# Research report 88

## NP-COMPLETENESS OF A COMBINATOR OPTIMISATION PROBLEM

**M S Joy**
**and**
**V J Rayward-Smith***

(RR88)

**Abstract**

Using the combinators S, K, I, B, C, S', B' and C', together with a rewrite system for a graph representation of combinator expressions such that reduction of a duplicator causes the duplicated expression to be "shared", we consider expressions in that combinatory logic which reduce to normal form. To each such expression we assign a weighting which is the number of reduction steps necessary to reduce the expression to normal form. Two expressions are considered equivalent if they represent the same lambda expression. The problem of minimising the number of reduction steps over equivalent combinator expressions is proved to be NP-complete.

Department of Computer Science
University of Warwick
Coventry
CV4 7AL, UK

*School of Information Systems
University of East Anglia
Norwich
NR4 7TJ, UK

Jan 1987

# NP-Completeness of a Combinator Optimisation Problem

*M. S. Joy*

Department of Computer Science,
University of Warwick,
Coventry,
CV4 7AL,
U.K.

*V. J. Rayward-Smith,*

School of Information Systems,
University of East Anglia,
Norwich,
NR4 7TJ,
U.K.

**Abstract:**

Using the combinators S, K, I, B, C, S', B' and C', together with a rewrite system for a graph representation of combinator expressions such that reduction of a duplicator causes the duplicated expression to be "shared", we consider expressions in that combinatory logic which reduce to normal form. To each such expression we assign a weighting which is the number of reduction steps necessary to reduce the expression to normal form. Two expressions are considered equivalent if they represent the same lambda expression. The problem of minimising the number of reduction steps over equivalent combinator expressions is proved to be NP-complete.

**References**

## 1. Introduction

The uses of the lambda-calculus [1] and combinatory logic [2,3] as notations for defining functions are well known. As branches of mathematical logic they have been explored in great depth. In recent years, however, both disciplines have been used in computer science as models for the evaluation of functional programs. The lambda calculus has served as a starting point for, for instance, SECD machines [4] and combinatory logic for graph reduction machines [5,6].

There is a "natural" correspondence between a lambda expression and the function it represents, but to evaluate a function in such a form leads to complications. This is due to the use in the lambda calculus of variable names, which results in environments needing to be stored when functions are called recursively. In combinatory logic no such variables are used, so the evaluation of a function is simplified. However such a combinator expression will probably not be easy to read. It is common practice to consider a function as being initially a lambda expression, and then to apply an algorithm to the lambda expression to eliminate all the variables and introduce combinators. We assume the reader is familiar with the fundamentals of the lambda calculus and combinatory logic. A good introduction can be found in [7]. Having created such a combinator expression, it can be considered in a natural way as being a graph, and to evaluate the function it represents we can apply rewrite rules to the graph until the graph becomes the required form representing "the answer".

A combinatory logic will often be augmented by extra primitives, such as integers, in order to improve its efficiency as a computer code. In order to simplify our analysis we shall assume that *no* such extra primitives are used. If we assume a small finite set of combinators in our combinatory logic, we can think of each as corresponding to a single "machine instruction", and can thus form a measure of time for the function to evaluate as being the "number of instructions (reduction steps) executed". This metric is

naïve, but it will be sufficient for our purposes.

For simplicity in describing the result here, we shall assume that our combinatory logic is augmented by a (countable) set of variables. Variables and combinators will be considered as "atomic" expressions.

Suppose we have a function $f_C$ written as a combinator expression. We consider the size $|f_C|$ of the combinator expression to be the number of occurrences of atoms (combinators or variables) in it. Suppose $f_C$ evaluates, using "normal order" reduction, to "the answer" (that is, an expression in normal form) in $r$ reduction steps (assuming, of course, that $f_C$ is a function which evaluates in finite time!). Then the problem of minimising $r$, considered as a function of $|f_C|$, is NP-complete.

This result was proved first in [8] and was published (without proof) in [9].

## 2. Notations and Assumptions

A *combinator expression* is

(i) a *variable* v, or

(ii) a *combinator* (an element of {S,K,I,B,C,S',B',C'}), or

(iii) an *application* (L M) where L and M are combinator expressions.

By default parentheses may be omitted for clarity on the assumption of left-associativity, for example

S w (I y) z

is equivalent to

(((S w) (I y)) z).

We adopt the convention that lower-case Roman letters (with or without subscripts) denote variables unless otherwise stated. We introduce *no* extra atoms, such as numbers. The above definition of a combinatory

logic is still sufficiently rich to be equivalent to a Turing Machine, that is, for any partial recursive function there exists an expression in the combinatory logic which can be used to compute that function. We have included variables in the definition of our combinatory logic - this is for convenience only and will simplify the description later on; it avoids the need to define the lambda-calculus as well. Let *CL* denote the set of all such combinator expressions.

Our plan of attack is to restrict our attention to a subset of lambda expressions which we know will reduce to normal form in a finite time after they have been given the correct number of arguments. These are of the shape

$\lambda v_1. \ldots \lambda v_m.E$

where E contains no lambdas and, as atomic subexpressions, only elements of $\{v_1,\ldots,v_m\}$. Thus they can be thought of as simple functions which rearrange, possibly with duplications, their arguments. If $v_1 \ldots v_m$ are provided as arguments such an function with *m* lambdas *will* reduce to normal form (viz. E).

The conversion of such an expression with m lambdas to a combinator expression containing no lambdas and no variables is equivalent to a map *abstract* from CL to CL, such that, for each E in CL,

(i) *abstract* (E) contains no variables, and

(ii) (*abstract* (E) $v_1 \ldots v_m$) reduces to E.

We use the symbol "≡" to mean "lexically equal to", and the symbol "=" (as a relation between combinator expressions) to mean "are equivalent".

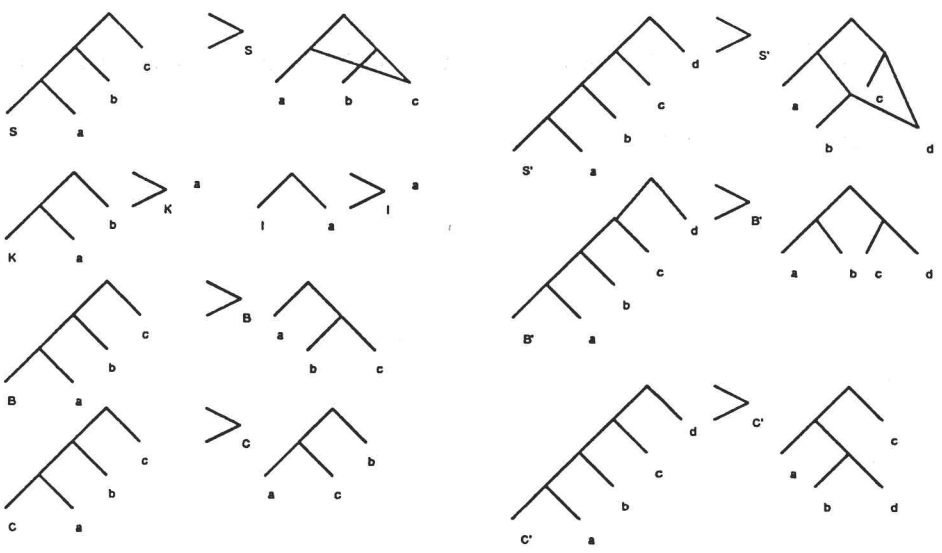
### 2.1. The Combinators Used

We use the symbol ">" to denote "reduces to", and "$>_X$" to mean "reduces in one X-reduction step to", where X is a combinator. The combinators used, originally introduced by Turner in [10], have definitions as follows (a, b, c, etc., are used here as meta-variables):

| | | |
|---|---|---|
| S a b c | $>_S$ | a c (b c) |
| K a b | $>_K$ | a |
| I a | $>_I$ | a |
| B a b c | $>_B$ | a (b c) |
| C a b c | $>_C$ | a c b |
| S' a b c d | $>_{S'}$ | a (b d) (c d) |
| B' a b c d | $>_{B'}$ | a b (c d) |
| C' a b c d | $>_{C'}$ | a (b d) c |

The graph rewrite rules are given in diagramatic form as follows, all lines are directed downwards (the arrows are omitted for clarity):

Figure 1



### 2.2. The Reduction Strategy

We assume that reduction is normal order, that is, "leftmost-outermost". This strategy minimises the number of reduction steps needed to reduce an expression to normal form (as redexes are reduced *only* if they are needed).

Initially, before any reductions are applied to an expression, that expression is stored either as a tree, or as a graph in which the only nodes with in-degree greater than 1 are atoms. This corresponds with the notion of a program being read in from a source in a way which naturally implies a simple storage mechanism (knowledge about code-sharing is itself a difficult problem). Also, in our examples, significant code-sharing would not be possible, thus we omit it altogether.

The phrase *code-sharing* will refer to nodes in a graph with in-degree greater than 1, and our result depends on the code-sharing yielded by the S and S' combinators (the *duplicators*).

### 2.3. An Almost Optimal Abstraction Algorithm

We describe an abstraction algorithm, originally due to Turner (although we phrase it somewhat differently) which produces code which in many cases is optimal. We shall prove the optimality of the algorithm for some of our expressions.

The algorithm takes the form of a map *abs* from {variables of CL} × CL → CL. For notational convenience we write $abs_x(E)$ in preference to *abs*(<x,E>), and $abs_{x,y}(E)$ as shorthand for $abs_x(abs_y(E))$.

E and F are here arbitrary combinator expressions, k is an arbitrary combinator expression which contains no variables, and "x occurs in E" is a shorthand for "there is no variable which is an atomic subexpression of E and which is lexically equal to x". The first possible of the following rules should be applied.

$abs_x(x) \equiv I$,

$abs_x(E\ x) \equiv E$, if x does not occur in E,

$abs_x(E) \equiv K\ E$, if x does not occur in E,

$abs_x(k\ x\ F) \equiv (S\ k\ abs_x(F))$, if x occurs in F,

$abs_x(k\ x\ F) \equiv (C\ k\ F)$, if x does not occur in F,

$abs_x(k\ E\ F) \equiv (S'\ k\ abs_x(E)\ abs_x(F))$, if x occurs in both E and F,

$abs_x(k\ E\ F) \equiv (C'\ k\ abs_x(E)\ F)$, if x occurs in E but not in F,

$abs_x(k\ E\ F) \equiv (B'\ k\ E\ abs_x(F))$, if x occurs in F but not in E,

$abs_x(E\ F) \equiv (S\ abs_x(E)\ abs_x(F))$, if x occurs in both E and F,

$abs_x(E\ F) \equiv (C\ abs_x(E)\ F)$, if x occurs in E but not in F,

$abs_x(E\ F) \equiv (B\ E\ abs_x(F))$, if x occurs in F but not in E.

### 2.3.1. Example

To illustrate this algorithm, consider $abs_{x,y}(y\ x\ x)$. The successive stages are as follows:

$abs_{x,y}(y\ x\ x)$

$= abs_x(abs_y(y\ x\ x))$

$= abs_x(C\ (abs_y(y\ x))\ x)$

$= abs_x(C\ (C\ (abs_y(y))\ x)\ x)$

$= abs_x(C\ (C\ I\ x)\ x)$

$= S'\ C\ (abs_x(C\ I\ x))\ (abs_x(x))$

$= S'\ C\ (C\ I)\ I.$

**2.4. Preliminary Definitions**

We give now some notations and algorithm definitions. Let x, y and v be variables, n a positive integer, and f and g combinator expressions, then

$\Psi_{0,f,g} \equiv g$,

$\Psi_{r,f,g} \equiv (f\ \Psi_{r-1,f,g})$, if $r > 0$,

$f^r \equiv \Psi_{r-1,f,f}$,

$\overline{n} = 16n$,

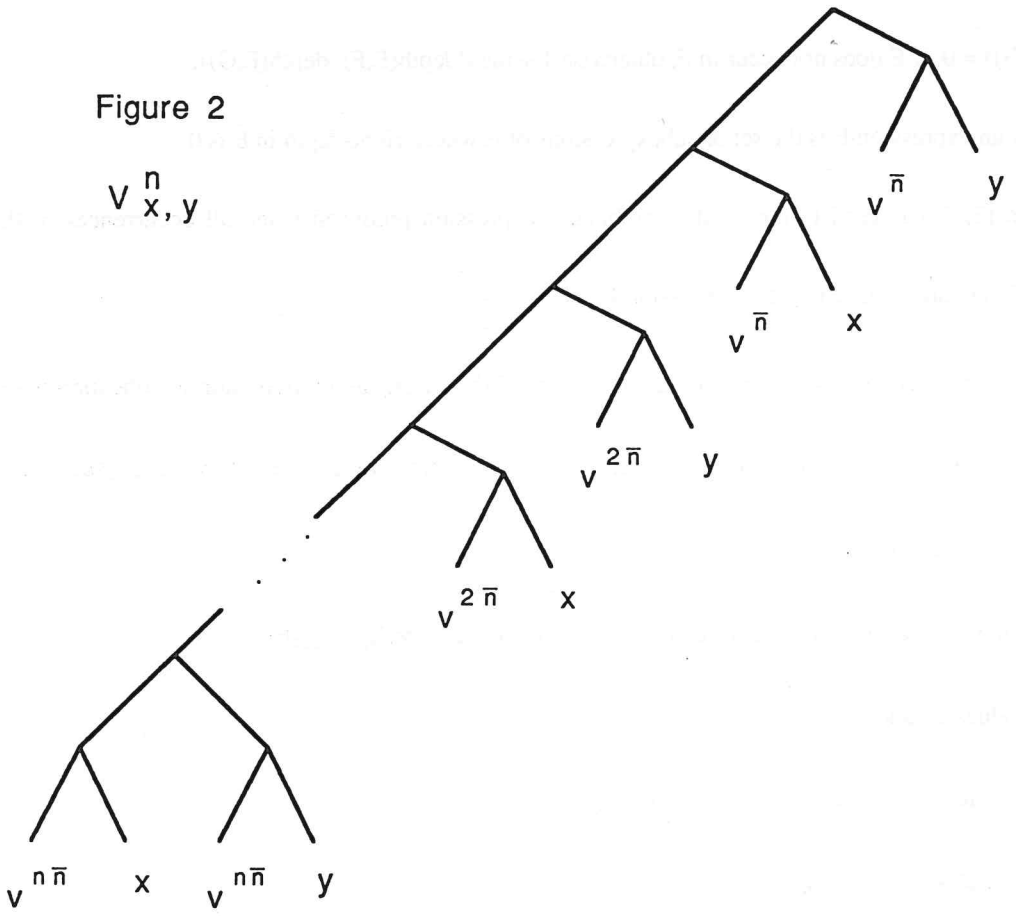
$V^n_{x,y} \equiv \overline{V}_{n,1,v,x,y}$, where

$\overline{V}_{n,m,v,x,y} \equiv \overline{V}_{n,m+1,v,x,y}\ (v^{m\overline{n}}\ x)\ (v^{m\overline{n}}\ y)$, if $n > m \geq 1$, else $\overline{V}_{n,n,v,x,y} \equiv (v^{n\overline{n}}\ x)\ (v^{n\overline{n}}\ y)$.

$W^n_{x,y} \equiv (V^n_{x,y}\ V^n_{y,x})$.

Figure 2

$V^n_{x,y}$



The *size* of a combinator expression is given by

|E| = 1, if E is an atom, else

|(F G)| = |F| + |G|.

We note that $|V^n_{y,x}| = n\bar{n}(n+1) + 2n$, which is polynomial in n.

The *left-depth* of a combinator expression is given by

left-depth(E,E) = 0;

left-depth(E,(F G)) = 0, if E does not occur in F, otherwise 1+left-depth(E,F).

We use the phrase "the left-depth of E in F" as shorthand for left-depth(E,F). *Right-depth* is defined similary, with (G F) replacing (F G) in the second clause.

The *depth* of a combinator expression is given by

depth(E,E) = 0;

depth(E,(F G)) = 0, if E does not occur in F, otherwise 1 + max(depth(E,F), depth(E,G)).

The *spine* of an expression E is the set of subexpressions of E whose right-depth in E is 0.

The notation [E/F]G is used to mean "the combinator expression produced when all occurrences of the expression F in G are replaced by the expression E".

Let F be a combinator expression in normal form containing $x_1,...,x_m$ as its only atomic subexpressions. Then $opt_{x_1,\ldots,x_m}(F)$ will be any combinator expression, not containing any element of $\{x_1,...,x_m\}$ such that

$(opt_{x_1,\ldots,x_m}(F)\ x_1 \ldots x_m)$

reduces to F in the minimum number of reduction steps, denoted by $red_{x_1,\ldots,x_m}(F)$.

We also introduce $Z_1$ and $Z_2$:

$Z_1 = [(C'S(S'C(K(KI))\ v^{n\bar{n}})v^{n\bar{n}})/(C'B\ v^{n\bar{n}}\ v^{n\bar{n}})]\ abs_{x,y}(\ V^n_{x,y})$,

$Z_2 = [(S'C(C'S(K(KI))\ v^{n\bar{n}})v^{n\bar{n}})/(B'C\ v^{n\bar{n}}\ v^{n\bar{n}})]\ abs_{x,y}(\ V^n_{y,x})$.

**2.5. Preliminary Lemmas**

We begin by giving some basic results on $V^n_{x,y}$, $W^n_{x,y}$, $Z_1$ and $Z_2$.

**LEMMA. 1.**

*$(abs_{x,y}(V^n_{x,y})\ x\ y)$ reduces to $V^n_{x,y}$ in 4n-2 reduction steps,*

*$(abs_{x,y}(V^n_{y,x})\ x\ y)$ reduces to $V^n_{y,x}$ in 4n-2 reduction steps,*

*$(Z_1\ x\ y)$ reduces to $V^n_{x,y}$ in 4n+3 reduction steps.*

*$(Z_2\ x\ y)$ reduces to $V^n_{y,x}$ in 4n+3 reduction steps.*

*$(Z_1\ x)$ and $(Z_2\ y)$ each reduces to normal form in 2n+1 reduction steps.*

*Proof.* These results are all immediate from the definitions of $V^n_{x,y}$, $Z_1$ and $Z_2$.

□

**LEMMA. 2.**

*$red_x(V^n_{x,y}) \geq 2n\text{-}1$, $red_x(V^n_{y,x}) \geq 2n\text{-}1$,*

*$red_{x,y}(V^n_{x,y}) \geq 4n\text{-}2$, $red_{x,y}(V^n_{y,x}) \geq 4n\text{-}2$.*

*Proof.* The left-depths of x and y in $V^n_{x,y}$ are 2n-1 and 2n-2 respectively, hence we get the first two inequalities, as a combinator of CL can increase the left- (or right-) depth of one of its arguments by at most 1.

Let $X_1 \equiv [w_1/v^{\bar{n}}]\ ([w_2/v^{2\bar{n}}]\ ([w_3/v^{3\bar{n}}]\ldots\ ([w_n/v^{n\bar{n}}]\ V^n_{x,y})\ldots))$,

and $X_2 \equiv [w_1/v^{\bar{n}}]\ ([w_2/v^{2\bar{n}}]\ ([w_3/v^{3\bar{n}}]\ldots\ ([w_n/v^{n\bar{n}}]\ V^n_{y,x})\ldots))$,

where the $w_i$ are distinct new variables. Thus $X_1 \equiv ((w_n\ x)(w_n\ y)\ldots\ ((w_1\ x)(w_1\ y))$, and v occurs in neither

$X_1$ nor $X_2$.

We note that $red_{x,y}(X_1) = red_{x,y}(V^n_{x,y})$, since the right-depth of $v^{i\bar{n}}$ in $v^{(i+1)\bar{n}}$ is $\bar{n}$, and thus any attempt to utilise the fact that there exist common sub-expressions of $V^n_{x,y}$ except the instances of $v^{i\bar{n}}$ in $(v^{i\bar{n}}\ x)$ and $(v^{i\bar{n}}\ y)$ for each i, will necessitate at least $(\bar{n}\text{-}1)$ extra reduction steps, which is more than the number needed by $abs_{x,y}(V^n_{x,y})$. $X_2$ is treated similarly. To *create* each sub-expression of the form $(w_i\ x)$ or $(w_i\ y)$, an A-reduction

$A\ a_1 \ldots a_r \ldots a_t >_A b_1 \ldots b_{r-1} \ldots a_{r+1} \ldots a_t\ (r \leq t)$.

where $a_r$ is either x or y, is needed. Each reduction step can increase the left-depth of either x or y (but not both) by at most 1. For, if it increased the left-depth of both by one, at least one more reduction step would be needed to "separate" them in order for them to be passed singly as arguments to the A combinators. We thus get

$red_{x,y}(X_1) \geq 4n\text{-}2$, and $red_{x,y}(X_2) \geq 4n\text{-}2$. The results for $V^n_{x,y}$ then follow.

□

**LEMMA. 3.**

$opt_{x,y}(V^n_{x,y}) \equiv abs_{x,y}(V^n_{x,y})$, $opt_{x,y}(V^n_{y,x}) \equiv abs_{x,y}(V^n_{y,x})$.

*Proof.* This follows from lemmas 1 and 2.

□

**LEMMA. 4.**

$opt_{x,y}(W^n_{x,y}) \equiv abs_{x,y}(W^n_{x,y})$, $opt_{x,y}(W^n_{y,x}) \equiv abs_{x,y}(W^n_{y,x})$.

*Proof.* Since no node in $X_1$ nor $X_2$, as defined in lemma 2, with right-depth 0 and left-depth less than 2n+1 can be shared, each reduction step in $opt_{x,y}(W^n_{x,y})$ may only affect the spine of $V^n_{x,y}$ or $V^n_{y,x}$ (but not both). So each reduction step using $opt_{x,y}(X_1\ X_2)$ can be associated with either $V^n_{x,y}$ or $V^n_{y,x}$. Thus $red_{x,y}(W^n_{x,y}) \geq red_{x,y}(V^n_{x,y}) + red_{y,x}(V^n_{x,y})$. The result then follows from lemmas 2 and 3.

□

**LEMMA. 5.**

$red_v(abs_{x,y}(V^n_{x,y})) \leq n\bar{n} + 2\bar{n} + 8n - 3$,

$red_v(abs_{x,y}(V^n_{y,x})) \leq n\bar{n} + 2\bar{n} + 8n - 3$.

*Proof.* Let $V_1 \equiv \psi_{n-1}(f,g)\bar{v}\ (v^{\bar{n}}) \equiv abs_{x,y}(V^n_{x,y})$, where

$f\ \alpha\ \beta\ \gamma > (C'S(S'C(\alpha\ \beta\ (\beta\ \gamma))\ \gamma)\ \gamma)$,

$g\ \beta\ \gamma > (C'\ B\ \gamma\ \gamma)$, and

$\bar{v} \equiv abs_h(\psi_{\bar{n}}(v,h))$.

Thus we have

$f = (C'\ (C'\ (S'\ (C'\ S)))(C'\ (C'\ (S'\ (S'\ C)))(C\ (S'\ B)\ I)\ I)\ I)$,

$g = (K\ (S\ (C'\ B)\ I))$,

$\bar{v} \equiv \psi_{\bar{n-1}}((B\ v),v) = \psi_{\bar{n-1}}(((S\ B),I)\ v)$, and

$v^{\bar{n}} = (\psi_{\bar{n-1}}((S\ I),\ I)\ v)$.

Hence,

$V_0 \equiv (S' (\psi_{n-1}(f,g)) \psi_{\bar{n}-1}((S\ B), I) (\psi_{\bar{n}-1}((S\ I), I))\ v)$

reduces to normal form $(abs_{x,y}\ (V^n_{x,y}))$.

$V_1 \equiv (S'\ F (\psi_{\bar{n}-1}((S\ B), I)) (\psi_{\bar{n}-1}((S\ I), I))\ v)$, where F is $(\psi_{n-1}(f,g))$

reduces to normal form $(abs_{x,y}\ (V^n_{x,y}))$ in at most

| | |
|---|---|
| 1 | *because of initial S'* |
| $+ 9(n-1)$ | *because of f* |
| $+ 5$ | *because of g* |
| $+ \bar{n} + (\bar{n}-1)(n-1)$ | *because of* $\psi_{\bar{n}-1}((S\ B),I)$*, since each B is used for each occurrence of f* |
| $+ 2\bar{n}-1$ | *because of* $\psi_{\bar{n}-1}((S\ I), I))$ |

$= n\bar{n} + 2\bar{n} + 8n - 3$ reduction steps.

The result for $abs_{y,x}\ (V^n_{x,y})$ is almost identical.

□

**LEMMA. 6.**

$red_v(Z_1) \leq n\bar{n} + 2\bar{n} + 8n + 6,$

$red_v(Z_2) \leq n\bar{n} + 2\bar{n} + 8n + 6.$

*Proof.* The proof is essentially the same as that for lemma 5, except that

$g\ \beta\ \gamma > (K\ (K\ I))$,

$g = (K\ (K\ (K\ (K\ I))))$,

$U_0 \equiv (S'\ (\psi_n(f,g))\ (\psi_{\bar{n}-1}((S\ B), I))\ (\psi_{\bar{n}-1}((S\ I), I))\ v)$

reduces to normal form $Z_1$;

$U_1 \equiv (S'\ F\ (\psi_{\bar{n}-1}((S\ B), I))\ (\psi_{\bar{n}-1}((S\ I), I))\ v)$, where f is $(\psi_n(f,g))$ reduced to normal form, reduces to normal form, $Z_1$, in at most

$n\bar{n} + 2\bar{n} + 8n + 6$ reduction steps. The result for $Z_2$ is almost identical, with the (C' S) and (S' C) in f interchanged.

□

We now examine $Z_1$ and $Z_2$ more closely. First of all, by using $Z_1$ instead of $abs_{x,y}(V^n_{x,y})$, and $Z_2$ instead of $abs_{x,y}(V^n_{y,x})$, we have a structure which is more "symmetric". The extra symmetry manifests itself in the following way:

$abs_{x,y}(V^n_{x,y}) = C'\ S\ (S'\ C\ (\ ...\ (C'\ B\ v^{n\bar{n}}\ v^{n\bar{n}})\ ...\ )\ v^{\bar{n}})\ v^{\bar{n}}$,

$Z_1 = C'\ S\ (S'\ C\ (\ ...\ (C'\ S\ (S'\ C(\ K\ (K\ I))\ v^{n\bar{n}})\ v^{n\bar{n}})\ ...\ )\ v^{\bar{n}})\ v^{\bar{n}}$,

so the former contains an expression $(C'\ B\ v^{n\bar{n}}\ v^{n\bar{n}})$, which corresponds to $(B'\ C\ v^{n\bar{n}}\ v^{n\bar{n}})$ in $abs_{x,y}(V^n_{y,x})$. Apart from the interchange of (C' B) and (B' C), $abs_{x,y}(V^n_{x,y})$ and $abs_{x,y}(V^n_{y,x})$ can be interconverted merely by swapping occurrences of (C' S) and (S' C). The anomaly of the (C' B) and (B' C) is not present in the $Z_i$.

Consider the proof of lemma 5. Since code which reduces to $Z_1$ can be created by swapping the occurrences of (S' C) and (C' S) in the definition of f, we may replace (C' S) and (S' C) in $U_0$ by variables

$t_1$ and $t_2$ respectively, and abstract them out. Thus f would become

(C' (C' (S' $t_1$))(C' (C' (S' $t_2$))(C (S' B) I) I) I).

After $U_0$ had then been reduced to normal form we would have

$U_0' = t_1\ (t_2\ (t_1\ (t_2 \ldots\ (t_1\ (t_2\ (K\ (K\ I))\ v^{n\bar{n}})\ v^{n\bar{n}})\ \ldots)\ v^{2\bar{n}})\ v^{2\bar{n}})\ v^{\bar{n}})\ v^{\bar{n}}$.

Abstracting $t_1$ and $t_2$ from this expression yields 8n new combinators, since

$U_0' = (U_1'\ t_1\ t_2)$ and $U_0' = (U_2'\ t_2\ t_1)$ where

$U_1' \equiv C\ (S\ C'\ (C'\ C\ (B'\ S\ I\ (\ldots))\ v^{\bar{n}}))\ v^{\bar{n}}$,

$U_2' \equiv C'\ C\ (B'\ S\ I\ (C\ (S\ C'\ (\ldots))\ v^{\bar{n}}))\ v^{\bar{n}}$,

and so an extra 16n reduction steps, as each combinator must be used twice.

**LEMMA. 7.**

$opt_{t_1,t_2}(U_0') = U_1'$, $opt_{t_2,t_1}(U_0') = U_2'$.

*Proof.* We examine the first case; the second is almost identical. As in lemma 2, we are unable to utilise the code-sharing possibilities offered by the $v^{i\bar{n}}$, and the other internal nodes of $U_0'$ cannot be shared. Due to the symmetry of $U_0'$, we are interested in code $\bar{U}$ and $\bar{U}'$ such that $(\bar{U}\ t_1\ t_2)$ reduces to $(t_1\ (t_2\ (\bar{U}'\ t_1\ t_2)\ v^{\bar{n}})\ v^{\bar{n}})$ in the minimal number of reduction steps. Each combinator A occurring in $\bar{U}$ must take as its last argument precisely one of $t_1$ or $t_2$. It is then straightforward to enumerate the possible $\bar{U}$, and the result follows.

□

*However*, we cannot simply abstract the $t_i$ from $U_0'$, since this process would alter the structure of the $Z_i$ - we would, as in lemma 6, need to consider $red_v(U_1')$ and $red_v(U_2')$.

**LEMMA. 8.**

$red_{w,v}(w\ Z_1\ Z_2) \leq n\bar{n} + 2\bar{n} + 28n + 17.$

*Proof.* Replace in $U_0$' above $t_1$ by (B C (S C')), and $t_2$ by (B (C' C) (B' S I)), thus obtaining $U_0$'', where ($U_0$'' (C' S) (S' C)) = $Z_1$ and ($U_0$'' (S' C) (C' S)) = $Z_2$. Thus we have introduced 10 combinators to create each $Z_i$ (total of 20n reduction steps). We have also $red_v(Z_1) = red_v(U_0'')$, since the structures of $Z_1$ and $U_0$'' are essentially identical. So (C' B (C (C S' (C (C I (C' S)) (S' C))) (C (C I (S' C)) (C' S))) $opt_v(U_0'')$) w v reduces to (w $Z_1$ $Z_2$) in at most $red_v(U_0'')$ + 11 + 20n reduction steps. Apply lemma 6.

□

**LEMMA. 9.**

$red_{w,v}(w\ abs_{x,y}(V^n_{x,y})\ abs_{x,y}(V^n_{y,x})) \geq red_{w,v}(w\ Z_1\ Z_2)$.

*Proof.* Clear, by symmetry.

□

**LEMMA. 10.**

$opt_v(v^{\bar{n}}) \equiv \Psi_{\bar{n}-1}((S\ I), I)$.

*Proof.* Clear, from inspection.

□

**LEMMA. 11.**

$red_v(Z_1) \geq n\bar{n} + 2\bar{n} + 8n - 10.$

*Proof.* We count the minimum number of combinators needed in $opt_v$ $(Z_1)$. We note first that it will be necessary to share certain sections of code. The occurrences of $v^{\bar{n}}$ must be shared, and by lemma 10, $red_v(v^{\bar{n}}) = 2\bar{n}\text{-}1$. Since the expressions $v^{i\bar{n}}$ must be shared there will be a function h: $v^{i\bar{n}} \rightarrow v^{(i+1)\bar{n}}$ which must be executed (n-1) times. Each execution of h must require at least $\bar{n}$-1 reduction steps, as the depth of $v^{i\bar{n}}$ in $v^{(i+1)\bar{n}}$ is $\bar{n}$. Since the right-depth of $\psi_{\bar{n}}(v,x)$ is $\bar{n}$, at least $\bar{n}$-1 reduction steps will be needed to create h initially. We are using the "simplest" method for obtaining each $v^{i\bar{n}}$. We thus need an expression $\bar{Z}$ which will take as arguments h and $v^{i\bar{n}}$, returning an expression of the form

(C' S (S' C $\bar{Z}$' $v^{i\bar{n}}$) $v^{i\bar{n}}$),

where $\bar{Z}$' is $\bar{Z}$ with arguments h and (h $v^{i\bar{n}}$). So

$\bar{Z}$ = S' (C' (C' S)) (S' (C' (S' C)) (S' B $\bar{Z}$' I) I) I.

This code is optimal. We get 9(n-1) extra reduction steps from the $\bar{Z}$, and the result follows. Note the effects at the "top" and "bottom" of $Z_1$ have been ignored, and will introduce (a few) extra combinators.

□

**LEMMA. 12.**

$red_{w,v}(w\ Z_1\ Z_2) \geq n\bar{n} + 2\bar{n} + 25n - 11.$

*Proof.* We note first of all that the only differences between $Z_1$ and $Z_2$ are the leftmost (C' S) and (S' C) expression referred to at the start of the subsection. Thus the "obvious" way to achieve the expression $opt_{w,v}(w\ Z_1\ Z_2)$ is to use a strategy similar to that outlined in lemma 8. Such a strategy involves replacing $t_1$ and $t_2$ in $U_0$' by expressions consisting only of combinators such that the resulting expression ($U_0$'', say) acts as if $t_1$ and $t_2$ had been abstracted out, yet is still of the same essential structure as $U_0$. Thus ($U_0$'' $t_1$ $t_2$) reduces to $U_0$'. If such a strategy is adopted, the replacements for $t_1$ and $t_2$ previously given are optimal. Unlike the previous lemma, it is not obvious that this reduction strategy is optimal. However, it is sufficiently close to optimal for our purposes.

There are two other possible reduction strategies. The first involves creating some (and by symmetry this implies *all*) of the $v^{\bar{n}}$, and passing them as arguments to code representing $Z_1$ and $Z_2$. This would require $O(n^2)$ extra reduction steps - so such a strategy is unacceptable.

The second involves amending the definition of f so that the number of reduction steps needed to abstract the $t_i$ is less. For instance,

f $\alpha$ $\beta$ $\gamma$ > C (S' C (C' C (B' S I ($\alpha$ $\beta$ ($\beta$ $\gamma$))) $\gamma$)) $\gamma$

would implement the optimal abstraction of $t_1$ and $t_2$ from $U_0$' given earlier. Now, suppose that we had decided on another, more efficient, abstraction of $t_1$ and $t_2$ from $U_0$'. The corresponding f will be such that f $\alpha$ $\beta$ $\gamma$ > F, where $\alpha$, $\beta$ and $\gamma$ occur in F, but the depth of $\alpha$ in F is increased by at least one, and thus abstracting $\alpha$, $\beta$ and $\gamma$ from F will yield at least one extra combinator, hence a total of n-1 extra reduction steps. The optimal number of combinators introduced to abstract $t_1$ and $t_2$ from $U_0$' is 8n, hence

$$red_{w,v}(w\ Z_1\ Z_2) \geq 2red_{t_1,t_2}(U_0') + (n\text{-}1) + red_v(Z_1)$$

$$= 2(8n) + (n\text{-}1) + (n\bar{n} + 2\bar{n} + 8n - 10)$$

(by lemma 11).

□

**LEMMA. 13.**

*Let* $s_{n,m} = \max_E |abs_{x_1,\ldots,x_m}(E)|$ *as E ranges over expressions in CL with* $|E| = n$.

*Then* $s_{n,m} < 2mn$.

*Proof.* See [9] or [11].

□

### 3. The NP-Complete Problem

The Optimisation Problem ("OP") is NP-complete; this is the specific result which we prove.

INSTANCE: A combinator expression E whose only atomic subexpressions are variables $x_1$, ..., $x_m$, and an integer k.

QUESTION: Does there exist an expression E', whose only atomic subexpressions are combinators, such that the expression (E' $x_1$ ... $x_m$) reduces, using a normal order reduction strategy, in k (or less) reduction steps to E?

### 3.1. The Vertex Cover Problem

The vertex cover problem ("VC") is NP-complete [12].

INSTANCE: Collection C of distinct subsets of a finite set S such that $c_i \in C$ satisfies $|c_i|=2$ and $S=\cup C$, a positive integer $k \leq |S|$.

QUESTION: Does there exist a subset S' of S such that

(i) $|S'| \leq k$, and

(ii) for each $c_i \in C$, $c_i \cap S' \neq \varnothing$.

### 3.2. The Optimisation Problem

Given an instance of VC, we construct an instance of OP as follows:

a combinator expression E, containing variables $v,d_1,...,d_m$ (all distinct),

$E \equiv (W^n_{c_{1,1}c_{1,2}} \ldots W^n_{c_{r,1}c_{r,2}})$, where $n = 100r^3$

an integer $k' = 30r(m+r) + 4n(r+k) + (n\bar{n} + 2\bar{n} + 28n)$.

### 3.3. Further Analysis

Let $\Gamma$ be the set of all functions from $\{1,...,r\} \rightarrow \{1,2\}$. Fix some $\phi \in \Gamma$, and let $a_i = c_{3-\phi(i)}$. Let $b_1,...,b_q$ be an enumeration of the $a_i$, and

$X_i^1 \equiv [(S\ (C\ (K\ I)\ (v^{n\bar{n}}\ c_{3-\phi(i)}))) / (B\ (v^{n\bar{n}}\ c_{3-\phi(i)})\ v^{n\bar{n}})]\ abs_{c_{\phi 0}}(\ V^n_{c_{i,1},c_{i,2}})$,

$X_i^2 \equiv [(C\ (S\ (K\ I)\ v^{n\bar{n}})\ (v^{n\bar{n}}\ c_{3-\phi(i)})) / (C\ v^{n\bar{n}}\ (v^{n\bar{n}}\ c_{3-\phi(i)}))]\ abs_{c_{\phi 0}}(\ V^n_{c_{i,2},c_{i,1}})$,

thus $(Z_j\ c_{i,\phi(i)})$ reduces to $X_i^j$.

Let $Y_1,...,Y_{2p}$ be an enumeration of the $X_i^1$ and $X_i^2$, where we note that, due to the symmetry of the $X_i^j$ there

must be an even number of $Y_i$.

Let $x_i^1$, $x_i^2$ and $y_j$ be variables which will correspond with $X_i^1$, $X_i^2$ and $Y_j$ respectively.

$\bar{E}_1 \equiv abs_{y_1,\ldots,y_{2p},b_1,\ldots,b_q}(\bar{E}_2)$,

$\bar{E}_2 \equiv ((x_1^1\ a_1)\ (x_1^2\ a_1)\ \ldots\ (x_r^1\ a_r)\ (x_r^2\ a_r))$,

$\bar{E}_3 \equiv \bar{E}_1\ Y_1\ \ldots\ Y_{2p}\ b_1\ \ldots\ b_q$.

Thus choice of $\phi(i)$ corresponds with code-sharing variables v and $c_{i,\phi(i)}$ in $W^n_{x,y}$, and p will correspond to k in VC.

$\bar{E}_3$ reduces to E in $e_1+e_x$ reduction steps, where, by lemma 13,

$e_1 < 2(2p+q)(4r) \leq 24r^2$

(since $q \leq r$ and $p \leq r$). $e_1$ is the number of combinators introduced by *abs* in $\bar{E}_1$, and, by lemma 1, $e_x = 2r(2n+2)$ is the number of reduction steps for the $X_i^1$ and $X_i^2$.

So we now have a situation where we have taken E and abstracted two variables (one of them being v) from each $W^n_{x,y}$ in E (we remember that there are three variables occurring in $W^n_{x,y}$). This has led to code-sharing; thus if, for instance, $W^n_{x,y}$ and $W^n_{z,x}$ are in E, then we *may* have chosen to code-share the occurrences of $(v^{i\bar{n}}\ x)$ in $W^n_{x,y}$ and $W^n_{z,x}$.

Now, $Y_i = (Z^i\ y_i)$, where $Z^i \in \{Z_1, Z_2\}$, $y_i$ occurs in $Y_i$, and $y_i \neq v$ $(1 \leq i \leq 2p)$.

Let $\bar{E}_4 = (abs_{z_1,z_2,d_1,\ldots,d_m}(\bar{E}_1(z^1\ y_1)\ \ldots\ (z^{2p}\ y_{2p})\ b_1 \ldots b_q))$,

where the $z^i$ are variables corresponding to the $Z^i$, and $z_1,z_2$ is the enumeration of the $z^i$ corresponding to $Z_1,Z_2$, and we note that each of $Z_1$ and $Z_2$ contains precisely *one* variable v.

$(\bar{E}_4\ Z_1\ Z_2\ d_1\ \ldots\ d_m)$ reduces to $\bar{E}_3$ in $e_4+e_y$ reduction steps, where

$e_4 < 2(m+2)(4p + q + 2) < 24mr$, by lemma 13.

$e_4$ is the number of combinators introduced by *abs* in $\bar{E}_4$,

$e_y = 2p(2n + 1)$ is the number of reduction steps for the $Z_1$ and $Z_2$.

Now, let $\bar{Z} = opt_{w,v}(w\ Z_1\ Z_2)$ and $e_z = red_{w,v}(w\ Z_1\ Z_2)$.

We have, by lemmas 8 and 12, $n\bar{n} + 2\bar{n} + 25n - 11 \leq e_z < n\bar{n} + 2\bar{n} + 28n + 17$, and

$(\bar{Z}\ \bar{E}_4\ v\ d_1 \ldots d_m)$ reduces to $\bar{E}_3$ in $e_4+e_y+e_z$ reduction steps.

We note here that, by using $Z_1$ instead of $abs_{x,y}(V^n_{x,y})$ and $Z_2$ instead of $abs_{x,y}(V^n_{y,x})$ we have introduced *at most* 12r extra reduction steps from using the optimal code for each individual $V^n_{x,y}$, and have got improved code for the abstraction of v from $V^n_{x,y}$, by lemma 9.

**LEMMA. 14.**

*There exists an expression $\bar{E}_5$ containing no variables such that (recalling that $v, d_1,\ldots,d_m$ is our enumeration of the variables occurring in E)*

*$\bar{E}_5\ v\ d_1 \ldots d_m$ reduces to E in e steps, where*

*$e < 30r(m+r) + 4n(r+p) + (n\bar{n} + 2\bar{n} + 28n)$.*

*Proof.* From the above discussion, let $\bar{E}_5 = (\bar{Z}\ \bar{E}_4)$.

$$
\begin{aligned}
e &= e_1 + e_x + e_4 + e_y + e_z \\
&< 24r^2 + 2r(2n+2) + 24mr + 2p(2n+1) + e_z \\
&< 27r(m+r) + 4n(r+p) + e_z, \text{ since } 4r+2p < 3r^2 \\
&< 30r(m+r) + 4n(r+p) + (n\bar{n} + 2\bar{n} + 28n), \text{ since } 17 < 3r^2.
\end{aligned}
$$

□

### 3.4. Résumé

We assume that n is "large" (though only polynomially so) compared to r and m. We have found an expression $\bar{E}_5$ which after suitable arguments have been added reduces to E in $4n(r+p) + e_z + O(r^2)$ reduction steps.

We associate p in this with k in VC. We next show that "optimal" code representing E reduces in approximately $4n(r+p) + e_z$ reduction steps.

We know the value of $e_z$ to within (approximately) 3n. Thus we know the "optimal" size of code, and have an algorithm for getting to within narrow bounds of such code, and certainly to sufficient accuracy to evaluate the value of k necessary to furnish a solution of VC. Thus we argue that, if we can find code representing E of size at most

$30r(m+r) + 4nr + 4nk + (n\bar{n} + 2\bar{n} + 28n)$

for E in polynomial time, we can solve VC in polynomial time also.

**LEMMA. 15.**

*$red_{v,d_1,\ldots,d_m}(E) \geq (4n-2)(r+p) + (n\bar{n} + 2\bar{n} + 25n - 11)$.*

*Proof.* Since the depth of $V^n_{x,y}$ is greater than $n\bar{n}$, optimal code to represent $V^n_{x,y}$ reduces in at least $n\bar{n}$ steps, and by lemma 5 there exists code representing E which reduces in less than $2n\bar{n}$ steps. Thus to produce optimal code for E some code-sharing will be necessary. An "obvious" strategy would be to share as many common subexpressions as possible, in particular all occurrences of $v^{i\bar{n}}$ and of $(v^{i\bar{n}}\, z)$, where $z \in \{d_1,\ldots,d_m\}$. This does not, however, yield a *strategy* for producing optimal code, since we may only assume that *most* of these subexpressions must be shared, and we have not exhibited an optimal method for generating them.

Consider $V^n_{x,y}$ and $V^n_{a,b}$, where x, y, a and b are distinct. The only shareable subexpresions are those containing only occurrences of v, that is the $v^{i\bar{n}}$. Suppose we require to find code $\bar{V}$ such that ($\bar{V}$ v x y a b) reduces to ($V^n_{x,y}$ $V^n_{a,b}$) in the minimum number of steps. Then we may assume that we share code X where (X x y) reduces to $V^n_{x,y}$. For, if we share code that allows more complicated arguments, we do *not* improve the code we produce, since we still require a similar amount of work for each $V^n_{x,y}$, of which less may be shared. By lemma 3, we may assume that $X \equiv abs_{x,y}(V^n_{x,y})$. If $a \equiv x$ then this allows us the possibility of sharing the instances of $v^{i\bar{n}}$ also.

Consider now $V^n_{x,y}$ and $V^n_{a,y}$, where x, y and a are distinct. Suppose we wish to find code $\bar{V}$ such that ($\bar{V}$ v x y a) reduces to ($V^n_{x,y}$ $V^n_{a,y}$) in the minimum number of steps. If we share code as X above, we lose the possibility of sharing expressions ($v^{i\bar{n}}$ y) containing y. However, if we have created $abs_{x,y}(V^n_{y,x})$, we *will* be able to share those expressions. By symmetry, for a non-trivial E it will be necessary to create both $abs_{x,y}(V^n_{x,y})$ and $abs_{x,y}(V^n_{y,x})$, hence we will need at least $e_z$ reduction steps to perform that creation.

At this point we note that, by lemma 4, we would not be better off treating each $W^n_{x,y}$ as a single unit rather than a combination of $V^n_{x,y}$ and $V^n_{y,x}$.

Each occurrence of $abs_{x,y}(V^n_{x,y})$ or $abs_{x,y}(V^n_{y,x})$ will, by lemma 1, require 4n-2 reduction steps, of which 2n-1 *cannot* be shared (viz. the second arguments), and 2n-1 *may* be shared (the first arguments), thus yielding 4n-2 for each $W^n_{x,y}$ (total (4n-2)r) and 4n-2 for each shared expression (total (4n-2)p). We also have, by lemma 12,

$e_z \geq n\bar{n} + 2\bar{n} + 25n - 11.$

Using $W^n_{x,y}$ ensures that, if at any point we introduce $abs_{x,y}(V^n_{x,y})$, we must also introduce $abs_{y,x}(V^n_{x,y})$, thus ensuring symmetry. Use of $Z_1$ and $Z_2$ in the previous analysis serves to iron out the asymmetry which is

introduced at the "bottom" of $V^n_{x,y}$ when applying *abs*.

□

### 3.5. Construction of an Instance of OP from an Instance of VC

We note that the map is injective, and that the size of the instance of OP is polynomial in the size of the instance of VC. We see also that $m \leq 2r$. We shall assume that r is large, for instance, $r > 100$.

**THEOREM. 1.**

*VC Transforms to OP*

*Proof.* We have, from lemmas 14 and 15,

(i) A map from an instance of VC to an instance of OP which can be evaluated in polynomial time, and which is injective,

(ii) An algorithm which will find code for an instance E of OP which reduces (after suitable arguments have been added) to E in e steps, where

$e < k' = 30r(m+r) + 4n(r+k) + (n\bar{n} + 2\bar{n} + 28n)$, and

(iii) A proof that

$red_{v,d_1,\ldots,d_m}(E) \geq (4n-2)(r+k) + (n\bar{n} + 2\bar{n} + 25n - 11)$.

The difference between these two bounds is $30r(m+r) + 2(r+k) + 3n - 11$, which is less than the change in value of either of them if k is altered by 1 (viz. 4n), since $n = 100r^3$. If we produce code which reduces in k' reduction steps, we can find a value for k which is uniquely determined, which will solve the corresponding instance of VC.

□

**THEOREM. 2.**

*The Optimisation Problem is in NP.*

*Proof.* To show this, we need only generate expressions E' "at random", with |E'|=k', such that the only atomic subexpressions of E' are combinators. For each such E' we form the expression (E $x_1 ... x_m$), and see whether it reduces to E in k' (or less) reduction steps. The steps necessary from creating the expression E' to deciding whether E' is a suitable expression can clearly be completed in polynomial time.

□

**THEOREM. 3.**

*The Optimisation Problem is NP-Complete*

*Proof.* This is a consequence of theorems 1 and 2.

□

**4. Further Observations**

If we restrict our attention to a subset of combinators, a *subbase*, and the corresponding set of functions which are representable using them, then the problem of producing optimal code *may* be simplified, as Batini in [13] shows for the subbase {B}.

However, it is reasonable to assume that the result we have given is true if we do not restrict the functions we allow, provided that we use only a finite set of combinators. Our proof is specific to one particular set of combinators (it would, for example, fail at lemmas 5 and 6 for a different set of combinators). A general proof is required.

## 5. Acknowledgements

We are grateful to Warren Burton for comments on previous versions of this document, and to the United Kingdom Science and Engineering Research Council for funding the initial research.

## 6. Bibliography

### References

1. H.P. Barendregt, *The Lambda Calculus, its Syntax and Semantics*, North-Holland, Amsterdam (1981).

2. H.B. Curry, W. Craig, and R. Feys, *Combinatory Logic, Vol. 1*, North-Holland, Amsterdam (1958).

3. H.B. Curry, J.R. Hindley, and J.P. Seldin, *Combinatory Logic, Vol. 2*, North-Holland, Amsterdam (1972).

4. H. Glaser, C. Hankin, and D. Till, *Principles of Functional Programming*, Prentice-Hall, Englewood Cliffs (1984).

5. D.A. Turner, "Another Algorithm for Bracket Abstraction," *Journal of Symbolic Logic* **44**(3) pp. 67-70 (1978).

6. D.A. Turner, "Combinator Reduction Machines," *Proceedings of the International Workshop on High Level Computer Architecture, Los Angeles*, (1984).

7. J.R. Hindley and J.P. Seldin, *Introduction to Combinators and λ-Calculus*, Cambridge University Press (1986). London Mathematical Society Student Texts 1

8. M.S. Joy, *On the Efficient Implementation of Combinators as an Object Code for Functional Programs*, University of East Anglia, Norwich (1985). PhD Thesis

9. M.S. Joy, V.J. Rayward-Smith, and F.W. Burton, "Efficient Combinator Code," *Computer Languages* **10**(3/4) pp. 211-224 (1985).

10. D.A. Turner, "A New Implementation Technique for Applicative Languages," *Software - Practice and Experience* **9** pp. 31-49 (1979).

11. J.R. Kennaway, "The Complexity of a Translation of λ-Calculus to Combinators," Internal Report CS/82/023/E, University of East Anglia, Norwich (1982).

12. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco (1979).

13. C. Batini and A. Pettorossi, "Some Properties of Subbases in Weak Combinatory Logic," Report 75-04, Istituto di Automatica, Roma (1975).