The University of Wisconsin
Computer Sciences Department
1210 West Dayton Street
Madison, Wisconsin 53706

Received: November, 1974

AN AXIOMATIC APPROACH TO EQUIVALENCE OF STRAIGHT-LINE PROGRAMS WITH STRUCTURED VARIABLES

by

Christoph M. Hoffmann
Lawrence H. Landweber

Computer Sciences Technical Report #228

November 1974

AN AXIOMATIC APPROACH TO EQUIVALENCE OF STRAIGHT-LINE PROGRAMS WITH STRUCTURED VARIABLES*

by

Christoph M. Hoffmann**
Lawrence H. Landweber

ABSTRACT

A program scheme which models straight line code admitting structured variables such as arrays, lists, queues, etc. is considered. A set of expressions is associated with a program reflecting the input-output transformations. A basic set of axioms is given and program equivalence is defined in terms of expression equivalence. Program transformations are then defined such that two programs are equivalent if and only if one program can be transformed to the other via the transformations. An application of these results to code optimization is then discussed.

---

*This research was supported in part by NSF Grant #GJ33087 and by University of Waterloo Research Grant #126 7048 02

**Present address - Computer Science Department, University of Waterloo, Waterloo, Canada

1. Introduction

We consider a program scheme which models straight-line programs having both scalar and structured variables with the goal of finding a minimal complete set of transformations on programs preserving program equivalence. Structured variables may be understood as representing arrays, stacks, lists, or other data structures.

Theoretical work on code optimization faces the difficulty that the general problem is recursively undecidable. Investigations, therefore, must restrict the problem suitably when the ultimate aim is its applicability to present day compilers. We restrict our attention to straight-line programs, that is programs which contain no branching or control instructions. Previous work has found analogous results in the absence of structured variables [1,4]. A less general model for arithmetic expressions with particular cost functions has also been studied, for example, in [6,7,12].

The addition of structured variables was earlier investigated in [2,3]. The problems posed therein seem to be due to the approach to program equivalence (see [3], p.128-130), and can be overcome by an axiomatic approach. Other work attacking structured variables restricts attention to special cases [5,8] such as address computation of array indices in loops.

In this paper we investigate the problems arising from structured variables in straight-line code. We define program equivalence axiomatically using a minimal interpretation of the operators which reference components of structures denoted by structured variables. As in [3], with each program we associate a set of expressions describing the computations performed in terms of input values. Two programs are equivalent if they have equivalent expression sets. A minimal set of code transformations is then found which

preserves equivalence and is complete in the sense that if π_1 and π_2 are equivalent programs, then the given transformations can be used to convert π_1 to π_2. This generalizes the results of [1].

In section 5, the results are applied to code optimization. A class of cost functions is defined which extends the concepts introduced in [1] and [4] and it is shown how optimization algorithms can be structured so as to achieve their goal efficiently while retaining generality.

2. Programs and Expressions

There are two types of symbols - operators and variable names:

(1) Ω is the set of operator symbols. Each operator ϕ in Ω is r-ary for some positive integer r. The assignment operator $\leftarrow$ and the selection operator $\cdot$ are elements of Ω.

(2) K is a countable set of scalar variable names, and S is a countable set of structured variable names.

Usually capital Latin letters will denote elements of K and lower case Greek letters will denote elements of S.

Assume that V is a set of scalar values. A structured variable α can be understood as a mapping from V into V, i.e., the expression $\alpha \cdot A$ denotes a scalar value "stored at location A in α". This value will be undefined for all those scalar values of A which are not in the domain of α.

A statement is a string of symbols having one of the following three forms:

(1) $X \leftarrow \phi\, Y_1 \ldots Y_r$ (ϕ an r-ary operator)

(2) $X \leftarrow \alpha \cdot Y$

(3) $\alpha \cdot X \leftarrow Y$

A statement of type (1) represents an instruction which applies the r-ary operator ϕ to the current values of the scalar variables $Y_1, Y_2, \ldots, Y_r$. A statement of type (2) assigns the current value of $\alpha \cdot Y$ to the scalar variable X, and a statement of type (3) assigns the current value of the scalar variable Y to the component of α addressed by the current value of the scalar variable X. The left-hand side and the right-hand side refer to the strings to the left and to the right of the $\leftarrow$ operator, respectively.

A program π is a triple (I, O, P) where I and O are finite subsets of $K \cup S$ representing the input and the output variables of π, respectively. P denotes a finite sequence of statements.

Example 2.1.

Let $\pi = (I,O,P)$ where $I = \{\alpha,X,Y\}$, $O = \{\alpha,Z\}$, P =

$$
\begin{aligned}
T_1 &\leftarrow *XY \\
T_2 &\leftarrow +XY \\
T_3 &\leftarrow \alpha \cdot T_1 \\
T_4 &\leftarrow \alpha \cdot T_2 \\
Z &\leftarrow +T_3T_4 \\
\alpha \cdot T_1 &\leftarrow T_2
\end{aligned}
$$

This program assigns to Z the sum $\alpha(X*Y) + \alpha(X+Y)$ and re-assigns to $\alpha(X*Y)$ the sum X+Y, understanding α as a vector; T_1, T_2, T_3 and T_4 serve as temporary scalar variables.

A program is proper, if all scalar variables referenced are either inputs or have been previously computed, and no referenced structure component is undefined. In the following only proper programs are considered. In general, it is not possible to determine properness. If every structured variable which occurs in the program is required to be an input variable, then properness is easily checked.

Intuitively, the program scheme defined models the computation of basic blocks, that is of programs in which there are only assignment statements. Most intermediate languages into which a compiler translates before optimizing are simple interpretations of this scheme. The three statement types, in particular, focus the attention of the investigation on the problems arising from the presence of structured variables in the source language, by leaving operators other than the assignment and selection operators uninterpreted.

We define expressions which, intuitively speaking, model the computation of programs in a more concise manner than do programs themselves. Expressions over K,S and Ω can be scalar or structured and are defined as follows:

(1) A in K is a scalar expression.

(2) α in S is a structured expression.

(3) If $E_1,E_2,\ldots,E_r$ are scalar expressions and ϕ is an r-ary operator, then $\phi E_1 E_2 \ldots E_r$ is a scalar expression.

(4) If E and F are scalar expressions and G is a structured expression, then G(E,F) is a structured expression.

(5) If E is a scalar expression and G a structured expression, then G·E is a scalar expression.

(6) Nothing else is a scalar or structured expression.

Denote the set of expression by E. The definition is analogous to the definition in [2,3]. By virtue of these rules, a structured expression can be written as $\alpha(E_1,V_1)(E_2,V_2)\ldots(E_n,V_n)$ where the E_i and V_i are scalar expressions. Intuitively, the expression list reflects the assignment history of the structured variable α: The first assignment to α was done by some statement $\alpha \cdot A \leftarrow B$ where the current value of A was the value of the expression E_1 and the current value of B was the value of the expression V_1; and the most recent assignment to α was done by some statement $\alpha \cdot C \leftarrow D$ where the current value of C was the value of the expression E_n and the current value of D was the value of the expression V_n. Because all programs are proper, all scalar variables appearing in E_i and V_i will be input variables which can be thought of as having their initial values. The following will make this more precise.

Given the program $\pi = (I,O,P)$, we associate with each variable name occurring in P an expression e(X,i) which describes the value of X after the execution of the statement S_i in P as an expression of the input variables and possibly some additional structured variables. Define for each $X \in K \cup S$ occurring in P

(V1) $e(X,0) = \begin{cases} X \text{ for all } X \text{ in } I \text{ and all structured variable} \\ \text{names occurring in } P, \\ \text{undefined otherwise} \end{cases}$

(V2) if S_i in P is $X \leftarrow \phi Y_1 \ldots Y_r$, then
$e(X,i) = \phi e(Y_1,i-1)e(Y_2,i-1)\ldots e(Y_r,i-1)$

(V3) if S_i in P is $X \leftarrow \alpha \cdot Y$, then
$e(X,i) = e(\alpha,i-1) \cdot e(Y,i-1)$

(V4) if S_i in P is $\alpha \cdot X \leftarrow Y$, then
$e(\alpha,i) = e(\alpha,i-1)(e(X,i-1),e(Y,i-1))$

(V5) if X in $K \cup S$ occurs in P and is not assigned by S_i, then
$e(X,i) = e(X,i-1)$.

(V6) An expression e(X,i) is undefined if any of the expressions required by (V1) to (V5) is undefined.

Associate with each program $\pi = (I,O,P)$ the set of expressions $v(\pi) = \{e(X,n) | X \in O\}$ where n is the number of statements in P.

<u>Example 2.2</u>

$v(\pi)$ for the program π of Example 2.1 is

$\{+\alpha \cdot *XY\alpha \cdot +XY,\ \alpha(*XY,+XY)\}$.

Note that each structured subexpression in $v(\pi)$ gives the complete assignment history of some structured variable up to the point of the computation of the subexpression (see Example 2.3). Since we assume that all programs are proper, every scalar variable in $v(\pi)$ must be an input variable. In $v(\pi)$, these variables can be considered to have their initial values. Because rule (V1) allows non-input structured variables to appear in expressions, any structured variable in π can appear in $v(\pi)$.

Example 2.3.

To illustrate how expressions are obtained, consider the following program where B, A and C are input variables. Only the changes of the associated expressions are indicated.

$\alpha \cdot B \leftarrow C$

$T \leftarrow \alpha \cdot B$

$\alpha \cdot A \leftarrow T$

$X \leftarrow \alpha \cdot A$

$e(B,0) = B,\ e(C,0) = C,\ e(A,0) = A,\ e(\alpha,0) = \alpha$

$e(\alpha,1) = \alpha(B,C)$

$e(T,2) = \alpha(B,C) \cdot B$

$e(\alpha,3) = \alpha(B,C)(A,\alpha(B,C) \cdot B)$

$e(X,4) = \alpha(B,C)(A,\alpha(B,C) \cdot B) \cdot A$

The expression set associated with a given program is unique. The converse is not true, because, for example, the names of the temporary scalar variables are not reflected by the expressions in $v(\pi)$.

3. Equivalence and Code Transformations

Let E and F be expressions in E, X be in $K \cup S$. Let X and F be of the same type, i.e., if F is a scalar expression, then X is a scalar variable and if F is a structured expression, then X is a structured variable. By $E(X \leftarrow F)$ we denote the expression H (in E) obtained by substituting F for every occurrence of X in E. If X does not occur in E, then H = E.

An axiom scheme is a pair of expressions (E_1,E_2) where E_1 and E_2 are in E. Let $\mathcal{R}$ be a finite set of axiom schemes. Two expressions E and F are called equivalent under $\mathcal{R}$ (denoted by $\equiv_{\mathcal{R}}$) if one of the following is true:

(E1) $E = F$ or $(E,F) \in \mathcal{R}$ or $(F,E) \in \mathcal{R}$.

(E2) There is an expression H in E and $E \equiv_{\mathcal{R}} H$ and $H \equiv_{\mathcal{R}} F$.

(E3) There are expressions E_0,E_1,F_0,F_1 in E, and a name $X \in K \cup S$ such that X, E_1 and F_1 are all of the same type, and $E = E_0(X \leftarrow E_1)$, $F = F_0(X \leftarrow F_1)$, $E_0 \equiv_{\mathcal{R}} F_0$ and $E_1 \equiv_{\mathcal{R}} F_1$.

(E4) Let $A_1,\ldots,A_n,B_1,\ldots,B_n \in K$, $\alpha \in S$.
$E = \alpha(A_1,B_1)\ldots(A_{i-1},B_{i-1})(A_n,B_i)(A_{i+1},B_{i+1})\ldots(A_n,B_n)$ and
$F = \alpha(A_1,B_1)\ldots(A_{i-1},B_{i-1})(A_{i+1},B_{i+1})\ldots(A_n,B_n)$ where $i < n$.

(E5) Let $A_1,\ldots,A_n,B_1,\ldots,B_n \in K$, $\alpha \in S$.
$E = \alpha(A_1,B_1)(A_2,B_2)\ldots(A_i,B_i)\ldots(A_n,B_n) \cdot A_i$ and
$F = \alpha(A_2,B_2)\ldots(A_i,B_i)\ldots(A_n,B_n) \cdot A_i$ where $i > 1$.

In the following we assume that $\mathcal{R}$ contains only one axiom scheme:

$\mathcal{R}_0 = \{(\alpha(X,Y) \cdot X, Y) \mid \alpha \in S,\ X,Y \in K\}$

and consider only equivalence under $\mathcal{R}_0$ (denoted by $\equiv$). The axiom scheme in $\mathcal{R}_0$ and the rules (E4) and (E5) reflect the interpretation given to the $\cdot$ operator and structured variables: The axiom guarantees that the value of $G(E,V) \cdot F$ is equivalent to the value of V whenever $E \equiv F$. The proof is as follows.

1.	$E \equiv F$	given
2.	$G(E,V)\cdot X \equiv G(E,V)\cdot X$	(E1)
3.	$G(E,V)\cdot E \equiv G(E,V)\cdot F$	(E3),1,2
4.	$\alpha(X,Y)\cdot X \equiv Y$	AXIOM
5.	$G(E,V)\cdot E \equiv V$	(E3),4
6.	$G(E,V)\cdot F \equiv V$	(E2),3,5

Rule (E5) reflects that no value constructed prior to (A_i,B_i) in α can be the value of the A_i component of α; rule (E4) reflects a reassignment of the A_n component of α.

If H is a subexpression of E and $H \equiv H'$, then the expression E' obtained from E by substituting H' for H is equivalent to E. This is justified by (E3). Moreover, (E3) can be used to justify the use of (E4), (E5) and the axiom in a more general manner. For (E4), $E = \alpha(A_1,B_1)\ldots(A_i,B_i)\ldots(A_n,B_n)$, where $A_i \equiv A_n$; for (E5), $E = \alpha(A_1,B_1)\ldots(A_i,B_i)\ldots(A_n,B_n).R$ where $A_i \equiv R$; and for the axiom $\alpha(E,V)\cdot F \equiv V$ if $E \equiv F$. These generalizations using (E3) will be referred to as <u>substitution</u>.

In considering these rules, the reader should recall that for a proper program π, each structured subexpression in $v(\pi)$ reflects the assignment history of a structured variable up to the point of the computation of that subexpression. The history is in terms of previously defined structured variable components and input variables.

Note that it would be improper to strengthen rule (E5) by allowing $F = B_i$, for it may be the case that for a particular assignment of input values the values of A_i and some A_{i+j} are identical.

<u>Definition</u>. Two proper programs π and π' are equivalent if π and π' have identical input sets and their expression sets $v(\pi)$ and $v(\pi')$ are equivalent (i.e., to each expression there is an equivalent expression in $v(\pi')$, and conversely, to each expression in $v(\pi')$ there is a corresponding equivalent one in $v(\pi)$).

<u>Example 3.1</u>.

The program $\pi' = (I',O',P')$ where $I' = \{\alpha,X,Y\}$, $O' = \{Z,\alpha\}$, $P' =$

$T_1 \leftarrow *XY$

$T_2 \leftarrow +XY$

$T_3 \leftarrow \alpha\cdot T_1$

$T_4 \leftarrow \alpha\cdot T_2$

$Z \leftarrow +T_3T_4$

$\alpha\cdot T_1 \leftarrow T_4$

$\alpha\cdot T_1 \leftarrow T_2$

has an expression set

$$v(\pi') = \{+\alpha\cdot *XY\alpha\cdot +XY, \alpha(*XY,\alpha\cdot +XY)(*XY,+XY)\}$$

and is equivalent to the program π of Example 2.1. The equivalence of $v(\pi')$ and $v(\pi) = \{+\alpha\cdot *XY\alpha\cdot +XY, \alpha(*XY,+XY)\}$ is established by (E4), (E3) and (E1). For example,

1.	$\alpha(U,V)(U,W) \equiv \alpha(U,W)$	(E4)
2.	$\alpha(*XY,\alpha\cdot +XY)(*XY,+XY) \equiv \alpha(*XY,+XY)$	(E3),1

Let $\pi = (I,O,P)$ be a proper program, $P = S_1,S_2,\ldots,S_n$. For the sake of compact definitions assume that there are fictitious statements S_0 and S_{n+1}; S_0 assigning a value to every scalar variable name and to each component of every structured variable name in I, and S_{n+1} referencing each scalar variable name in O and each component of every structured variable name in O.

Definition. The assignment $S_i = \alpha \cdot R \leftarrow T$ to the structured variable name α in the program π is irrelevant to a reference $S_k = X \leftarrow \alpha \cdot Y$ $(k>i)$, if there is a statement $S_j = \alpha \cdot A \leftarrow B (i<j<k)$, and either $e(R,i) \equiv e(A,j)$, or $e(A,j) \equiv e(Y,k-1)$.

Definition. For $i > 0$, let S_i be an assignment to the scalar variable T. The scope of S_i is defined by one of the following:

1) If there is a statement S_j $(i<j\leq n+1)$ which contains the last reference to this value of T, then the scope of S_i is the left-hand side of S_i, the statements $S_{i+1},\ldots,S_{j-1}$, and the right-hand side of S_j.
2) If there is no such S_j, then the scope of S_i is the left-hand side of S_i only.

We can now give the formal definition of four transformations on π, analogous to [1].

Recall the convention that all scalar variables in O and all components of structured variables in O are referenced in S_{n+1}. In the following, when a variable on the rhs of S_{n+1} is changed, then the set O is canged accordingly. This change does not alter $v(\pi)$.

(T1) Deletion of a useless statement S_i $(1\leq i\leq n)$ from P.

(1) If S_i is an assignment to a scalar variable and the scope of S_i is only the left hand side of S_i, then S_i is useless and can be deleted.

(2) If $S_i = \alpha \cdot R \leftarrow T$, and S_i is irrelevant to all subsequent references to α, then S_i is useless and can be deleted.

(T2) Deletion of a redundant statement S_j $(1\leq j\leq n)$ from P.

(1) Let S_i and S_j $(i<j)$ be assignments to the scalar variable names T and T', respectively, and assume $e(T,i) \equiv e(T',j)$; then the statement S_j can be deleted from P after the following operations:

(a) Find a variable name $X \in K$ which when substituted for T in the scope of S_i and for T' in the scope of S_j does not alter $v(\pi)$;

(b) Substitute X for every occurrence of T in the scope of S_i and for every occurrence of T' in the scope of S_j.

(2) The statement $S_j = E \leftarrow \alpha \cdot F$ can be deleted from P as redundant, if there is a statement $S_i = \alpha \cdot R \leftarrow T$, $i < j$, and $e(\alpha,i) \equiv e(\alpha,j)$ and $e(R,i) \equiv e(F,j-1)$, after the following has been performed:

(a) If S_k $(k<i)$ is an assignment to T and the right-hand side of S_i is in the scope of S_k, find a name $X \in K$ (which might be T or E) which, when substituted for T in the scope of S_k and for E in the scope of S_j, does not alter $v(\pi)$. Substitute X for T in the scope of S_k, and for E in the scope of S_j.

(b) If T is an input name which is not assigned before S_i, and if there is any S_r $(r>i, r\neq j)$ which is an assignment to T, substitute for T in the scope of S_r a name $X \in K$. Choose X such that $v(\pi)$ is not altered. Then substitute T for E in the scope of S_j.

Note that in T2.2(b) X is substituted for T in the scope of all such S_r simultaneously. This is always possible since the scopes of the S_r cannot intersect. Also note that since E might be T we must exclude $r = j$ in the above.

(3) Let $S_j = \alpha{\cdot}E \leftarrow F$. Then if there is some statement $S_i = \alpha{\cdot}R \leftarrow T$ $(i<j)$, S_j is redundant and can be deleted provided that $e(\alpha,i) \equiv e(\alpha,j-1)$, $e(R,i) \equiv e(E,j)$ and $e(T,i) \equiv e(F,j)$.

(T3) Two independent adjacent statements S_i and S_{i+1} in P can be interchanged; i.e., let

S_i = (1) $A \leftarrow \phi B_1 B_2 \ldots B_r$

(2) $\alpha{\cdot}A \leftarrow B_1$

(3) $A \leftarrow \alpha{\cdot}B_1$

and S_{i+1} = (a) $X \leftarrow \psi\, Y_1 Y_2 \ldots Y_s$

(b) $\beta{\cdot}X \leftarrow Y_1$

(c) $X \leftarrow \beta{\cdot}Y_1$

then S_i and S_{i+1} are independent if for the following combinations these conditions are met:

(1a,1c,3a,3c):		$A \neq X$, $X \neq B_1, \ldots, X \neq B_r$, $A \neq Y_1, \ldots, A \neq Y_s$
(2a):		$A \neq X$, $X \neq B_1$
(1b):		$A \neq X$, $A \neq Y_1$
(2b):	$\alpha \neq \beta$	
(3b):	$\alpha \neq \beta$,	$A \neq X$, $A \neq Y_1$
(2c):	$\alpha \neq \beta$,	$X \neq A$, $X \neq B_1$

(T4) A scalar variable name assigned by some statement S_i in P can be replaced by another scalar variable name X in the scope of S_i if $v(\pi)$ does not change.

Example 3.2.

Consider the program $\pi = (I,O,P)$ where $I = \{A,B,\alpha\}$, $O = \{\alpha\}$, and P =

1. $F \leftarrow *AB$
2. $S \leftarrow +AB$
3. $\alpha{\cdot}F \leftarrow S$
4. $E \leftarrow \alpha{\cdot}F$
5. $Q \leftarrow +EA$
6. $\alpha{\cdot}F \leftarrow Q$

Since $e(\alpha,3) = e(\alpha,4)$ and $e(F,3) = e(F,4-1)$, T2.2 is applicable to π with $i = 3$, $j = 4$ and $R = F$. Case (a) is given here with $k = 2$, and the name X may be chosen to be S. Thus, after applying T2.2, the resulting program is $\pi' = (I,O,P')$, where P' =

1. $F \leftarrow *AB$
2. $S \leftarrow +AB$
3. $\alpha{\cdot}F \leftarrow S$
4. $Q \leftarrow +SA$
5. $\alpha{\cdot}F \leftarrow Q$

Next, consider the statements 3 and 5 in P'. Since $e(F,3) = e(F,5)$ and statement 4 does not reference α, the third statement is useless and can be removed by T1.2. After this, the resulting program is $\pi'' = (I,0,P'')$, where P'' =

1. $F \leftarrow *AB$
2. $S \leftarrow +AB$
3. $Q \leftarrow +SA$
4. $\alpha \cdot F \leftarrow Q$

In π'' the first two statements are independent (combination 1a) and could be interchanged, while the last two statements are not independent. Also, Q could be replaced in the last two statements by, for example, S, applying T4.

Write $\pi \underset{T}{\rightarrow} \pi'$ if π' is obtained from π by a single application of the transformation T, and define T^{-1} by $\pi \underset{T^{-1}}{\rightarrow} \pi'$ if $\pi' \underset{T}{\rightarrow} \pi$. In the following, we are particularly interested in the following sets of transformations:

$$T_0 = \{T1,T2\},\ T = \{T1,T2,T1^{-1},T2^{-1}\},\ F = \{T3,T4\}.$$

Denote by $\overset{*}{\rightarrow}$ the reflexive, transitive closure of the relation $\rightarrow$. As a preliminary characterization we prove the following properties:

Lemma 3.1. If $\pi \underset{T3}{\rightarrow} \pi'$ then $\pi' \underset{T3}{\rightarrow} \pi$.

If $\pi \underset{T4}{\rightarrow} \pi'$ then $\pi' \underset{T4}{\rightarrow} \pi'$.

Proof: Obvious from the definitions. ∎

By this result, $\underset{F}{\overset{*}{\rightarrow}}$ defines an equivalence relation on programs. Programs which are equivalent in this sense are called <u>isomorphic</u>. Denote by $\underset{T}{\overset{*}{\rightarrow}}$ a sequence (possibly empty) of transformations in T. The next result shows that T3 and T4 can be defined using certain sequences of transformations in T.

Lemma 3.2. If $\pi \underset{T3}{\rightarrow} \pi'$ then $\pi \underset{T}{\overset{*}{\rightarrow}} \pi'$.

If $\pi \underset{T4}{\rightarrow} \pi'$ then $\pi \underset{T}{\overset{*}{\rightarrow}} \pi'$.

<u>Proof</u>: Assume $\pi = (I,0,P) \underset{T3}{\rightarrow} \pi'$ by exchanging statements S_i and S_{i+1} in π. Consider the program $\pi'' = (I,0,P'')$ which has statements $P'' = S_1,S_2,\ldots,S_i,S_{i+1},S_r,S_{i+2},\ldots,S_n$ where $S_r = S_i$.

(1) $\pi'' \underset{T2}{\rightarrow} \pi$ by elimination of S_r:

All combinations of S_i, S_{i+1} must be considered.

It is easily verified that the hypotheses of T3 are sufficient to allow a legal application of T2 to S_r. For example, in the case 2c:

$S_i \quad : \alpha \cdot A \leftarrow B$

$S_{i+1} : \quad X \leftarrow \beta \cdot Y$

$S_r \quad : \alpha \cdot A \leftarrow B$

We have by hypothesis $\alpha \neq \beta$, $X \neq A$, $X \neq B$. Therefore

$e(A,i) = e(A,r)$, $e(B,i) = e(B,r)$, and

$e(\alpha,i) = e(\alpha,r-1)$

So T2.3 is applicable. The other combinations are verified in like manner.

(2) $\pi'' \underset{T1}{\rightarrow} \pi'$ by deletion of S_i:

Again, it is easily verified that the hypotheses for all statement combinations are sufficient to warrant a legal application of T1.

Next, assume $\pi = (I,O,P) \underset{T4}{\rightarrow} \pi'$ by replacing the scalar variable T by X in the scope of S_k; $S_k = T \leftarrow \sigma$. Consider the program $\pi'' = (I,O'',P'')$ where P'' is P with the following additions: There is a statement $S_{k'} = X \leftarrow \sigma$ immediately following S_k, and for each statement S_j referencing T in the scope of S_k there is a statement $S_{j'}$, identical to S_j except that the references to T are replaced by X. Finally, if the scope of S_k includes S_{n+1}, let $O'' = O \cup \{X\} - \{T\}$, otherwise $O'' = O$.

(1) $\pi'' \overset{*}{\underset{T2}{\rightarrow}} \pi$:

It can be seen that the computation $S_{k'}$ is redundant. Apply T2.1 substituting T for X in the scope of S_k. Thereafter, all statements $S_{j'}$ must be redundant and are eliminated. No new name substitutions are necessary.

(2) $\pi'' \overset{*}{\underset{T1}{\rightarrow}} \pi'$:

The computations S_j are useless. After their elimination, the statement S_k is useless and can be eliminated. ∎

<u>Lemma 3.3</u>. There are proper programs π_1, π_2, π_3, π_4 such that

$$\pi_1 \underset{T1}{\rightarrow} \pi_2 \text{ but not } \pi_1 \overset{*}{\underset{T2}{\rightarrow}} \pi_2 \text{ and}$$

$$\pi_3 \underset{T2}{\rightarrow} \pi_4 \text{ but not } \pi_3 \overset{*}{\underset{T1}{\rightarrow}} \pi_4 .$$

<u>Proof</u>. Observe first, that the effect of T1.1 cannot be achieved by T1.2 or T2.3, since by those transformations different statement types are eliminated. Similarly, other transformation combinations can be excluded. Define programs $\pi_i = (I,O,P_i)$ where $i = \{A,B,\alpha\}$, $O = \{X,\alpha\}$,

$$P_1 = \begin{array}{l} \alpha\cdot A \leftarrow B \\ X \leftarrow +AB \\ \alpha\cdot A \leftarrow B \end{array} \qquad P_2 = \begin{array}{l} X \leftarrow +AB \\ \alpha\cdot A \leftarrow B \end{array}$$

then $\pi_1 \xrightarrow[T1.2]{} \pi_2$, but no application of T1.1 or T2 can eliminate the first statement of π_1. Next, let

$$P_3 = \begin{array}{l} \alpha\cdot A \leftarrow B \\ T \leftarrow \alpha\cdot A \\ X \leftarrow +AT \end{array} \qquad P_4 = \begin{array}{l} \alpha\cdot A \leftarrow B \\ X \leftarrow +AB \end{array}$$

then $\pi_3 \xrightarrow[T2.2]{} \pi_4$, but no application of T2.1, T2.3 or T1 can eliminate the second statement of P_3. The proof for the other transformations is left as an exercise. ∎

The lemma proves the indepeendence of the transformations in T_0. Note that in effect we have shown that T1.1,T1.2,T2.1,T2.2,T2.3 are independent. We conclude the preliminary characterization by establishing the equivalence preserving property of the transformations T1 and T2. Since none of these transformations changes the input set, it suffices to show that the resulting associated expression sets are equivalent. Although the theorem is intuitive, the complete proof is lengthy and must cover many details.

<u>Theorem 3.4</u>. Let π and π' be programs and assume $\pi \xrightarrow[T_0]{*} \pi'$.

Then $v(\pi) \equiv v(\pi')$.

<u>Proof</u>: It suffices to show the theorem for a single application of T1 and T2. In particular, if we show that the associated expressions in π' are equivalent to their corresponding expressions in π where $\pi \xrightarrow[T_1]{} \pi'$ or $\pi \xrightarrow[T_2]{} \pi'$, then $v(\pi) \equiv v(\pi')$ follows trivially.

(a) T1.1 was applied. Trivial

(b) T1.2 was applied to $S_i = \alpha \cdot R + T$.

<u>Case 1</u>: There was no subsequent reference to α: Trivial.

<u>Case 2</u>: It suffices to show that for each subsequent reference

$$S_k = X + \alpha \cdot Y$$

to α we have

$$e(X,k) \equiv e'(X,k-1).$$

where S'_{k-1} is the corresponding statement in π' and $e'(\ldots)$ denotes the associated expression in π'. This is accomplished by induction on the number of these references.

<u>Basis</u>

Let S_k be the first reference to α following S_i. There is some statement

$$S_j = \alpha.A + B, \ i < j < k,$$

such that either

$$e(R,i) \equiv e(A,j) \quad \text{(subcase aa)},$$

or

$$e(A,j) \equiv e(Y,k-1) \quad \text{(subcase bb)}.$$

Observe that $e(A,j) = e(A,j-1)$ and $e(R,i) = e(R,i-1)$. Since S_k is the first reference to α following the eliminated statement, necessarily

$$e(Y,k-1) \equiv e'(Y,k-2) \quad (*).$$

Now,

$$e(X,k) = e(\alpha,i-1)(t)(E_2,V_2)..(t')..(E_n,V_n).e(Y,k-1),$$

where $t = e(R,i),e(T,i-1)$

and $t' = e(A,j),e(B,j-1)$.

So, in subcase aa,

$$e(X,k) \equiv e(\alpha,i-1)(E_2,V_2)..(t')..(E_n,V_n)\cdot e(Y,k-1)$$

rule (E4) and substitution

$$\equiv e(\alpha,i-1)(E_2,V_2)..(t')..(E_n,V_n)\cdot e'(Y,k-2)$$

substitution using (*)

$$= e'(X,k-1);$$

and, in subcase bb,

$$e(X,k) \equiv e(\alpha,i-1)(E_2,V_2)..(t')..(E_n,V_n)\cdot e(Y,k-1)$$

rule (E5) and substitution

$$\equiv e(\alpha,i-1)(E_2,V_2)..(t')..(E_n,V_n)\cdot e'(Y,k-2)$$

substitution using (*)

$$= e'(X,k-1).$$

<u>Induction step</u>

Suppose the claim is true for the first m references to α following S_i. Let

$$S_k = X \leftarrow \alpha.Y$$

be the m+1st reference to α . The induction hypothesis implies that $e(Z,k-1) \equiv e'(Z,k-2)$ for all scalar variables in the program π . Therefore $e(Y,k-1) \equiv e'(Y,k-2)$. The same argument as in the induction basis completes the induction step.

(c) T2.1 was applied.

Since $e(T,i) \equiv e(T',j)$, the theorem trivially holds.

(d) T2.2 was applied.

There are statements

$$S_i = \alpha.R \leftarrow T, \text{ and}$$

$$S_j = E \leftarrow \alpha.F,\ i < j,$$

such that $e(\alpha,i) \equiv e(\alpha,j-1)$ and $e(R,i) \equiv e(F,j-1)$.

Then
$$
\begin{aligned}
e(E,j) &= e(\alpha,j-1).e(F,j-1) \\
&= e(\alpha,j).e(F,j-1) \\
&\equiv e(\alpha,i).e(R,i) && \text{(substitution)} \\
&= e(\alpha,i-1)(e(R,i-1),e(T,i-1)).e(R,i-1) \\
&\equiv e(T,i-1) && \text{(axiom scheme)}
\end{aligned}
$$

Assume that substitution scheme (a) of T2.2 is applicable. Having substituted X for T and for E we get

$$e(T,i-1) = e'(X,i-1) \text{ and}$$

$$e'(X,j-1) = e'(X,i-1).$$

But then $e(E,j) \equiv e'(X,j-1)$ which implies that the theorem is true for this case.

Now consider substitution scheme (b) of T2.2. The change of occurrences of T in the scope of assignments to T does not change $v(\pi)$. The change of E to T in the scope of S_j, the deletion of S_j and the earlier variable changes give $e'(T,j-1) = e'(T,i-1) \equiv e(E,j)$. Since $e(T,i-1) = e'(T,i-1)$ the theorem is also true in this case.

(e) Let $S_i = \alpha.R \leftarrow T$

and suppose that

$S_j = \alpha.E \leftarrow F,\ i < j,$

was eliminated from π using T2.3.

Then $e(\alpha,i) \equiv e(\alpha,j-1)$,

$e(R,i) = e(R,i-1) \equiv e(E,j) = e(E,j-1)$, and

$e(T,i-1) = e(T,i) \equiv e(F,j) = e(F,j-1)$.

Any reference to α following S_j in π computes an expression of the form $G = e(\alpha,j)(G_1,V_1)\ldots(G_n,V_n).H$ for some scalar expressions G_j, V_j and H, where

$$e(\alpha,j) = e(\alpha,i-1)(e(R,i),e(T,i))(e(E,j),e(F,j)).$$

But then $e(\alpha,j) \equiv e(\alpha,i-1)(e(R,i),e(T,i))(e(R,i),e(T,i))$ by substitution

$\equiv e(\alpha,i-1)(e(R,i),e(T,i))$, by (E4) and substitution.

Therefore, substituting,

$$G \equiv e(\alpha,i-1)(e(R,i-1),e(T,i-1))(G_1,V_1)\ldots(G_n,V_n).H$$
$$= G',$$

which is the expression computed by the corresponding statement in π'.

This completes the proof of the theorem. ∎

4. The Classes E_0, R_0 and I_0

We have showed that various program transformations preserve program (expression) equivalence (Theorem 3.4). In this section we further investigate the relationship between transformations on programs and program equivalence. In particular, the converse of Theorem 3.4 is proved. We also define certain classes of programs, which have interesting properties. The following definition provides a useful tool for proving several of the theorems of this section.

Definition. A proper program is called open, if no scalar name is assigned more than once, and no assignment is made to an input name.

The program of Example 2.1 is open. Intuitively, each multiple assignment can be removed by applications of T4. This can always be achieved.

Lemma 4.1. To each proper program π there is an equivalent open one π'. Moreover, π' has an expression set identical to π.

Proof. Observe first, that $\pi \overset{*}{\underset{F}{\rightarrow}} \pi'$ implies that π and π' are equivalent (Theorem 3.4 and Lemma 3.2), and that $v(\pi) = v(\pi')$, by the definition of T3 and T4. The remainder of the proof is trivial. ∎

Of interest is the property of reducedness. A proper program is reduced, if neither T1 nor T2 can be applied.

The program π'' of Example 3.2 is reduced, but the program π of Example 3.2 is not.

In the following all programs are assumed proper. We define the following sets:

$$E_0(\pi) = \{\pi' \mid v(\pi) \equiv v(\pi'), \text{ and } \pi \text{ and } \pi' \text{ have identical inputs}\}$$

$$R_0(\pi) = \{\pi' \in E_0(\pi) \mid \pi' \text{ is reduced}\}$$

$$I_0(\pi) = \{\pi' \in E_0(\pi) \mid \pi \underset{F}{\rightarrow} \pi'\}.$$

$E_0(\pi)$ is the class of all programs equivalent to π, $R_0(\pi)$ is the subset of reduced equivalent programs, and $I_0(\pi)$ is the isomorphism class of π, i.e., those programs equivalent to π which differ only in the sequence of statements and/or names of scalar variables from π. We characterize R_0 first.

<u>Theorem 4.2</u>. Let π and π' be open equivalent programs. If $v(\pi) = v(\pi')$, then there exist programs π_0 and π_0' such that

$$\pi \overset{*}{\underset{T_0}{\rightarrow}} \pi_0 \overset{*}{\underset{F}{\leftrightarrow}} \pi_0' \overset{*}{\underset{T_0}{\leftarrow}} \pi' .$$

<u>Proof</u>. Apply T1 and T2 to both π and π' eliminating useless and redundant statements, so long as $v(\pi) = v(\pi')$ is not altered. This can be done in such a way that the resulting programs π_0 and π_0' are open, although not necessarily reduced.

Let π_0' have the statements $S_1, S_2, \ldots, S_n$, and assume, without loss of generality, that the number of statements

in π_0 is not less than n. We will construct programs $\pi_0, \pi_1, \ldots, \pi_n$, such that 1) the first i statements of π_i are $S_1, S_2, \ldots, S_i$; 2) π_i is open and; 3) $\pi_i \overset{*}{\underset{F}{\rightarrow}} \pi_{i+1}$ for $i < n$. Clearly, π_0 satisfies all these requirements.

Transformation of π_i into π_{i+1}:

Consider the statement S_{i+1}. The expression which it computes appears as a subexpression in $v(\pi_0') = v(\pi_i)$, for otherwise S_{i+1} is useless in π_0' and could have been eliminated from π_0' without altering $v(\pi')$. Since π_0' and π_i have the first i statements in common, there is a statement T_j $(j>i)$ in π_i computing the same expression. Were this not the case, S_{i+1} could have been eliminated from π_0' as redundant without altering $v(\pi')$ contrary to assumption.

If the variable X assigned by S_{i+1} in π_0' is scalar and is assigned in π_i, this must be done by some T_k, $k > i$, because the first i statements in π_i are $S_1, S_2, \ldots, S_i$, and π_0' is open. Rename X in π_i to be some name not yet occurring in π_i or π_0' (T4), then apply T4 to π_i renaming scalar variables in T_j and elsewhere in π_i if necessary such that the resulting $T_j' = S_{i+1}$. If S_{i+1} is a structured assignment, similarly rename scalar variables in π_i so that $T_j' = S_{i+1}$. The resulting program π_{i+1}' must be open and $v(\pi_i) = v(\pi_{i+1}')$. It remains to be shown that a sequence of applications of T3 can move T_j' into position i+1, thereby obtaining the program π_{i+1}:

Case 1: $T'_j = X \leftarrow \phi Y_1 Y_2 \dots Y_r$.

There is no problem since π'_{i+1} is open and the Y_k are computed by $S_1, \dots, S_i$.

Case 2: $T'_j = \alpha \cdot X \leftarrow Y$

There can be no assignment to α among the $T_{i+1}, \dots, T_{j-1}$, for otherwise T'_j and S_{i+1} cannot compute the same expression.

Let $B \leftarrow \beta . A$ be T_s for $i < s < j$. If $\alpha = \beta$, then either T_s could have been eliminated from π_0 without altering $v(\pi_0)$, or there must be a corresponding statement among the $S_{i+2}, \dots, S_n$ in π'_0 computing the same expression. But that cannot be as the subexpressions for the value of α before T_s and after S_{i+1}, must always differ. Since $T'_j = S_{i+1}$ and the programs are open, X and Y are either inputs or are assigned in $S_1, \dots, S_i$. Hence $X \neq B$ and $Y \neq B$ so T3 can be used for T_s and T'_j. If T_s, $i < s < j$, is $B \leftarrow \phi A_1 \dots A_n$, then again $X \neq B$ and $Y \neq B$ so T3 can be used. Therefore, T3 can be legally applied to move T'_j to the i+1 position giving π_{i+1}.

Case 3: $T_j = X \leftarrow \alpha \cdot Y$

By analogous argumentation it can be shown that T3 can be applied to move T'_j to the i+1 position giving π_{i+1}.

After n steps this process ends with the program π_n. If π_n has more than n statements, they are now useless, so

they must have been redundant in π_0 and could have been eliminated without altering $v(\pi_0)$, contrary to assumption. Thus, $\pi_n = \pi'_0$, and, by symmetry, $\pi_0 \overset{*}{\underset{F}{\longleftrightarrow}} \pi'_0$. ∎

Lemma. Assume E and E' are different but equivalent expressions and such that whenever G is a proper subexpression of E to which there exists a corresponding equivalent subexpression G' of E', then $G = G'$. Then the following are true:

(1) At least one application of the rules (E4), (E5) or the axiom scheme must be made on establishing $E \equiv E'$.

(2) One of these, in combination with (E3), must be applied to E or E', rather than to subexpressions of E and E'. References in the following to applications of (E4), (E5) or the axiom implicitly assume the use of (E3).

(3) If the axiom scheme is applied to E such that the length of the expression increases, then it can be applied to E' the reverse direction, decreasing the laugh of E'.

(4) If (E4) or (E5) must be applied to E(or E') in establishing $E \equiv E'$, then there is a part of expressions (H,V) occurring as subexpression of E (or E') to which there is no corresponding equivalent part of expressions on E' (or E). (I.e., no pair of sub-expressions of E' (or E) which results from (H,V) via the transformations which demonstrate the equivalence of E and E'.)

(5) Hence if (E4) or (E5) must be applied to E or E' in establishing $E \equiv E'$, then (E4) or (E5) must be applied in the length decreasing direction to either E or E'. The pair of subexpressions eliminated by this step in E or E' has no corresponding equivalent pair of subexpression in E' or E.

Proof.

(1) From the definition of equivalence it is clear that if neither the axiom scheme, nor one of the rules (E4) or (E5) is applied in establishing the equivalence, then $E = E'$.

(2) If these application(s) are made only to subexpressions of E and E', then we can find proper equivalent subexpressions in E and E' which are not equal.

(3) Assume

$$E \equiv G(H,E).F \equiv E', \text{ where } H \equiv F$$

and the first step is the conversion of E to $G(H,E).F$. If the first step is to be necessary, then E' must be of the form

$$G1.F'$$

with $G1 \equiv G(H,E)$ and $F' \equiv F$. But then

$$G1 = G'(H',V')$$

with $G' \equiv G$, $H' \equiv H$, and $V' \equiv E$, therefore, by assumption, $V' = E$. Thus,

$$E' = G'(H',E).F' \text{ with } H' \equiv F'.$$

Assume (E5) must be applied in the length increasing direction. Assume $E \equiv E_1 \equiv E'$ where

$$E = \alpha(H_1,V_1)\ldots(H_n,V_n).F$$

$$E_1 = \alpha(H_0,V_0)(H_1,V_1)\ldots(H_n,V_n).F \text{ and } F \equiv H_j \text{ for some } j \geq 1.$$

and E_1 is obtained in one step from E via (E5) and (E3).

Now

$$E' = \alpha(H_1',V_1')\ldots(H_m',V_m').F'$$

is the only case of interest, otherwise the axiom must have been applied to E', in which case the argument of (3) completes the proof.

Consider the subexpression $G_0 = \alpha(H_0,V_0)$ of E_1. If in establishing $E \equiv E'$ the transformation from E to E_1 is necessary, then there must be a subexpression G' of E' equivalent to G_0:

Assume there is no such G', then the part (H_0,V_0) must have been eliminated in going to E'. If this was done by (E5), then clearly going to E_1 is unnecessary. If this was done by (E4), then there must be some $i \geq 1$ such that $H_i \equiv H_0$. Take k to be the largest such i, and consider

$$G_k = \alpha(H_0,V_0)\ldots(H_k,V_k).$$

If we cannot find a subexpression G" of E' equivalent to G_k, then the part (H_k,V_k) is eliminated in going to E'. This now can only be done by (E5) and therefore also (H_0,V_0) must be eliminated by (E5) and it was unnecessary to go to E_1. If G" exists, then $G" \equiv G_k \equiv G'_k = \alpha(H_1,V_1)\dots(H_k,V_k)$, and it was also unnecessary to go to E_1.

Therefore there exists a subexpression G' of E' equivalent to $G_0 = \alpha(H_0,V_0)$. Let G' be the subexpression having this property which comes from G_0 as a result of the proof that $E_1 \equiv E'$. Clearly then $G' = W(H',V')$ for some W,H',V', so

$$E' = \alpha(H'_1,V'_1)\dots(H',V')\dots(H'_m,V'_m).F'$$

and then (H',V') is a pair to which there exists no corresponding equivalent part of expressions on E. (Because the equivalence proof in going from E' to E_1 converts $G' = W(H',V')$ to $G_0 = \alpha(H_0,V_0)$ and (H_0,V_0) is then eliminated in going to E.)

The argument in the case of applying (E4) is analogous.

<u>Theorem 4.3</u>. Let π and π' be reduced and equivalent. Then $v(\pi) = v(\pi')$.

<u>Proof</u>. Assume $v(\pi) \neq v(\pi')$. We then can find expressions M in $v(\pi)$ and M' in $v(\pi')$ such that M and M' are equivalent but not equal. There must be subexpressions E in M and E' in M' satisfying the hypotheses of the Lemma preceding, since M and M' have only finitely many subexpressions. By the lemma, and by symmetry, only three cases need to be considered, the axiom, (E4) or (E5) is used on E or E' as an essential step in establishing $E \equiv E'$. We show that in each of these cases either π or π' is not reduced, contradicting the hypothesis.

<u>Case 1</u>. The axiom scheme and substitution has been applied.

$$\alpha(X,Y).S \equiv Y \text{ if } X \equiv S.$$

By the lemma (3) it must be applied in the length decreasing direction to E or E'. Assume that E has this property. Then E must be of the form $\alpha(E_1,V_1)\dots(E_n,V_n).F$ with $E \equiv F$. The expression E is computed by some statements in π among which we can find

$$S_i = \alpha.X \leftarrow Y$$
$$S_j = T \leftarrow \alpha.S$$

with $i < j$, $e(T,j) = E = e(\alpha,i-1)(e(X,i-1),e(Y,i-1)).e(S,j-1)$. Since

$$e(\alpha,i) \equiv e(\alpha,j) \text{ and}$$
$$e(X,i-1) = e(X,i) \equiv e(S,j-1) \text{ by hypothesis,}$$

T2.2 is applicable to π contradicting its reducedness.

<u>Case 2</u>. Rule (E4) $\alpha(E_1,V_1)\dots(E_i,V_i)\dots(E_n,V_n)$

$$\equiv \alpha(E_1,V_1)\dots(E_{i-1},V_{i-1})(E_{i+1},V_{i+1})\dots(E_n,V_n),$$

$E_n \equiv E_i$, was applied to E in the length decreasing direction. We must be able to find statements in

$$S_j = \alpha.X \leftarrow Y$$
$$S_k = \alpha.T \leftarrow R \qquad k > j$$

with $e(X,j) \equiv e(T,k)$.

Clearly, S_j is irrelevant to all references to α following S_k. We show that it is also irrelevant to all references to α between S_j and S_k. Since the expression $H = e(\alpha,k)$ is equal to E and is equivalent to a subexpression H' of E' not containing a term corresponding to

$F_0 = (e(X,j),e(Y,j))$, (cf. Lemma , part (4), (5)) there is no statement in π' corresponding to S_j. Assume S_h, $j < h < k$, references α and consider the expression F computed by S_h which is a subexpression of an expression G occurring in $v(\pi)$. The corresponding expression G' in $v(\pi')$ cannot contain a term corresponding to F_0 occurring in F, since there is no statement corresponding to S_j in π'. Therefore, this term can be eliminated in going from G to G' by one of the rules (E4), (E5), or the axiom scheme. The conditions which permit the use of (E4), (E5) or the axiom also show that S_j is irrelevant to the reference to π in S_h, therefore S_j is useless, contradicting the reducedness of π.

<u>Case 3</u>: Rule (E5) $\alpha(E_1,V_1)(E_2,V_2)\ldots(E_i,V_i)\ldots(E_n,V_n).F$

$$\equiv \alpha(E_2,V_2)\ldots(E_i,V_i)\ldots(E_n,V_n).F,\quad F \equiv E_i$$

was applied to E in the length decreasing direction. We therefore can find statements in π

$$S_j = \alpha.X \leftarrow Y$$

$$S_h = \alpha.T \leftarrow R$$

$$S_k = \quad Q \leftarrow \alpha.P \qquad j < h < k$$

with $e(X,j-1) = E_1$, $e(Y,j-1) = V_i$,

$$e(T,h) = E_i \equiv e(P,k-1) = F,\ e(R,h-1) = V_i.$$

By an argument essentially like that of case 2 it is straight-forward to see that there will be no statement in π' corresponding to S_j, and that S_j is useless, contradicting the reducedness of π. ∎

The deeper reason for this fairly strong result lies in the fact, that because $v(\pi)$ is constructed from the program π the expressions in $v(\pi)$ are not independent: The elimination of an assignment statement to the structured variable name α affects all subsequently computed expressions containing α. Also, it should be noted that given a set of expressions and of inputs it is not always possible to construct a program computing these expressions:

<u>Example 4.2</u>. Assume a set of input names $\{A,B,C,D,\alpha\}$ and a set of expressions to be computed

$$\{\alpha(C,D)(A,B).B,\ \alpha(A,B).D\} .$$

There is no straight line program computing this set of expressions from the given inputs. The first expression requires that the first assignment to α in such a program be

$$\alpha.C \leftarrow D,$$

whereas the second expression to be computed requires

$$\alpha.A \leftarrow B$$

to be the first assignment to α.

It is, however, possible to compute the equivalent set

$$\{\alpha(A,B)(A,\alpha.A)(C,D)(A,B).B,\alpha(A,B).D\}$$

by the program $\pi = (\{A,B,C,D,\alpha\},\{X,Y\},P)$, where

$$
\begin{aligned}
P = \quad & T \leftarrow \alpha.A \\
& \alpha.A \leftarrow B \\
& X \leftarrow \alpha.D \\
& \alpha.A \leftarrow T \\
& \alpha.C \leftarrow D \\
& \alpha.A \leftarrow B \\
& Y \leftarrow \alpha.B
\end{aligned}
$$

Equivalence of the two expressions is established by

$$
\begin{aligned}
& \alpha(A,B)(A,\alpha.A)(C,D)(A,B).B & \\
\equiv\ & \alpha(A,\alpha.A)(C,D)(A,B).B & \text{(E4 and substitution)} \\
\equiv\ & \alpha(C,D)(A,B).B & \text{(E4)}
\end{aligned}
$$

therefore, the simplification of an expression set may result in an expression set which cannot be computed. This constitutes another reason why it is not desirable to optimize programs based on the representation of associated expression sets.

Lemma 4.4. If π is reduced and $\pi' \in I_0(\pi)$, then π' is reduced.

Proof. It suffices to show that a single application of T3 or T4 preserves reducedness. For T4 this is obvious. Assume $\pi \xrightarrow[T3]{} \pi'$ and π' is not reduced. Since T3 does not alter the expressions computed by the interchanged statements, any expression computed remains unchanged. Also T3 does not alter names. A case by case analysis can be done to show that if π' is not reduced, then π was not reduced. The details are left to the reader. ∎

Theorem 4.5. Assume π is reduced. Then $I_0(\pi) = R_0(\pi)$, i.e., reduced equivalent programs are isomorphic, and if π' is equivalent to π and is not reduced, then π and π' cannot be isomorphic.

Proof. $I_0(\pi) \subseteq R_0(\pi)$: Lemma 4.4.

$R_0(\pi) \subseteq I_0(\pi)$: The inclusion follows trivially from Lemma 4.1, Theorem 4.3 and the proof of Theorem 4.2. ∎

The theorem summarizes the characterization of $R_0(\pi)$ in terms of transformations: All equivalent reduced programs are of same length, are isomorphic, and have equal expression sets. Thus expression sets reflect all the vital information about a reduced program. Apart from being valuable in its own right the theorem lends itself to an elegant characterization of $E_0(\pi)$ in terms of code transformations.

Theorem 4.6. Let π and π' be programs. Then π is equivalent to π' if and only if $\pi \overset{*}{\underset{T}{\leftrightarrow}} \pi'$.

Proof. " $\Leftarrow$ " Theorem 3.4.

" $\Rightarrow$ " Apply T1 and T2 to π and π' obtaining the reduced programs π_0 and π_0'. By Theorem 3.4, $v(\pi) \equiv v(\pi_0) \equiv v(\pi') \equiv v(\pi_0')$. By Theorem 4.5 $\pi_0 \overset{*}{\underset{F}{\leftrightarrow}} \pi_0'$. Since T3 and T4 are combinations of T1, T2 and their inverses (Lemmas 3.2 and 3.3), the theorem is proved. ∎

The theorem shows that the search of the class $E_0(\pi)$ for a certain equivalent program can be reduced to the search for a suitable sequence of code transformations which applied to π gives the desired program as result. In conjunction with Theorem 3.4 it establishes T as a minimal complete set of code transformations characterizing program equivalence.

<u>Corollary 4.7</u>. Let π and π' be programs. Then π is equivalent to π' if and only if

$$\pi \xrightarrow[T_0]{*} \pi_0 \xleftrightarrow[F]{*} \pi_0' \xleftarrow[T_0]{*} \pi' . \qquad \blacksquare$$

5. Cost Functions and Optimization

As the primary application of the theory developed so far we give some consequences it has on code optimization. Formally an optimization algorithm finds for a given input program π, a program π_0, equivalent to π, which is minimal with respect to a given cost function $C(\pi)$. In particular, such an algorithm will apply certain transformations to the input program which eventually result in the program π_0. We show how such algorithms optimizing with respect to a large class of cost functions can be structured in terms of the code transformations T1, T2, T3 and T4.

<u>Definition</u>. A function C mapping programs π into a non-negative number $C(\pi)$ is said to be a <u>sequential cost function</u> iff

(C1) for any programs π and π', if $\pi \xrightarrow[T_0]{*} \pi'$, then $C(\pi') \leq C(\pi)$.

(C2) given any program π there is a program $\pi_0 \in E_0(\pi)$ which is minimal with respect to C, i.e., $C(\pi_0) \leq C(\pi')$ for all $\pi' \in E_0(\pi)$.

The definition follows [4]. The reader is referred to [4] (pp. 861-867) for a discussion of the motivation of this definition as well as examples of functions which do not qualify as cost functions. Again, we consider only proper programs.

Optimizing with respect to a given sequential cost function C means therefore a search of $E_0(\pi)$ for a program π_0 for which cost is minimal. Because of the above definition this search can be restricted to the class $R_0(\pi)$.

<u>Lemma 5.1</u>. Let π be a program. There is a program π' in $E_0(\pi)$ and a program π'' in $R_0(\pi)$ such that $\pi \xrightarrow[T2]{*} \pi' \xrightarrow[T1]{*} \pi''$.

<u>Proof</u>. See Appendix.

A modification of the proof of the lemma can be used to define an algorithm finding π' and π''. The lemma shows that a reduced program can always be found by applications of T2 followed by applications of T1.

<u>Lemma 5.2</u>. Let π be open and reduced, and suppose $\pi \xrightarrow[T4]{*} \pi_1 \xrightarrow[T3]{} \pi'$. Then there is a program π_2 such that $\pi \xrightarrow[T3]{} \pi_2 \xrightarrow[T4]{*} \pi'$.

<u>Proof</u>. See Appendix.

The lemma states a restricted kind of commutativity which exists for T3 and T4 if π is open and reduced. There is an obvious

<u>Corollary 5.3</u>. Let π be open and reduced, and $\pi' \in R_0(\pi)$. Then there is a program π'' such that $\pi \xrightarrow[T3]{*} \pi'' \xrightarrow[T4]{*} \pi'$.

<u>Proof</u>. By Theorem 4.5, $R_0(\pi) = I_0(\pi)$. The result then follows directly from the lemma. ∎

It can be shown that the hypotheses of Lemma 5.2 are necessary. There exist reduced programs such that $\pi \xrightarrow[T3]{} \pi_1 \xrightarrow[T4]{*} \pi'$ but no reduced program π_2 such that $\pi \xrightarrow[T4]{*} \pi_2 \xrightarrow[T3]{} \pi'$, and similarly if T4 has been applied first.

The main result concerning the structuring of optimization algorithms is the following.

<u>Theorem 5.4</u> (Canonical Form Theorem). Let C be a sequential cost function. There is an algorithm which optimizes any program π with respect to C by applying first the operation ξ and then the operation χ to π. Furthermore, ξ is independent of C and can be written as $\xi = \xrightarrow[T2]{*} \xrightarrow[T1]{*} \xrightarrow[T4]{*}$, and χ can be written as $\chi = \xrightarrow[T3]{*} \xrightarrow[T4]{*}$.

<u>Proof</u>. Let π be a program. Using Lemma 5.1, a reduced program π_1 can be found by applying T_2 repeatedly, then T_1 repeatedly. I.e.,

$$\pi \xrightarrow[T_2]{*} \bar{\pi} \xrightarrow[T_1]{*} \pi_1$$

where $\pi_1 \in R_0(\pi)$. Next use T_4 to get an open program π_2.

$$\pi \xrightarrow[T_2]{*} \bar{\pi} \xrightarrow[T_1]{*} \pi_1 \xrightarrow[T_4]{*} \pi_2.$$

Now $\pi_2 \in R_0(\pi)$ and π_2 is open so by Lemma 5.2 any member of $R_0(\pi_2) = I_0(\pi_2)$ can be reached via transformations of the form $\xrightarrow[T_3]{*} \xrightarrow[T_4]{*}$. But the optimal program with respect to C is in $R_0(\pi)$. ∎

Intuitively, ξ first applies the transformation of Lemma 5.1 and then T4 repeatedly to find an open reduced program π_1 equivalent to π. Having thus satisfied the hypotheses of Corollary 5.3, the operation χ then transforms π_1 into the optimal program.

It is not difficult to give an algorithm for ξ. But even for a conceptually simple sequential cost function an algorithm for the T3 part of χ may be hard to make efficient.

Example. Define a sequential cost function $C(\pi) = n+m$, where n is the number of statements in π and m is the number of different variable names occurring in π. It is easy to see that this function satisfies (C1) and (C2). The optimal program π_0 equivalent to π minimizes the total storage requirements and execution time of the program.

The problem of specifying a good algorithm for χ has been solved in the case where the value of a computation is referred to at most once, i.e., no common subexpressions exist, and there are no structured variables. See for example [6,7,12]. An algorithm for χ in the case of common subexpressions however has been shown by Sethi to be polynomial complete [11].

6. Conclusions

Assuming a minimal interpretation of the $\cdot$ operator for structured variables we have investigated the problem of finding a minimal complete set of code transformations preserving program equivalence. As an application of the results of Section 4, we have shown how optimization algorithms for a large class of sequential cost functions can be structured, and that such algorithms can be designed modularly with a first phase independent of the particular cost function.

The results of this investigation generalize the work of Aho and Ullman in [1,4]. The approach should provide an environment for further systematic research of code optimization with specific emphasis on the problems arising from the presence of structured variables. Present research by the first author investigates the extensibility of optimization algorithms when more axiom schemes are added.

A number of related interesting questions arise naturally. Among them we mention the following problems: Given a more complete definition of the set of values V, can the addition of a finite number of axiom schemes "grow" the equivalence class to the "limit" class $E_1(\pi) = \{\pi' | \pi$ and π' compute the same output values for all input assignments$\}$, assuming specific interpretations of the operators $\phi \in \Omega$. If so, what would be a minimal complete axiom set?

APPENDIX

Lemma 5.1. Let π be a legal program. There is a program π_1 in $E_0(\pi)$ and a program π_2 in $R_0(\pi)$ such that

$$\pi \overset{*}{\underset{T2}{\rightarrow}} \pi_1 \overset{*}{\underset{T1}{\rightarrow}} \pi_2 .$$

Proof. Apply T2 as long as possible to π yielding a program π_1, and then apply T1 as long as possible to π_1 yielding a program π_2. We need to show that π_2 is reduced. Since T1 is not applicable to π_2 it suffices to show that T2 cannot be applied.

Since $\pi_1 \overset{*}{\underset{T1}{\rightarrow}} \pi_2$, to each statement in π_2 there is a corresponding one in π_1 computing an equivalent sub-expression of an expression occurring in $v(\pi_1)$. Such corresponding statements are identical since T1 does not change names. Note, that the values referenced by the two statements are also equivalent. Denote by e_1 and e_2 the associated expressions of π_1 and π_2, respectively.

Case 1: T2.1 can be applied to π_2:
We can find statements S_i and S_j in π_2 assigning the scalar variables names T and $\bar{T}$, respectively, with

$$e_2(T,i) \equiv e_2(\bar{T},j)$$

and can find corresponding statements $S_{i'}$ and $S_{j'}$ in π_1 assigning T and $\bar{T}$, and

$$e_1(T,i') \equiv e_2(T,i),$$
$$e_1(\bar{T},j') \equiv e_2(\bar{T},j).$$

But then $e_1(T,i') \equiv e_1(\bar{T},j)$, i.e. T2.1, is applicable to π_1, contrary to assumption.

Case 2: T2.2 can be applied to π_2:
By similar reasoning, there are statements S_i and S_j in π_2 and $S_{i'}$ and $S_{j'}$ in π_1 with

$$S_i = S_{i'} = \alpha.X \leftarrow Y$$
$$S_j = S_{j'} = \quad T \leftarrow \alpha.R$$

and
$$\left.\begin{aligned} &e_2(\alpha,i) \equiv e_2(\alpha,j) \\ &e_2(X,i) \equiv e_2(R,j-1) \end{aligned}\right\} \text{(applicability of T2.2)}$$
$$\left.\begin{aligned} &e_2(X,i) \equiv e_1(X,i') \\ &e_2(\alpha,i) \equiv e_1(\alpha,i') \\ &e_2(\alpha,j) \equiv e_1(\alpha,j') \\ &e_2(R,j-1) \equiv e_1(R,j'-1) \end{aligned}\right. \quad (\text{since } \pi_1 \overset{*}{\underset{T1}{\rightarrow}} \pi_2)$$

but then

$$e_1(\alpha,i') \equiv e_1(\alpha,j')$$
$$e_1(X,i') \equiv e_1(R,j'-1),$$

i.e. T2.2, is applicable to π_1, contrary to assumption.

Case 3: T2.3 can be applied to π_2:
By reasoning analogous to case 2, then T2.3 is also applicable to π_1, contrary to assumption. ∎

Lemma 5.2: Let π be open and reduced. If there are programs π_1 and π' such that $\pi \xrightarrow[T4]{*} \pi_1 \xrightarrow[T3]{} \pi'$,

then there is a program π_2 such that

$$\pi \xrightarrow[T3]{} \pi_2 \xrightarrow[T4]{*} \pi'.$$

Proof. Consider π_1: There are statements S_i, S_{i+1} which, by a correct application of T3, are interchanged. We want to show that T3 is applicable to the statements R_i and R_{i+1} or π, thus obtaining π_2, for then clearly $\pi_2 \xrightarrow[T4]{*} \pi'$.

1. $S_i = \alpha.X \leftarrow X_1$, $S_{i+1} = Y \leftarrow \psi Y_1 Y_2, \ldots, Y_s$ or $S_i = Y \leftarrow \psi Y_1 Y_2, \ldots, Y_s$, $S_{i+1} = \alpha.X \leftarrow X_1$

   For applicability of T3 we must have

   $Y \neq X$ and $Y \neq X_1$.

   Since X and X_1 refer to assignments other than S_{i+1} or S_i, respectively, the names corresponding to X and X_1 in π must be different from the name corresponding to Y, for otherwise π would not be open. Thus T3 is applicable to R_i and R_{i+1}.

2. $S_i = \alpha.X \leftarrow X_1$

   $S_{i+1} = \beta.Y \leftarrow Y_1$

   For applicability of T3 we must have $\alpha \neq \beta$. Since T4 cannot rename structured variables T3 can be applied to R_i and R_{i+1}.

3. $S_i = \alpha.X \leftarrow X_1$, $S_{i+1} = Y \leftarrow \beta.Y_1$ or $S_i = Y \leftarrow \beta.Y_1$, $S_{i+1} = \alpha.X \leftarrow X_1$

   Here $\alpha \neq \beta$, $Y \neq X_1$, $Y \neq X$.

   Again, since X and X_1 reference assignments other than S_{i+1} or S_i, respectively, and since π is open, the names corresponding to X and X_1 in π must differ from the name corresponding to Y, and T3 therefore is applicable to R_i, R_{i+1}.

4. For the remaining statement combinations an analogous argument establishes the applicability of T3 to R_i, R_{i+1} in π. ∎

REFERENCES

1. A. V. Aho and J. D. Ullman,
Optimization of Straight Line Programs,
SIAM Journal of Computing, Vol. 1 #1, p. 1-19, March 1972.

2. A. V. Aho and J. D. Ullman,
Equivalence of Programs with Structured Variables,
11th Annul. Symp. on Switching and Aut. Theory,
p. 25-31, 1970.

3. A. V. Aho and J. D. Ullman,
Equivalence of Programs with Structured Variables,
Journal of Comp. and Systems Science, Vol. 6 #2,
p. 125-137, April 1972.

4. A. V. Aho and J. D. Ullman,
The Theory of Parsing, Translation and Compiling,
Vol. II Prentice Hall, p. 844-960, 1973.

5. F. E. Allen,
Program Optimization,
in Annual Review of Automatic Programming, Vol. 5,
p. 239-308, Halpern and Shaw, ed., Pergamon, 1969.

6. J. C. Beatty,
An Axiomatic Approach to Code Optimization of Expressions, JACM Vol. 17 #4, p. 613-640, October 1972.

7. G. Chroust,
Expression Evaluation with Minimal Average Working
Storage, Inform. Processing Letters, Vol. 1 #3,
p. 111-114, February 1972.

8. J. Cocke and J. T. Schwartz,
Programming Languages and Their Compilers,
Courant Institute of Math. Sciences (767 p), 1970.

9. C. M. Hoffmann and L. H. Landweber,
Axiomatic Equivalence of Programs with Structured
Variables, 15th Annual. Symp. on Switching and Automata Theory, 1974.

10. C. M. Hoffmann,
Axiomatic Code Optimization in the Presence of Structured
Variables, Ph.D. Dissertation, Uniersity of Wisconsin,
Madison, 1974.

11. R. Sethi,
Complete Register Allocation Problems,
Proc. 5th Annual. ACM Symp on the Theory of
Computing, 1973.

12. R. Sethi and J. D. Ullman,
The Generation of Optimal Code for Arithmetic
Expressions, JACM Vol. 17 #4, p. 715-728, October 1970.