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 pro-
gram equivalence is defined in terms of expression equiva-
lence. 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 trans-
formations. 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 \( \pi_1 \) and \( \pi_2 \) are equivalent programs, then the given transformations can be used to convert \( \pi_1 \) to \( \pi_2 \). 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) \( \Omega \) is the set of operator symbols. Each operator \( \varphi \) in \( \Omega \) is \( r \)-ary for some positive integer \( r \). The assignment operator \( = \) and the selection operator \( \cdot \) are elements of \( \Omega \).
(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 \( \alpha \) 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 \( \alpha \)". This value will be undefined for all those scalar values of \( A \) which are not in the domain of \( \alpha \).

A statement is a string of symbols having one of the following three forms:
(1) \( X + \varphi Y_1 \ldots Y_r \) (\( \varphi \) an \( r \)-ary operator)
(2) \( X + \alpha \cdot Y \)
(3) \( \alpha \cdot X + Y \)

A statement of type (1) represents an instruction which applies the \( r \)-ary operator \( \varphi \) 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 \( \alpha \) 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 \( + \) operator, respectively.

A program \( \pi \) 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 \( \pi \), respectively. \( P \) denotes a finite sequence of statements.

Example 2.1.

Let \( \pi = (I, O, P) \) where \( I = \{ \alpha \cdot X, Y \} \), \( O = \{ \alpha \cdot Z \} \), \( P = \)
\[
T_1 + \alpha \cdot Y \\
T_2 + \alpha \cdot X \\
T_3 + \alpha \cdot T_1 \\
T_4 + \alpha \cdot T_2 \\
Z + \alpha \cdot T_3 T_4 \\
\alpha \cdot T_1 + T_2
\]

This program assigns to \( Z \) the sum \( \alpha (X \cdot Y) + \alpha (X \cdot Y) \) and re-assigns to \( \alpha (X \cdot Y) \) the sum \( X \cdot Y \), understanding \( \alpha \) 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 \( \Omega \) can be scalar or structured and are defined as follows:

1. \( A \) in \( K \) is a scalar expression.
2. \( \alpha \) in \( S \) is a structured expression.
3. If \( E_1, E_2, \ldots, E_r \) are scalar expressions and \( \phi \) 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 \( G \cdot E \) 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 \( \alpha \): The first assignment to \( \alpha \) was done by some statement \( \alpha \cdot A \rightarrow B \) 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 \( \alpha \) was done by some statement \( \alpha \cdot C \rightarrow D \) 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 \( E_k \) and \( V_k \) will be input variables which can be thought of as having their initial values. The following will make this more precise.

Given the program \( m = (I, O, P) \), 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 \( P \)

\[
\begin{align*}
V1: & \quad e(X,0) = \begin{cases} 
X \text{ for all } X \text{ in } I \text{ and all structured variable names occurring in } P, \\
\text{undefined otherwise}
\end{cases} \\
V2: & \quad \text{if } S_i \text{ in } P \text{ is } X = \phi Y_1 \ldots Y_r, \text{ then} \\
& \quad e(X,i) = e(X,i-1) e(Y_1, i-1) \ldots e(Y_r, i-1) \\
V3: & \quad \text{if } S_i \text{ in } P \text{ is } X = \alpha Y, \text{ then} \\
& \quad e(X,i) = e(\alpha, i-1) e(Y, i-1) \\
V4: & \quad \text{if } S_i \text{ in } P \text{ is } X = \alpha Y + Y, \text{ then} \\
& \quad e(X,i) = e(\alpha, i-1) e(X, i-1) e(Y, i-1) \\
V5: & \quad \text{if } X \text{ in } K \cup S \text{ occurs in } P \text{ and is not assigned by } S_i, \text{ then} \\
& \quad e(X,i) = e(X, i-1) \\
V6: & \quad \text{An expression } e(X,i) \text{ is undefined if any of the expressions required by } (V1) \text{ to } (V5) \text{ is undefined.}
\end{align*}
\]

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

**Example 2.2**

v(m) for the program m of Example 2.1 is 
\( \{+\alpha \cdot X Y a \rightarrow X Y, \alpha \cdot (X Y, +XY)\} \).

Note that each structured subexpression in \( v(m) \) 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 \( v(m) \) must be an input variable. In \( v(m) \), these variables can be
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 $E(X = F)$ 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 $\kappa$ be a finite set of axiom schemes. Two expressions $E$ and $F$ are called equivalent under $\kappa$ (denoted by $E \equiv_{\kappa} F$) if one of the following is true:

(E1) $E = F$ or $(E,F) \in \kappa$ or $(F,E) \in \kappa$.

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

(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 $E = E_0(X = E_1), F = F_0(X = F_1), E_0 \equiv_{\kappa} F_0$ and $E_1 \equiv_{\kappa} F_1$.

(E4) Let $A_1, \ldots, A_n, B_1, \ldots, B_n \in K \cup S$. Then $E = a(a_{1_{1}}(A_{1}, B_{1})\ldots(A_{1_{i-1}}, B_{1_{i-1}})(A_{1_{i}}, B_{1_{i}})(A_{1_{i+1}}, B_{1_{i+1}})\ldots(A_{1_{n}}, B_{1_{n}})$ and $F = a(a_{1_{1}}(B_{1}, A_{1})\ldots(B_{1_{i-1}}, A_{1_{i-1}})(B_{1_{i}}, A_{1_{i}})(B_{1_{i+1}}, A_{1_{i+1}})\ldots(B_{1_{n}}, A_{1_{n}}))$ where $i \leq n$.

(E5) Let $A_1, \ldots, A_n, B_1, \ldots, B_n \in K \cup S$. Then $E = a(a_{1_{1}}(A_{2}, B_{2})\ldots(A_{2_{i-1}}, B_{2_{i-1}})(A_{2_{i}}, B_{2_{i}})(A_{2_{i+1}}, B_{2_{i+1}})\ldots(A_{2_{n}}, B_{2_{n}}))$ and $F = a(a_{2_{1}}(B_{2}, A_{2})\ldots(B_{2_{i-1}}, A_{2_{i-1}})(B_{2_{i}}, A_{2_{i}})(B_{2_{i+1}}, A_{2_{i+1}})\ldots(B_{2_{n}}, A_{2_{n}}))$ where $i > 1$.

In the following we assume that $\kappa$ contains only one axiom scheme:

$\kappa_0 = \{(a(X,Y), X,Y) | X, Y \in K\}$

and consider only equivalence under $\kappa_0$ (denoted by $\equiv$). The axiom scheme in $\kappa_0$ and the rules (E4) and (E5) reflect the interpretation given to the $a$ 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. \( E \equiv F \) \hspace{1cm} \text{given}

2. \( G(E,Y) \cdot X \equiv G(E,Y) \cdot X \) \hspace{1cm} (E1)

3. \( G(E,Y) \cdot E \equiv G(E,Y) \cdot F \) \hspace{1cm} (E3), 1, 2

4. \( \alpha(X,Y) \cdot X \equiv Y \) \hspace{1cm} \text{AXIOM}

5. \( G(E,Y) \cdot E \equiv Y \) \hspace{1cm} (E3), 4

6. \( G(E,Y) \cdot F \equiv V \) \hspace{1cm} (E2), 3, 5

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

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), \(E = \alpha(A_i, B_j) \ldots (A_i, B_j) \ldots (A_n, B_n)\), where \(A_i \equiv A_n\); for (E5), \(E = \alpha(A_i, B_j) \ldots (A_i, B_j) \ldots (A_n, B_n) \cdot R\), 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 substitution.

In considering these rules, the reader should recall that for a proper program \(\pi\), 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.

**Definition.** Two proper programs \(\pi\) and \(\pi'\) are equivalent if \(\pi\) and \(\pi'\) 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 \(v(\pi)\)).

**Example 3.1.**

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

\(O' = (Z, \alpha)\), \(P' = \)

\(T_1 = *XY\)
\(T_2 = +XY\)
\(T_3 = \alpha \cdot T_1\)
\(T_4 = \alpha \cdot T_2\)
\(Z = +T_3 T_4\)
\(\alpha \cdot T_1 = T_4\)
\(\alpha \cdot T_1 = T_2\)

has an expression set

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

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

1. \(\alpha(U, Y) \cdot (U, W) = \alpha(U, W)\) \hspace{1cm} (E4)
2. \(\alpha(*XY, \alpha \cdot +XY) \cdot (*XY, +XY) = \alpha(*XY, *XY)\) \hspace{1cm} (E3), 1

Let \(\pi = (1, 0, P)\) 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 $S_i = \alpha \cdot R + T$ to the structured variable name $\alpha$ in the program $\pi$ is irrelevant to a reference $S_k = X + \alpha \cdot Y (k > i)$, if there is a statement $S_j = \alpha \cdot A + B (i < j < k)$, 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 $S_j$, the statements $S_{j+1}, \ldots, S_{j-1}$, 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 $\pi$, 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 changed 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.

**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 $S_j = E + \alpha \cdot F$ can be deleted from $P$ as redundant, if there is a statement $S_i = \alpha \cdot R + T$, $i < j$, 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 $S_j = a \cdot E - F$. Then if there is some statement

$$S_j = a \cdot R + T \{i<j\},$$

$S_j$ is redundant and can be deleted provided that $e(a,i) = e(a,j-1)$, $e(R,i) = e(E,j)$ and $e(T,i) = e(F,j)$.

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

$$S_i = (1) \quad A + aB_1B_2...B_p$$

$$S_{i+1} = (2) \quad a\cdot A - B_1$$

$$S_{i+1} = (3) \quad A + a\cdot B_1$$

and $S_{i+1} = (a) \quad X + \psi Y_1Y_2...Y_5$

(b) $B\cdot X - Y_1$

(c) $X - B\cdot Y_1$

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

(1a, 1c, 3a, 3c):

$$A \neq X, X \neq B_1, ..., X \neq B_p,$$

$$A \neq Y_1, ..., A \neq Y_5$$

(2a):

$$A \neq X, X \neq B_1$$

(1b):

$$A \neq X, A \neq Y_1$$

(2b):

$$a \neq B$$

(3b):

$$a \neq B, \quad A \neq X, A \neq Y_1$$

(2c):

$$a \neq B, \quad X \neq A, X \neq Y_1$$

(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 $I = (a, B, \alpha)$, $O = \{a\}$, and $P =$

1. $F + A B$
2. $S + A B$
3. $\alpha \cdot F + S$
4. $E + \alpha \cdot F$
5. $Q + S A$
6. $\alpha \cdot F + Q$

Since $e(a, 3) = e(a, 4)$ and $e(F, 3) = e(F, 4-1)$, T2.2 is applicable to $\pi$ 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. $F + A B$
2. $S + A B$
3. $\alpha \cdot F + S$
4. $Q + S A$
5. $\alpha \cdot F + Q$
Next, consider the statements 3 and 5 in $P'$. Since $e(F,3) = e(F,5)$ and statement 4 does not reference $a$, the third statement is useless and can be removed by T1.2. After this, the resulting program is $\pi^* = (I, O, P^*)$, where $P^*$ is

1. $F = \pi AB$
2. $S = \pi AB$
3. $Q = \pi SA$
4. $a \cdot F = Q$

In $\pi^*$ the first two statements are independent (combination la) 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 $\pi \rightarrow \pi'$ if $\pi'$ is obtained from $\pi$ by a single application of the transformation $T$, and define $T^{-1}$ by $\pi \rightarrow \pi'$ if $\pi' \rightarrow \pi$. 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 $\rightarrow^*$ the reflexive, transitive closure of the relation $\rightarrow$. As a preliminary characterization we prove the following properties:

Lemma 3.1. If $\pi \rightarrow \pi'$ then $\pi' \rightarrow \pi$.

If $\pi \rightarrow \pi'$ then $\pi' \rightarrow \pi'$.

Proof: Obvious from the definitions.

By this result, $\pi^*$ defines an equivalence relation on programs. Programs which are equivalent in this sense are called isomorphic. Denote by $\pi T$ 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 $\pi \rightarrow \pi'$ then $\pi T \rightarrow \pi' T$.

If $\pi \rightarrow \pi'$ then $\pi T \rightarrow \pi' T$.

Proof: Assume $\pi = (I, O, P) \rightarrow \pi'$ by exchanging statements $T3$ and $T4$ in $\pi$. Consider the program $\pi^* = (I, O, P^*)$ 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) $\pi^* \rightarrow \pi$ by elimination of $S_r$.

T2: All combinations of $S_1$, $S_{i+1}$ 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:

$S_1 : a \cdot A + B$

$S_{i+1} : X \cdot B \cdot Y$

$S_r : a \cdot A + B$

We have by hypothesis $a \neq B$, $X \neq A$, $X \neq B$. Therefore
e(A,i) = e(A,r), e(B,i) = e(B,r), and
e(a,i) = e(a,r-1)

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

(2) \( \pi' \rightarrow \pi \) 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 \( \pi = (I,0,P) \rightarrow \pi' \) by replacing the
scalar variable \( T \) by \( X \) in the scope of \( S_k \); \( S_k = T + \sigma \).

Consider the program \( \pi'' = (I,0'',P'') \) where \( P'' \) is \( P \) with
the following additions: There is a statement \( S_{k'} = X + \sigma \) 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 \( 0'' = 0 \cup \{X\} - \{T\} \), otherwise \( 0'' = 0 \).

(1) \( \pi'' \rightarrow \pi' \):

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.

\[(2) \quad \pi'' \rightarrow \pi' \quad \text{T1} \]

The computations \( S_j \) are useless. After their
elimination, the statement \( S_k \) is useless and can
be eliminated. \( \Box \)

Lemma 3.3. There are proper programs \( \pi_1, \pi_2, \pi_3, \pi_4 \)
such that

\[
\pi_1 \rightarrow \pi_2 \quad \text{but not} \quad \pi_1 \overset{T1}{\rightarrow} \pi_2 \quad \text{and}
\pi_3 \overset{T2}{\rightarrow} \pi_4 \quad \text{but not} \quad \pi_3 \overset{T1}{\rightarrow} \pi_4 .
\]

Proof. 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 \( \pi_i = (I,0,P_i) \) where \( i = (A,B,a), 0 = (X,a) \),

\[
P_1 = \alpha \cdot A + B \\
P_2 = X + \alpha AB \\
X + \alpha AB \\
\]

then \( \pi_1 \rightarrow \pi_2 \), but no application of T1.1 or T2 can
eliminate the first statement of \( \pi_1 \). Next, let

\[
P_3 = \alpha \cdot A + B \\
P_4 = \alpha \cdot A + B \\
T + \alpha \cdot A \\
X + \alpha AB \\
X + \alpha T
\]
then \( \pi_3 \xrightarrow{T_2.2} \pi_4 \), 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 independence 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.

**Theorem 3.4.** Let \( \pi \) and \( \pi' \) be programs and assume \( \pi \xrightarrow{T_0} \pi' \).

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

**Proof:** It suffices to show the theorem for a single application of \( T1 \) and \( T2 \). In particular, if we show that the associated expressions in \( \pi' \) are equivalent to their corresponding expressions in \( \pi \) 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 \( S_1 = \alpha R + T \).

**Case 1:** There was no subsequent reference to \( \alpha \): Trivial.

**Case 2:** It suffices to show that for each subsequent reference

\[
S_k = X \xrightarrow{\alpha} Y
\]

to \( \alpha \) we have

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

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

**Basis**

Let \( S_k \) be the first reference to \( \alpha \) following \( S_1 \). 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 \( \alpha \) following the eliminated statement, necessarily

\[
e(Y,k-1) \equiv e'(Y,k-2) \quad (\star).
\]

Now,

\[
e(X,k) = e(\alpha,i-1)(t)(E_2, V_2) \ldots (t')(E_n, V_n). e(Y,k-1),
\]
where \( t = e(R,i), e(T,i-1) \)
and \( t' = e(A,j), e(B,j-1) \).

So, in subcase aa,
\[
e(X,k) \equiv e(\alpha,i-1)(E_2, V_2) \ldots (t') \ldots (E_n,V_n) \cdot e(Y,k-1)
\]
rule (E4) and substitution
\[
\equiv e(\alpha,i-1)(E_2, V_2) \ldots (t') \ldots (E_n,V_n) \cdot e'(Y,k-2)
\]
substitution using (*)
\[
= e'(X,k-1);
\]
and, in subcase bb,
\[
e(X,k) \equiv e(\alpha,i-1)(E_2, V_2) \ldots (t') \ldots (E_n,V_n) \cdot e(Y,k-1)
\]
rule (E5) and substitution
\[
\equiv e(\alpha,i-1)(E_2, V_2) \ldots (t') \ldots (E_n,V_n) \cdot e'(Y,k-2)
\]
substitution using (*)
\[
= e'(X,k-1).
\]

**induction step**

Suppose the claim is true for the first \( m \) references to \( \alpha \) following \( S_1 \). Let
\[
S_k = X = \alpha.X
\]
be the \( m+1 \) reference to \( \alpha \). The induction hypothesis implies that \( e(Z,k-1) \equiv e'(Z,k-2) \) for all scalar variables in the program \( \pi \). 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
\[
S_i = \alpha.R + T \quad \text{and} \quad S_j = E + \alpha.F, \quad i < j,
\]
such that \( e(\alpha,i) \equiv e(\alpha,i-1) \) and \( e(R,i) \equiv e(F,j-1) \).

Then
\[
e(E,j) = e(\alpha,j-1).e(F,j-1)
\]
\[
= e(\alpha,j).e(F,j-1)
\]
\[
= e(\alpha,i).e(R,i) \quad \text{(substitution)}
\]
\[
= e(\alpha,i-1)(e(R,i-1), e(T,i-1)).e(R,i-1)
\]
\[
= e(T,i-1) \quad \text{(axiom scheme)}
\]

Assume that substitution scheme (a) of T2.2 is applicable. Having substituted \( X \) for \( T \) and for \( E \) we get
\[
e(T,i-1) = e'(X,i-1)
\]
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 $S_i = \alpha.R + T$
and suppose that
$S_j = \alpha.E + F$, $i < j$,
was eliminated from $\pi$ using T2.3.
Then $e(\alpha, i) \equiv e(\alpha, j-1)$,
$e(R, i) \equiv e(R, i-1)$, $e(E, j) \equiv e(E, j-1)$, and
$e(T, i-1) \equiv e(T, i)$, $e(F, j) \equiv e(F, j-1)$.
Any reference to a following $S_j$ in $\pi$ 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 $\pi'$.
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 $\pi$ 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 $\pi$ there is an equivalent
open one $\pi'$. Moreover, $\pi'$ has an expression set identical
to $\pi$.

Proof. Observe first, that $\pi \approx \pi'$ implies that $\pi$ and $\pi'$
are equivalent (Theorem 3.4 and Lemma 3.2), and that
$\nu(\pi) = \nu(\pi')$, 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 \( \pi^n \) of Example 3.2 is reduced, but the program \( \pi \) of Example 3.2 is not.

In the following all programs are assumed proper.

We define the following sets:

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

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

\[
I_0(\pi) = \{ \pi' \in E_0(\pi) \mid \pi = \pi' \}
\]

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

**Theorem 4.2.** Let \( \pi \) and \( \pi' \) be open equivalent programs. If \( v(\pi) = v(\pi') \), then there exist programs \( \pi_0 \) and \( \pi'_0 \) such that

\[
\pi \xrightarrow{*} \pi_0 \xrightarrow{*} \pi'_0 \xrightarrow{F} \pi'.
\]

**Proof.** Apply T1 and T2 to both \( \pi \) and \( \pi' \) 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 \( \pi_0 \) and \( \pi'_0 \) are open, although not necessarily reduced.

Let \( \pi_0 \) have the statements \( S_1, S_2, \ldots, S_n \), and assume, without loss of generality, that the number of statements in \( \pi_0 \) is not less than \( n \). We will construct programs \( \pi'_0, \pi_1, \ldots, \pi_n \) such that 1) the first \( i \) statements of \( \pi_i \) are \( S_1, S_2, \ldots, S_i \); 2) \( \pi_i \) is open and 3) \( \pi_i \xrightarrow{F} \pi_{i+1} \) for \( i < n \). Clearly, \( \pi_0 \) satisfies all these requirements.

**Transformation of \( \pi_0 \) into \( \pi_{i+1} \):**

Consider the statement \( T_{i+1} \). The expression which it computes appears as a subexpression in \( v(\pi'_0) = v(\pi_i) \), for otherwise \( S_{i+1} \) is useless in \( \pi_0 \) and could have been eliminated from \( \pi'_0 \) without altering \( v(\pi') \). Since \( \pi_0 \) and \( \pi_i \) have the first \( i \) statements in common, there is a statement \( T_j \) in \( \pi_i \) computing the same expression. Here this not the case, \( S_{i+1} \) could have been eliminated from \( \pi'_0 \) as redundant without altering \( v(\pi') \) contrary to assumption.

If the variable \( X \) assigned by \( T_{i+1} \) in \( \pi'_0 \) is scalar and is assigned in \( \pi_i \), this must be done by some \( T_k \), \( k > i \), because the first \( i \) statements in \( \pi_i \) are \( S_1, S_2, \ldots, S_i \), and \( \pi'_0 \) is open. Rename \( X \) in \( \pi_i \) to be some name not yet occurring in \( \pi_i \) or \( \pi'_0 \) (T4), then apply T4 to \( \pi_i \) renaming scalar variables in \( T_j \) and elsewhere in \( \pi_i \) if necessary such that the resulting \( T'_j = S_{i+1} \). If \( S_{i+1} \) is a structured assignment, similarly rename scalar variables in \( \pi_i \) so that \( T'_j = S_{i+1} \). The resulting program \( \pi'_0 \) must be open and \( v(\pi_i) = v(\pi'_0) \). It remains to be shown that a sequence of applications of T3 can move \( T'_j \) into position \( i+1 \), thereby obtaining the program \( \pi_{i+1} \).
Case 1: \( T'_j = X + \phi Y_1 Y_2 \ldots Y_n \).

There is no problem since \( \pi_{i+1} \) is open and the \( Y_k \) are computed by \( S_1, \ldots, S_i \).

Case 2: \( T'_j = \alpha X + Y \)

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

Let \( B \vdash B.A \) be \( T_s \) for \( i < s < j \). If \( \alpha = B \), then either \( T_s \) could have been eliminated from \( \pi_0 \) without altering \( v(\pi_0) \), or there must be a corresponding statement among the \( S_{i+2}, \ldots, S_n \) in \( \pi'_0 \) computing the same expression. But that cannot be as the subexpressions for the value of \( \alpha \) 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 \( S_1, \ldots, S_i \). Hence \( X \not\equiv B \) and \( Y \not\equiv B \) so \( T_3 \) can be used for \( T_s \) and \( T'_j \). If \( T_s \), \( i < s < j \), is \( B \equiv \phi A_1 A_2 \ldots A_n \), then again \( X \not\equiv B \) and \( Y \not\equiv B \) so \( T_3 \) can be used. Therefore, \( T_3 \) can be legally applied to move \( T'_j \) to the \( i+1 \) position giving \( \pi_{i+1} \).

Case 3: \( T'_j = X + \alpha Y \)

By analogous argumentation it can be shown that \( T_3 \) can be applied to move \( T'_j \) to the \( i+1 \) position giving \( \pi_{i+1} \).

After \( n \) steps this process ends with the program \( \pi_n \). If \( \pi_n \) has more than \( n \) statements, they are now useless, so they must have been redundant in \( \pi_0 \) and could have been eliminated without altering \( v(\pi_0) \), contrary to assumption. Thus, \( \pi_n = \pi'_0 \) and, by symmetry, \( \pi_0 \vdash \pi'_0 \).  B

**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 length 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 = 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

\[ E = G(H,E), F = E', \] 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

\[ G_i.F' \]

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

\[ G_i = G(H',V') \]

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

\[ E' = G(H',E).F' \] with \( H' \equiv E' \).

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

\[ E = a(H_1,V_1)\ldots(H_n,V_n).F \]
\[ E_1 = a(H_0,V_0)(H_1,V_1)\ldots(H_n,V_n).F \]
and \( F \equiv H_j \) for some \( j \geq 1 \).

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

Now

\[ E' = a(H_1',V_1')\ldots(H_n',V_n').F' \]

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 \( G_0 = a(H_0,V_0) \) 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 \( H_i = H_0 \).

Take \( k \) to be the largest such \( i \), and consider

\[ G_k = a(H_0,V_0)\ldots(H_k,V_k). \]

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 $G' = G_k = \alpha(H_1, V_1) \ldots (H_m, V_m)$, 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

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

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 $E_1$.)

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

Theorem 4.3. Let $\pi$ and $\pi'$ be reduced and equivalent. Then $v(\pi) = v(\pi')$.

Proof. 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 $\pi$ or $\pi'$ is not reduced, contradicting the hypothesis.

Case 1. The axiom scheme and substitution has been applied.

$$a(x, y). S \equiv Y \iff x \equiv S.$$ 

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 $\alpha(E_1, V_1) \ldots (E_n, V_n).F$ with $E \equiv F$. The expression $E$ is computed by some statements in $\pi$ among which we can find

$$S_i = a(x, y)$$
$$S_j = \top = a.S$$

with $i < j$, $e(T, j) = E = e(a, i-1)(e(x, i-1), e(y, i-1)), e(S, j-1)$.

Since

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

T2.2 is applicable to $\pi$ contradicting its reducedness.

Case 2. Rule (E4)

$$a(E_1, V_1) \ldots (E_i, V_i) \ldots (E_n, V_n) \equiv a(E_1, V_1) \ldots (E_{i-1}, V_{i-1})(E_i, V_i) \ldots (E_n, V_n),$$

$E_n \equiv E_1$, was applied to $E$ in the length decreasing direction. We must be able to find statements in $S_j = a(x, y)$

$$S_k = a.T \equiv R \quad k > j$$

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

Clearly, $S_j$ is irrelevant to all references to $a$ following $S_k$. We show that it is also irrelevant to all references to $a$ between $S_j$ and $S_k$. Since the expression $H = e(a, k)$ is equal to $E$ and is equivalent to a subexpression $H'$ of $E'$ not containing a term corresponding to
there is no statement in \( \pi' \) corresponding to \( S_j \). Assume

\( S_h, j < h < k \), references \( \alpha \) and consider the expression

\( F \) computed by \( S_h \) which is a subexpression of an expression

\( G \) occurring in \( \nu(\pi) \). The corresponding expression \( G' \) in

\( \nu(\pi') \) cannot contain a term corresponding to \( F \) occurring in \( F \), since there is no statement corresponding to \( S_j \) in

\( \pi' \). 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 \( \pi \)
in \( S_h \); therefore \( S_j \) is useless, contradicting the reducedness

of \( \pi \).

Case 2: Rule (E5) \( \alpha(E_1, V_1)(E_2, V_2)\ldots(E_n, V_n).F = \alpha(E_1, V_1)\ldots(E_n, V_n).F, F \equiv E \)

was applied to \( E \) in the length decreasing direction. We

therefore can find statements in \( \pi \)

\( S_j = \alpha.X \rightarrow Y \)

\( S_h = \alpha.T \rightarrow R \)

\( S_k = Q \rightarrow \alpha.P \quad j < h < k \)

with \( e(X, j-1) = E_1, e(Y, j-1) = V_1 \)

\( e(T, h) = E_1 \equiv e(P, k-1) = F, e(R, h-1) = V_1 \).

By an argument essentially like that of case 2 it is

straight-forward to see that there will be no statement in \( \pi' \)
corresponding to \( S_j \), and that \( S_j \) is useless, contradicting

the reducedness of \( \pi \).
\[ P = T + a.A \\
= T + a.A + B \\
= a.A + B \\
X = a.D \\
= a.A + T \\
= a.C + D \\
= a.A + B \\
= a.B \\
\]

Equivalence of the two expressions is established by

\[ \alpha(A,B)(A,\alpha.A)(C,D)(A,B).B \]
\[ \equiv \alpha(A,\alpha.A)(C,D)(A,B).B \quad \text{(E4 and substitution)} \]
\[ \equiv \alpha(C,D)(A,B).B \quad \text{(E4)} \]

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 \( \pi \) is reduced and \( \pi' \in I_0(\pi) \), then \( \pi' \) is reduced.

**Proof.** It suffices to show that a single application of T3 or T4 preserves reducedness. For T4 this is obvious. Assume \( \pi \rightarrow \pi' \) and \( \pi' \) 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 \( \pi' \) is not reduced, then \( \pi \) was not reduced. The details are left to the reader. \( \square \)

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

The theorem summarizes the characterization of \( R_0(\pi) \) in terms of transformations. All equivalent reduced programs are of the 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 \( \pi \) and \( \pi' \) be programs. Then \( \pi \) is equivalent to \( \pi' \) if and only if \( \pi \frac{T}{\pi'} \).

**Proof.** " \( \iff \) " Theorem 3.4.
" \( \Rightarrow \) " Apply \( T_1 \) and \( T_2 \) to \( \pi \) and \( \pi' \) obtaining the reduced programs \( \pi_0 \) and \( \pi'_0 \). By Theorem 3.4, \( v(\pi) \equiv v(\pi_0) \equiv v(\pi') \equiv v(\pi'_0) \). By Theorem 4.5 \( \pi_0 \frac{T}{\pi'_0} \). Since \( T_3 \) and \( T_4 \) are combinations of \( T_1 \) and \( T_2 \) and their inverses (Lemmas 3.2 and 3.3), the theorem is proved. \( \square \)

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 \( \pi \) 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.

**Corollary 4.7.** Let \( \pi \) and \( \pi' \) be programs. Then \( \pi \) is equivalent to \( \pi' \) if and only if

\[
\begin{align*}
\pi & \overset{*}{\rightarrow} \pi_0 \\
& \overset{*}{\rightarrow} \pi'_0 \\
& \overset{*}{\rightarrow} \pi'.
\end{align*}
\]

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 \( \pi \), a program \( \pi_0 \), equivalent to \( \pi \), 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 \( \pi_0 \). We show how such algorithms optimizing with respect to a large class of cost functions can be structured in terms of the code transformations \( T_1, T_2, T_3 \) and \( T_4 \).

**Definition.** A function \( C \) mapping programs \( \pi \) into a nonnegative number \( C(\pi) \) is said to be a **sequential cost function** iff

(C1) for any programs \( \pi \) and \( \pi' \), if \( \pi \overset{*}{\rightarrow} \pi' \), then \( C(\pi') \leq C(\pi) \).

(C2) given any program \( \pi \) 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 \( \pi_0 \) for which cost is minimal. Because of the above definition this search can be restricted to the class \( R_0(\pi) \).

**Lemma 5.1.** Let \( \pi \) be a program. There is a program \( \pi' \) in \( E_0(\pi) \) and a program \( \pi'' \) in \( R_0(\pi) \) such that \( \pi \xrightarrow{T_2} \pi' \xrightarrow{T_1} \pi'' \).

**Proof.** See Appendix.

A modification of the proof of the lemma can be used to define an algorithm finding \( \pi' \) and \( \pi'' \). The lemma shows that a reduced program can always be found by applications of \( T_2 \) followed by applications of \( T_1 \).

**Lemma 5.2.** Let \( \pi \) be open and reduced, and suppose \( \pi \xrightarrow{T_4} \pi_1 \xrightarrow{T_3} \pi'' \). Then there is a program \( \pi_2 \) such that \( \pi \xrightarrow{T_2} \pi_2 \xrightarrow{T_1} \pi'' \).

**Proof.** See Appendix.

The lemma states a restricted kind of commutativity which exists for \( T_3 \) and \( T_4 \) if \( \pi \) is open and reduced.

There is an obvious

**Corollary 5.3.** Let \( \pi \) be open and reduced, and \( \pi' \in R_0(\pi) \). Then there is a program \( \pi'' \) such that \( \pi \xrightarrow{T_3} \pi'' \xrightarrow{T_4} \pi' \).

**Proof.** 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 \( \pi \xrightarrow{T_3} \pi_1 \xrightarrow{T_4} \pi' \) but no reduced program \( \pi_2 \) such that \( \pi \xrightarrow{T_4} \pi_2 \xrightarrow{T_3} \pi' \), and similarly if \( T_4 \) has been applied first.

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

**Theorem 5.4 (Canonical Form Theorem).** Let \( C \) be a sequential cost function. There is an algorithm which optimizes any program \( \pi \) with respect to \( C \) by applying first the operation \( \xi \) and then the operation \( \chi \) to \( \pi \).

Furthermore, \( \xi \) is independent of \( C \) and can be written as \( \xi = \xrightarrow{T_2} \xrightarrow{T_1} \), and \( \chi \) can be written as \( \chi = \xrightarrow{T_3} \xrightarrow{T_4} \).

**Proof.** Let \( \pi \) be a program. Using Lemma 5.1, a reduced program \( \pi_1 \) can be found by applying \( T_2 \) repeatedly, then \( T_1 \) repeatedly. I.e.,

\[ \pi \xrightarrow{T_2} \pi_1 \]

where \( \pi_1 \in R_0(\pi) \). Next use \( T_4 \) to get an open program \( \pi_2 \).

\[ \pi \xrightarrow{T_2} \pi \xrightarrow{T_1} \pi_1 \xrightarrow{T_4} \pi_2 \]

Now \( \pi_2 \in R_0(\pi) \) and \( \pi_2 \) 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 \( \pi \xrightarrow{T_3} \pi \xrightarrow{T_4} \). 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 π₁ equivalent to π. Having thus satisfied the hypotheses of Corollary 5.3, the operation χ then transforms π₁ 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(π) = 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 π₀ 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 · 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 E₁(π) = {π'|π and π' compute the same output values for all input assignments), assuming specific interpretations of the operators φ ∈ A. If so, what would be a minimal complete axiom set?
Lemma 5.1. Let \( \pi \) be a legal program. There is a program \( \pi_1 \) in \( E_0(\pi) \) and a program \( \pi_2 \) in \( R_0(\pi) \) such that
\[
\pi \rightarrow^* \pi_1 \rightarrow^* \pi_2.
\]

Proof. Apply T2 as long as possible to \( \pi \) yielding a program \( \pi_1 \), and then apply T1 as long as possible to \( \pi_1 \) yielding a program \( \pi_2 \). We need to show that \( \pi_2 \) is reduced. Since T1 is not applicable to \( \pi_2 \) it suffices to show that T2 cannot be applied.

Since \( \pi_1 \rightarrow^* \pi_2 \), to each statement in \( \pi_2 \) there is a corresponding one in \( \pi_1 \) computing an equivalent sub-expression of an expression occurring in \( \nu(\pi_1) \). 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 \( \pi_1 \) and \( \pi_2 \), respectively.

Case 1: T2.1 can be applied to \( \pi_2 \):
We can find statements \( S_i \) and \( S_j \) in \( \pi_2 \) assigning the scalar variables names \( T \) and \( \tilde{T} \), respectively, with
\[
e_2(T,i) \equiv e_2(\tilde{T},j)
\]
and can find corresponding statements \( S_i' \) and \( S_j' \) in \( \pi_1 \) assigning \( T \) and \( \tilde{T} \), and
\[
e_1(T,i') \equiv e_2(T,i),
\]
\[
e_1(\tilde{T},j') \equiv e_2(\tilde{T},j).
\]
But then \( e_1(T,i') \equiv e_1(\tilde{T},j) \), i.e. T2.1, is applicable to \( \pi_1 \), contrary to assumption.

Case 2: T2.2 can be applied to \( \pi_2 \):
By similar reasoning, there are statements \( S_i \) and \( S_j \) in \( \pi_2 \) and \( S_i' \) and \( S_j' \) in \( \pi_1 \) with
\[
S_i = S_i', = \alpha.\text{true}
\]
\[
S_j = S_j', = T = \alpha.R
\]
and
\[
e_2(\alpha,i) \equiv e_2(\alpha,j)\hspace{1cm}(\text{applicability of T2.2})
\]
\[
e_2(X,i) \equiv e_2(R,j-1)
\]
\[
e_2(X,i') \equiv e_2(\alpha,i')
\]
\[
e_2(\alpha,i) \equiv e_1(\alpha,i')\hspace{1cm}(\text{since } \pi_1 \rightarrow^* \pi_2)
\]
\[
e_2(\alpha,j) \equiv e_1(\alpha,j')
\]
\[
e_2(R,j-1) \equiv e_1(R,j'-1)
\]
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 \( \pi_1 \), contrary to assumption.

Case 3: T2.3 can be applied to \( \pi_2 \):
By reasoning analogous to case 2, then T2.3 is also applicable to \( \pi_1 \), contrary to assumption. \( \blacksquare \)
Lemma 5.2: Let \( \pi \) be open and reduced. If there are
programs \( \pi_1 \) and \( \pi' \) such that \( \pi \xrightarrow{T4} \pi_1 \xrightarrow{T3} \pi' \),
then there is a program \( \pi_2 \) such that
\[ \pi \xrightarrow{T3} \pi_2 \xrightarrow{T4} \pi' . \]

Proof. Consider \( \pi_1 \): 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 \( \pi \), thus obtaining \( \pi_2 \), for then clearly
\[ \pi \xrightarrow{T4} \pi' \]
REFERENCES


