original expressions, which are shown as Example 2 together with their translations by the schemes T′ and T. Here the first abstraction is by x and then by y. There are only two cases when the two translations coincide, which are indicated by asterisks. The average sizes under T′ and T in this case are 6.50 and 3.79, respectively. (The average expansion rates are 2. 17 and 1.26, respectively.)

For fixed n there are C_{n-1} distinct rooted binary ordered trees with n leaves, where $C_{n-1}=\binom{2n-2}{n-1}/n$ is the $(n-1)$-th Catalan number. There are

$$A(n,k)=\sum_{\substack{r_1+\cdots+r_k\leqq n\\ r_1,\cdots,r_k\geqq 1}}\binom{n}{r_1}\binom{n-r_1}{r_2}\cdots\binom{n-r_1-\cdots-r_{k-1}}{r_k}$$

assignments of k distinct variables in the n leaves of each tree. Thus there are in total $C_{n-1}\times A(n,k)$ distinct original expressions of size n with k distinct variables.

Example 2 Translations by the schemes T′ and T ($n=3$ and $k=2$).

	T′	T	
x(ya)	B'CI(CIa)	B'CI(CIa)	*
x(ay)	B'CI(BaI)	B'CIa	
x(yy)	B'CI(SII)	B'CI(SII)	*
y(xa)	C'BI(CIa)	CB(CIa)	
a(xy)	B'Ba(B'CII)	B'Ba(CI)	
y(xy)	S'BI(B'CII)	SB(CI)	
y(ax)	C'BI(BaI)	CBa	
a(yx)	B'Ba(C'BII)	Ba	
y(yx)	S'BI(C'BII)	SBI	
x(xy)	B'SI(B'CII)	B'SI(CI)	
x(yx)	B'SI(C'BII)	SI	
y(xx)	C'BI(SII)	CB(SII)	
xya	C'C(B'CII)a	C'C(CI)a	
xay	B'C(CIa)I	C(CIa)	
xyy	S'C(B'CII)I	S'C(CI)I	
yxa	C'C(C'BII)a	CCa	
axy	B'C(BaI)I	Ca	
yxy	S'C(C'BII)I	SCI	
yax	C'B(CIa)I	CIa	
ayx	C'B(BaI)I	a	
yyx	C'B(SII)I	SII	
xxy	B'C(SII)I	C(SII)	
xyx	C'S(B'CII)I	C'S(CI)I	
yxx	C'S(C'BII)I	CSI	

Let $\bar{t}'(n,k)$ and $\bar{t}(n,k)$ denote the average sizes of the translated expressions of all distinct original expressions in $\tilde{E}_{n,k}$ under the schemes T′ and T, respectively. First we need a definition. The *external path length* of a tree is defined to be the sum, taken over all leaf nodes, of the lengths of the paths from the root to each leaf node. The following lemma is known (see e.g. [4], Section 2.3.4.5).

Lemma 4.1. The average of external path lengths over all binary trees with n leaf nodes is asymptotically $n\sqrt{\pi n}$.

Proposition 4.2. We have $\bar{t}'(n,n)\sim n\sqrt{\pi n}$.

Proof: When $k=n$, original expressions consist entirely of variables and each variable occurs just once. By Lemma 3.1 the size of the translated expression by the scheme T′ in this case is equal to the external path length of the tree which an original expression represents. (The size does not depend on the assignment of variables in the leaves of the tree.) Thus the proposition follows from Lemma 4.1.

Theorem 4.3. The order of the average size $\bar{t}'(n,k)$ of the translation T′ is bounded by $n^{3/2}$.

Proof: This follows immediately from Proposition 4.2 and an easy fact that

$$\bar{t}'(n,k)\leqq\bar{t}'(n,n)$$

for each $1\leqq k\leqq n$.

Theorem 4.4. The order of the average size $\bar{t}(n,k)$ of the translation T is bounded by $n^{3/2}$.

Proof: This follows immediately from Theorem 4.3 and the obvious fact that $\bar{t}(n,k)$ is always less than or equal to $\bar{t}'(n,k)$.

5. Lower Bounds of Average Size

In this section some partial results on the lower bounds of average sizes will be given in the case of original expressions with only one distinct variable.

Proposition 5.1. We have $\bar{t}'(n,1)\geqq\frac{3}{2}n$ if $n\geqq 2$.

Proof: There are in total $(2^n-1)\times\binom{2n-2}{n-1}/n$ distinct original expressions in $\tilde{E}_{n,1}$. The following argument is independent of the shape of a tree which represents an expression. Let r be the number of occurrences of the (unique) variable in the expression, thus $1\leqq r\leqq n$. When $1\leqq r\leqq n-1$, from Lemma 3.1 we can easily see that the size of the translated expression is at least $n+r$. When $r=n$ the size is always $2n-1$. Thus, averaging the translated sizes over all 2^n-1 expressions which are represented by some tree, we have

$$\begin{aligned}\bar{l}'(n,1) &\geqq \frac{1}{2^n-1}\left\{(n+1)\binom{n}{1}+(n+2)\binom{n}{2}+\cdots\right.\\ &\quad \left.+(2n-1)\binom{n}{n-1}+(2n-1)\right\}\\ &= n+\frac{1}{2^n-1}(n\cdot 2^{n-1}-1)\\ &\geqq \frac{3}{2}n.\end{aligned}$$

We now introduce our third translation scheme T* which is obtained by replacing the conditions of cases 3 and 4 in T by the following:

3. *EF* E is not an application, or $E=E_1E_2$ and E_1 contains constants or variables
4. *(EF)G* E consists of combinators

The difference between the schemes T and T* is illustrated by the following example. An expression $a(xx)x$ is translated to $S'a(SII)I$ by T, and to $S(Ba(SII))I$ by T* (and T'). The difference between T' and T* is exactly the extensionality rule. Thus we immediately see that the "power" of T* is just between T' and T; we always have

$$\bar{l}(n,k)\leqq \bar{l}^*(n,k)\leqq \bar{l}'(n,k)$$

where $\bar{l}^*(n,k)$ is the average size of translated expressions under T*.

Now we give a lower bound for $\bar{l}^*(n,1)$.

Lemma 5.2. We have $\frac{1}{3}\{\bar{l}'(n,1)-n\}\leqq \bar{l}^*(n,1)-n$.

Proof: Fix a tree with n leaf nodes. Then, by changing the assignment of occurrences of the (unique) variable at the leaf nodes, the tree can represent $A(n,1)=2^n-1$ distinct expressions with one variable. There are $n-1$ interior nodes in the tree. We divide these interior nodes into the following four classes according to whether each of its two son nodes is a leaf node (constant/variable) or an interior node (application), and we compare in each class the increases of the size at the node under the translation schemes T' and T*. In the following, a and b denote a constant or a variable, E and F an application, and x the (unique) variable.

class (1) *ab.* (Both sons are constant/variable)

This class is further divided into four subclasses according to whether each of a and b is constant or variable. The increases of the sizes under T' and T*, and the probability p that each subcase occurs in the class, are summarized as follows.

	$x\neq a, x\neq b$	$x\neq a, x=b$	$x=a, x\neq b$	$x=a, x=b$
T'	0	+1	+1	+1
T*	0	−1	+1	+1
p	$\frac{2^{n-2}-1}{2^n-1}$	$\frac{2^{n-2}}{2^n-1}$	$\frac{2^{n-2}}{2^n-1}$	$\frac{2^{n-2}}{2^n-1}$

class (2) *Eb.* (Only right son is constant/variable)

The situation of subcases is summarized as follows.

	$x\notin E, x\neq b$	$x\notin E, x=b$	$x\in E, x\neq b$	$x\in E, x=b$
T'	0	+1	+1	+1
T*	0	−1	+1	+1
p	$<\frac{2^{n-2}-1}{2^n-1}$	$<\frac{2^{n-2}}{2^n-1}$	$>\frac{2^{n-2}}{2^n-1}$	$>\frac{2^{n-2}}{2^n-1}$

class (3) *aF.* (Only left son is constant/variable)

The increase of size at the node is 0 when $x\neq a$ and $x\notin F$, and +1 otherwise. The same under T' and T*.

class (4) *EF.* (Both sons are applications)

The situation is similar to class (3).

Now, let α_i and β_i be the values of expectation of size increase at a node in class (i), $1\leqq i\leqq 4$, by the schemes T' and T*, respectively. By the above observations we have the following:

$$\alpha_1=3\beta_1, \tag{1}$$

$$\alpha_2<3\beta_2, \tag{2}$$

$$\alpha_3=\beta_3, \tag{3}$$

$$\alpha_4=\beta_4. \tag{4}$$

Let m_i, $1\leqq i\leqq 4$, be the number of interior nodes in class (i) of the fixed tree. Then the total increases averaged over 2^n-1 expressions are $\sum m_i\alpha_i$ under T' and $\sum m_i\beta_i$ under T*. Now by virtue of (in)equalities (1)–(4) we obtain the lemma.

Proposition 5.3. We have $\bar{l}^*(n,1)\geqq\frac{7}{6}n$ if $n\geqq 2$.

Proof: This follows immediately from Proposition 5.1 and Lemma 5.2.

For expressions of length $\leqq 3$ there is no difference in the translated expressions by the schemes T and T*. An observation from numerical estimations by computer is that the difference between $\bar{l}(n,k)$ and $\bar{l}^*(n,k)$ is marginal, at least when n is small. But presently we do not have a lower bound for $\bar{l}(n,k)$, even for $k=1$. We should also note that the lower bounds of Propositions 5.1 and 5.3 are not very tight.

6. Numerical Results

In this section we exhibit numerical results obtained by computer calculation. Table 1 shows the average size of

Table 1 Average translation size by the scheme T' for programs of size n with k variables.

k \ n	1	2	3	4	5	6
1	1.00	2.33	4.14	5.83	7.50	9.17
2		1.50	3.79	6.17	8.38	10.54
3			3.00	6.17	8.96	11.61
4				5.96	9.39	12.51
5					9.70	13.31
6						14.02

Table 2 Average translation size by the scheme T for programs of size n with k variables.

k \ n	1	2	3	4	5	6
1	1.00	3.00	4.86	6.71	8.57	10.44
2		4.00	6.50	8.96	11.42	13.91
3			8.00	10.96	13.90	16.87
4				12.80	16.15	19.53
5					18.24	21.98
6						24.30

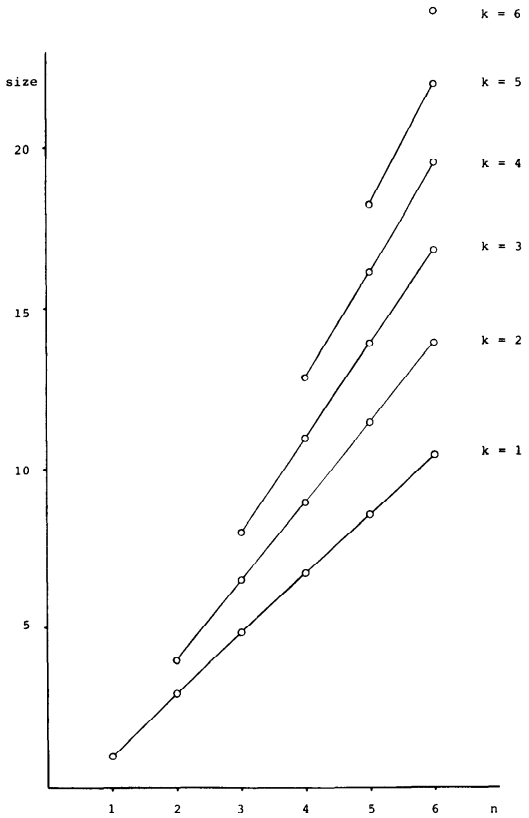

Fig. 1 Average translation size by the scheme T′ for programs of size n with k variables.

the translated expressions under the scheme T′ in the range $1 \leqq n \leqq 6$, and Table 2 is the corresponding one under the scheme T.

Fig. 1 and Fig. 2 are drawn from Table 1 and Table 2, respectively.

These tables and figures indicate the following facts.

a) When k is fixed, the average size is almost a linear function of n, both under the schemes T′ and T. That is, the average expansion rate depends only on k and not on n.

b) Some of the values of expansion rates under the scheme T are ~ 1.7 when $k=1$, and ~ 2.2 when $k=2$.

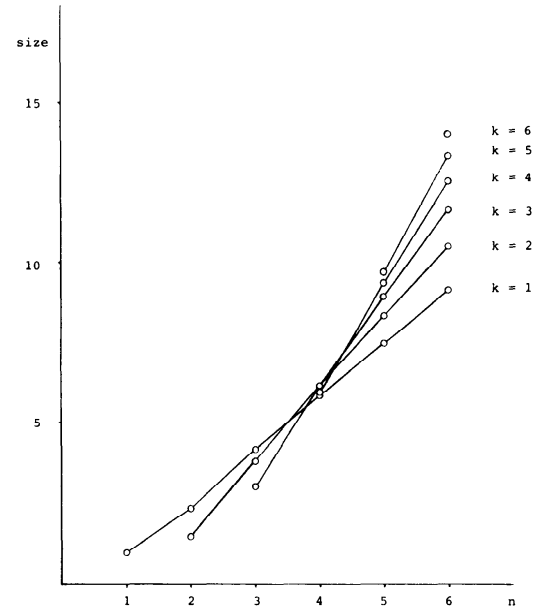

Fig. 2 Average translation size by the scheme T for programs of size n with k variables.

c) When n is fixed, the average size is a "less-than-linear" function of k.

d) Comparing the average sizes under the schemes T′ and T, we see that the extensionality rule plays an essential role in reducing the translation size, at least when n is small.

7. Concluding Remarks

In this paper an initial study has been made on the average size of the translation of functional programs to Turner combinators from both theoretical and numerical viewpoints. Main points drawn from the study are:

i) While the order of the translation size is n^2 in the worst case, it is bounded by $n^{3/2}$ in average, where n is the size of an original program.

ii) The average rate of expansion of size seems to depend only on k, the number of distinct variables appearing in a program, and not on n, at least when n is small.

iii) The extensionality rule plays an essential role in reducing the translation size, at least when n is small.

Our definition of average is of course subject to discussion. First, in our definition we discriminate between variables but not between constants. This is because there seems no reasonable way to theoretically assume a particular finite set of distinct constants. In practical situations, however, there are perhaps many distinct constants, and in such cases the "average" would be much lower than the theoretical one of the present paper. Secondly, also in practical situations, it seems

common that the number of distinct variables in a program is small, and that the number of occurrences of each variable is also small. By these reasons we may say that in practice the average translation size should be much lower than our theoretical one. Our present results, however, are of value in clarifying the theoretical aspect of the behavior of Turner's translation.

Acknowledgment

This work was partially supported by Grant-in-Aid for Scientific Research, No. 58580035, of the Ministry of Education. The author is thankful to Y. Toyama and a referee for their detailed comments on the manuscript, and he also acknowledges helpful discussions in WG5 of the Fifth Generation Computer Systems Project.

References

1. BURTON, F. W. A Linear Space Translation of Functional Programs to Turner Combinators, *Inf. Process. Lett.* **14**, (1982), 201–204.
2. CURRY, H. B., FEYS, R. and CRAIG, W. Combinatory Logic, Vol. I, North-Holland, Amsterdam, 1958.
3. KENNAWAY, J. R. The Complexity of a Translation of λ-Calculus to Combinators, School of Computing Studies and Accountancy, University of East Anglia, Norwich, 1982.
4. KNUTH, D. E. The Art of Computer Programming, Vol. I, Fundamental Algorithms, 2nd ed., Addison-Wesley, Reading, Mass., 1973.
5. PEYTON JONES, S. L. An Investigation of the Relative Efficiencies of Combinators and Lambda Expressions, *Conf. Rec. of the* 1982 *ACM Symp. on LISP and Functional Programming*, 150–158.
6. TURNER, D. A. A New Implementation Technique for Applicative Languages, *Softw. Pract. Exper.* **9**, (1979), 31–49.
7. TURNER, D. A. Another Algorithm for Bracket Abstraction, *J. Symbolic Logic* **44**, (1979), 267–270.

(Received September 19, 1983; revised February 27, 1984)