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.

This revision dated 3 August 1990.

Address for sending Proofs:

Dr. M.S. Joy, Department of Computer Science, University of Warwick, Coventry, CV4 7AL, U.K.

Abstract:

We consider a deterministic rewrite system for combinatory logic over combinators S, K, I, B, C, S’, B’ and C’. Terms will be represented by graphs so that reduction of a duplicator will cause the duplicated expression to be "shared" rather than copied. To each normalising term we assign a weighting which is the number of reduction steps necessary to reduce the expression to normal form. A lambda expression may be represented by several distinct expressions in combinatory logic, and two combinatory logic expressions are considered equivalent if they represent the same lambda expression (up to β-η-equivalence). The problem of minimising the number of reduction steps over equivalent combinator expressions (i.e. the problem of finding the "fastest running" combinator representation for a specific lambda expression) is proved to be NP-complete by reduction from the "Hitting Set" problem.

List of symbols:

α	Greek lower-case alpha
β	Greek lower-case beta
γ	Greek lower-case gamma
η	Greek lower-case eta
λ	Greek lower-case lambda
ψ	Greek lower-case psi
Γ	Greek upper-case gamma
Σ	Greek upper-case sigma
□	square (for "end of proof")
O	Capital oh
0	Numeral 0
1	Number one
l	Lower-case ell (not used as a sub/superscript)
I	Upper-case eye
é	Lower-case ee acute
ï	Lower-case eye diresis
∩	Set intersection
∪	Set union
∅	Empty set
∝	"Reduces to" symbol

1. Introduction

The uses of the lambda-calculus [1] and combinatory logic [4,5] 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 [7] and combinatory logic for graph reduction machines [14,16] .

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 recursively-defined functions are called, in order to avoid clashes of local variable names. 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 [8] . 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".

We shall consider the set {S,K,I,B,C,S’,B’,C’} of combinators, partly because it is a set in common use, partly since it has known abstraction algorithms associated with it. The results we prove will be applicable to many sets of combinators, but the details of the proof are valid only for this set.

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* written as a combinator expression. We consider the size $|f|$ of the combinator expression to be the number of occurrences of atoms (combinators or variables) in it. Suppose *f* evaluates, using "normal order" reduction, to "the answer" (that is, an expression in normal form) in *r* reduction steps (assuming, of course, that *f* is a function which evaluates in finite time!). Then the problem of minimising *r* over equivalent combinatory logic expressions of size $|f|$ is NP-complete. We prove this by reduction from the "Hitting Set" problem.

Investigation into this result was motivated by recent techniques for the implementation of functional programming languages involving the use of combinatory logic not just as a semantic domain, but with combinators implemented as primitive machine instructions [3,13] . Given a translation of a functional program to such combinator code, it is often desirable to optimise the code, and our result establishes an upper bound to the possibilities for such code improvement techniques.

This result was proved first in [9] and was published (without proof) in [10] .

2. The Optimisation Problem

The main result of this paper is that the following Optimisation Problem ("OP") is NP-complete.

OPTIMISATION PROBLEM (OP)

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’, without variables, such that the expression $(E'\ x_1\ ...\ x_m)$ reduces, using a normal order reduction strategy, in k (or less) reduction steps to E?

Thus E’ is a combinator expression equivalent to the lambda-expression $(\lambda x_1.\ ...\ \lambda x_m.\ E)$.

We will establish OP $\in$ NP, and then the NP-completeness of OP will be proved by exhibiting a polynomial transformation to OP from a known NP-complete problem. We choose to use the following problem proved to be NP-complete in [6] .

HITTING SET (HS)

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) $|S'| \le k$, and

(ii) for each $c_i \in C$, $c_i \cap S' \neq \emptyset$?

Before we can detail the transformation HS $\propto$ OP, we need to establish our notation and prove some intermediate results. We do this in sections 3 and 4. In section 5 we return to the transformation and give the necessary detail.

3. Notations and Assumptions

3.1. Combinator Expressions

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. In order to simplify our calculation later on, we do not formally define the lambda calculus. Instead we include variables in our definition of combinatory logic. Let *CL* denote the set of all such combinator expressions.

The *size* of a combinator expression is given by

|E| = 1, if E is an atom, else |(F G)| = |F| + |G|.

For instance, |S w x (I (I y))| = 6.

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 "proper combinators" 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", that is, represent the same lambda-expression. Thus, if E and F are combinator expressions such that (E $v_1 \ldots v_m$) reduces to an expression G containing only variables as atomic subexpressions, and (F $v_1 \ldots v_m$) reduces to G also, then E = F.

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 [15] , 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 diagrammatic form in figure 1 below; all lines are directed downwards (the arrows are omitted for clarity). In each rule except those for I and for K the root node of the redex is *overwritten*. For the I and K rules the pointer to the redex is redirected. An I or K reduction where the redex is the root of the whole graph is handled as a special case.

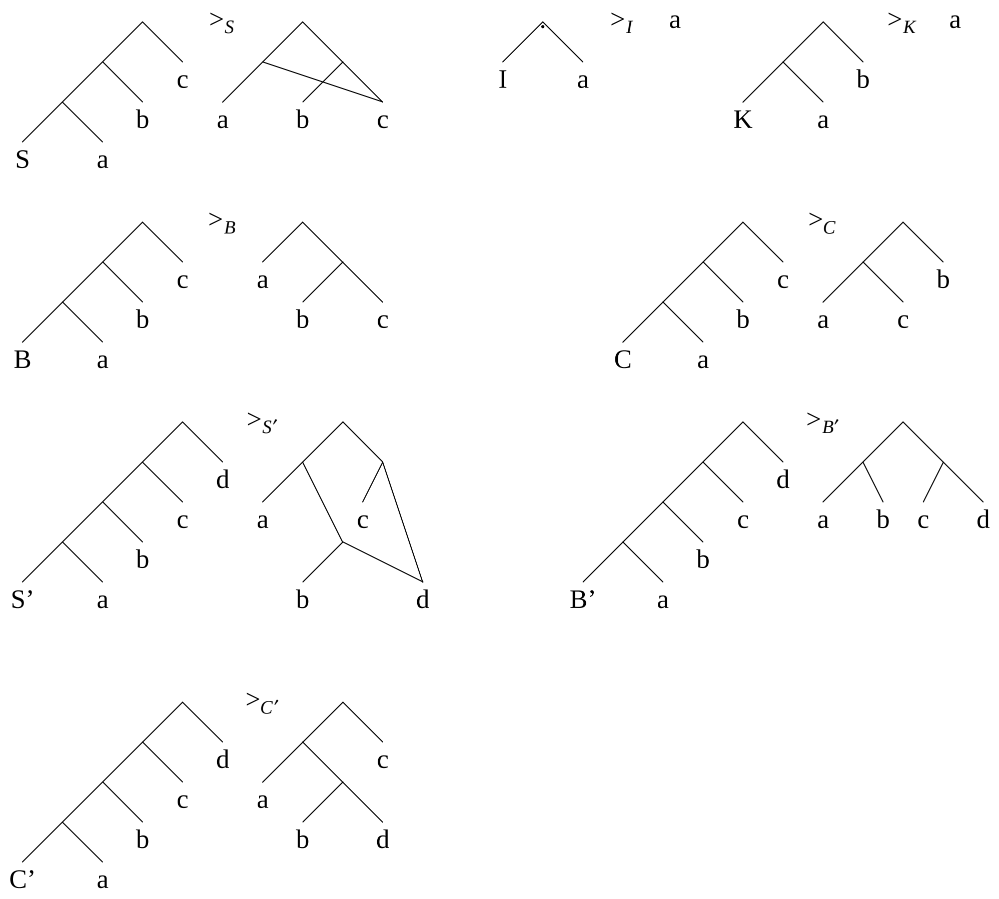

Figure 1: The Graph Rewrite Rules

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) [12] .

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 diffi cult problem).

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*). Thus

S (a b) (c (d e)) (x y z) > (a b (x y z) (c (d e) (x y z)))

will cause the subgraph which (x y z) represents to be shared after the reduction step, rather than copied.

3.2. An Almost Optimal Abstraction Algorithm

We describe an abstraction algorithm, originally due to Turner [14] (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} $\times$ CL $\rightarrow$ 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. 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.

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.

4. Intermediate Definitions and Results

The construction of the transformation HS ∝ OP relies on the use of combinator expressions of the form $W^n_{x,y}$, which we now define.

The functions ψ and $\bar{V}$ will also be used later on.

Let x, y and v be variables, n a positive integer, and f and g combinator expressions, then we define

$\psi_{0,f,g} \equiv g$, and $\psi_{r,f,g} \equiv (f\ \psi_{r-1,f,g})$ if $r > 0$. Thus $\psi_{r,f,g} \equiv f^r g$,

thus

$f^r \equiv \psi_{r-1,f,f}$.

Let $\bar{n} = 16n$, then we define, as illustrated in figure 2 below,

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

$\bar{V}_{n,m,v,x,y} \equiv \bar{V}_{n,m+1,v,x,y}\ (v^{m\bar{n}}\ x)\ (v^{m\bar{n}}\ y)$, if $n > m \geq 1$, else $\bar{V}_{n,n,v,x,y} \equiv (v^{n\bar{n}}\ x)\ (v^{n\bar{n}}\ y)$. This is illustrated in figure 2.

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

For example, left-depth(x,(a b (c x d) e f)) = 2.

We use the phrase "the left-depth of E in F" as shorthand for left-depth(E,F). *Right-depth* is defined similarly, 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 G), otherwise 1 + max(depth(E,F), depth(E,G)).

For example, depth(x,(a b (c x d) e f)) = 5.

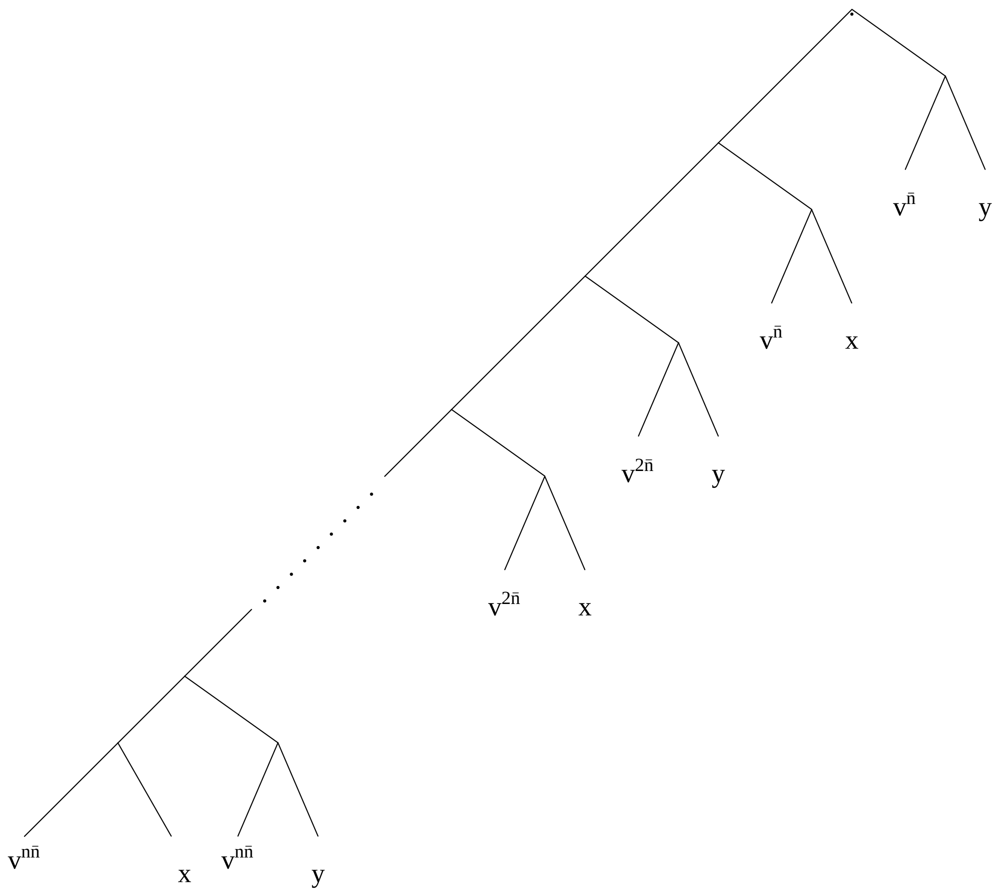

Figure 2: $V^{n}_{x,y}$

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

For example, spine(a b (c x d) e f) = {(a b (c x d) e f), (a b (c x d) e), (a b (c x d)), (a b), (a)}.

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 $opt_{x_1,...,x_m}$(F) will be any combinator expression, not containing any element of $\{x_1,...,x_m\}$ such that ($opt_{x_1,...,x_m}$(F) x_1 ... x_m) reduces to F in the minimum number of reduction steps using normal order reduction, denoted by $red_{x_1,...,x_m}$(F).

We also need to 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 $v^{n\bar{n}}$ $v^{n\bar{n}}$)] $abs_{x,y}$($V^{n}_{y,x}$).

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 defi nitions of $V^n_{x,y}$, Z_1 and Z_2.

□

LEMMA. 2.

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

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

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

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

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

where the w_i are distinct new variables. Thus $X_1 \equiv ((w_n\ x)(w_n\ y)...\ ((w_1\ x)(w_1\ y))$, 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 $(\bar{n}-1)$ 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

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

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

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

□

LEMMA. 3.

$\mathit{opt}_{x,y}(V^n_{x,y}) \equiv \mathit{abs}_{x,y}(V^n_{x,y}),\ \mathit{opt}_{x,y}(V^n_{y,x}) \equiv \mathit{abs}_{x,y}(V^n_{y,x}).$

Proof. This follows from lemmas 1 and 2.

□

LEMMA. 4.

$\mathit{opt}_{x,y}(W^n_{x,y}) \equiv \mathit{abs}_{x,y}(W^n_{x,y}),\ \mathit{opt}_{x,y}(W^n_{y,x}) \equiv \mathit{abs}_{x,y}(W^n_{y,x}).$

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 $\mathit{opt}_{x,y}(W^n_{x,y})$ may only affect the spine of $V^n_{x,y}$ or $V^n_{y,x}$ (but not both). So each reduction step using $\mathit{opt}_{x,y}(X_1\ X_2)$ can be associated with either $V^n_{x,y}$ or $V^n_{y,x}$. Thus

$\mathit{red}_{x,y}(W^n_{x,y}) \geq \mathit{red}_{x,y}(V^n_{x,y}) + \mathit{red}_{y,x}(V^n_{x,y})$.

The result then follows from lemmas 2 and 3.

□

LEMMA. 5.

$\mathit{red}_v(\mathit{abs}_{x,y}(V^n_{x,y})) \leq n\bar{n} + 2\bar{n} + 8n\ \text{-}3,$

$\mathit{red}_v(\mathit{abs}_{x,y}(V^n_{y,x})) \leq n\bar{n} + 2\bar{n} + 8n\ \text{-}3.$

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

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

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

$\bar{v} \equiv \mathit{abs}_h(\psi_{\bar{n}}(v,h))$.

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’ (ψ_{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 (ψ_{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’ (ψ_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 (ψ_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 $abs_{x,y}(V^n_{y,x})$.

Note carefully the ordering of the subscripts x and y.

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). It is not necessary also to swap occurrences of (C’ B) and (B’ C) 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 ... (t_1 (t_2 (K (K I)) $v^{n\bar{n}}$) $v^{n\bar{n}}$) ...) $v^{2\bar{n}}$) $v^{2\bar{n}}$) $v^{\bar{n}}$) $v^{\bar{n}}$.

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’ ≡ C (S C’ (C’ C (B’ S I (...)) $v^{\bar{n}}$)) $v^{\bar{n}}$,

U_2’ ≡ C’ C (B’ S I (C (S C’ (...)) $v^{\bar{n}}$)) $v^{\bar{n}}$,

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'. 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*$_v$($v^{\bar{n}}$) = $2\bar{n}$-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 $v^{i\bar{n}}$, 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 $(\alpha\,\beta\,(\beta\,\gamma))$) γ)) γ

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 α, β and γ occur in F, but the depth of α in F is increased by at least one, and thus abstracting α, β and γ 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

$$\begin{aligned} red_{w,v}(w\ Z_1\ Z_2) &\geq 2red_{t_1,t_2}(U_0') + (n-1) + red_v(Z_1) \\ &= 2(8n) + (n-1) + (n\bar{n} + 2\bar{n} + 8n - 10) \end{aligned}$$

(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] .

□

5. The Transformation

Given an instance, I, of HS, we construct an instance f(I) of OP as follows.

We assume $m = |S|$, $r = |C|$ and $c_i = \{c_{i,1}, c_{i,2}\}$, then f(I) comprises

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

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

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

Note that the f so constructed is injective, and that the size of the instance of OP is polynomial in the size of the instance of HS. We see also that $m \le r2$. We shall assume that r is large, for instance $r \ge 100$. To compute the transformation, we need to show that I is a YES-instance of HS iff f(I) is a YES-instance of OP. Before doing this we motivate our definition and establish two further lemmas.

Let Γ be the set of all functions from $\{1,...,r\} \rightarrow \{1,2\}$. Thus Γ represents the possibilities for ordering the elements of the c_i as the suffices of the W's. Fix some $\phi \in \Gamma$, and let $a_i = c_{i,3-\phi(i)}$. Let $b_1,...,b_q$ be an enumeration of the a_i, so we have not presupposed an ordering, on the a_i, and

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

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

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

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

Let x^1_i, x^2_i and y_j be variables which will correspond with X^1_i, X^2_i and Y_j respectively.

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

$\bar{E}_2 \equiv ((x^1_1\ a_1)\ (x^2_1\ a_1) \ldots (x^1_r\ a_r)\ (x^2_r\ a_r))$,

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

Thus the 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 HS.

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

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

(since $q \le r$ and $p \le r$). 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 $W^n_{x,y}$). By introducing the b_i we have ensured that no predefined ordering has been specified for the abstraction of the variables in E different to v. 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 $y_i \ne v$ $(1 \le i \le 2p)$.

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, $n\bar{n} + 2\bar{n} + 25n - 11 \le e_z < n\bar{n} + 2\bar{n} + 28n + 17$, 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 $\bar{E}_5 = (\bar{Z}\ \bar{E}_4)$.

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

□

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 HS. 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 HS. 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 HS 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 subexpressions 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*.

□

THEOREM. 1.

HS Transforms to OP

Proof. We have, from lemmas 14 and 15,

(i) A map f from an instance of HS 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 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 HS.

□

LEMMA. 16.

Suppose expression E_1, which contains only combinators, is such that (E_1 x_1 ... x_m) reduces to E_3 in p reduction steps, where E_3 contains no combinators and the x_i are distinct variables. Then there exists an expression E_2, containing only combinators, such that (E_2 x_1 ... x_m) reduces to E_3 in at most p steps, with $|E_2| \leq (|E_3| + 4p - m)^2$.

Proof. Let the combinators for the p reduction steps when (E_1 x_1 ... x_m) is reduced be (in order) c_1, ..., c_p. We can construct the expression E_2 by working "backwards" from the graph representing E_3, effectively "mimicking" the original reduction in reverse. Where necessary we insert a "dummy" symbol, which is then replaced by an expression when appropriate.

When (E_2 x_1 ... x_m) is finally constructed, each remaining dummy symbol is replaced by a single combinator, as this atom will have been "deleted" when (E_2 x_1 ... x_m) is reduced.

Since the original reduction was normal order, E_2 will be in normal form. However there may be some code-sharing in E_2, but since this expression is in normal form the number of reduction steps for (E_2 x_1 ... x_m) will be the same as if E_2 were considered as a tree with the shared subgraphs copied. This is because no shared node in E_2 will be overwritten.

At no point must we introduce more than 4 extra symbols at any one step (for example, suppose our expression after reduction step n is α; and an S' reduction is used at reduction step n, then the expression after step (n-1) would be: $(S'\ \kappa\ \beta\ \gamma\ \delta)$, i.e. symbol α has been replaced by expression $(\kappa\ (\beta\ \delta)(\gamma\ \delta))$.

We are constructing a graph, therefore the number of leaf nodes in E_2 will be at most ($|E_3|+4p-m$). Since this graph may contain shared nodes, $|E_2| \le (|E_3|+4p-m)^2$.

Note that this construction is nondeterministic; it assumes one has been able to choose which subexpression of an an intermediate expression to rewrite in order to mimic the original reduction.

□

THEOREM. 2.

The Optimisation Problem is in NP.

Proof. From lemma 16, we need only generate expressions E' nondeterministically, with $|E'| \le (|E|+4k'-m)^2$, such that the only atomic subexpressions of E' are combinators and (E' x_1 ... x_m) reduces to E in at most k' steps. We note here that we have already produced *an* expression E' which such that (E' x_1 ... x_m) reduces to E in at most k' steps (see lemma 14 above), and we may without loss of generality assume that k<k'. 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.

□

6. Final 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 simplifi ed, as Batini in [2] 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 fi nite set of combinators. Our proof is specifi c 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.

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

References

1. Barendregt, H.P., *The Lambda Calculus, its Syntax and Semantics,* North-Holland, Amsterdam, NL (1981). Studies in Logic and the Foundations of Mathematics.

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

3. Clarke, T.J.W., Gladstone, P.J.S., MacLean, C.D., and Norman, A.C., “SKIM - The S,K,I Reduction Machine” in *Conference Record of the 1980 LISP Conference, Stanford University* (1980).

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

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

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

7. Glaser, H., Hankin, C., and Till, D., *Principles of Functional Programming,* Prentice-Hall, Englewood Cliffs, NJ (1984). ISBN 0-13-709163-X (pbk).

8. Hindley, J.R. and Seldin, J.P., *Introduction to Combinators and λ-Calculus,* Cambridge University Press, Cambridge, UK (1986). London Mathematical Society Student Texts 1. ISBN 0-521-31839-4 (pbk).

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

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

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

12. Klop, J.W., *Combinatory Reduction Systems,* Mathematisch Centrum, Amsterdam, NL (1980). Mathematical Centre Tracts 127.

13. Stoye, W.R., “The Implementation of Functional Languages using Custom Hardware,” Technical Report 81, University of Cambridge Computer Laboratory, Cambridge, UK (1985).

14. Turner, D.A., “Another Algorithm for Bracket Abstraction,” *Journal of Symbolic Logic,* 44, 3, pp. 67-70 (1979).

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

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