STANFORD ARTIFICIAL INTELLIGENCE
MEMO NO. 48

July 19, 1967

CORRECTNESS OF A COMPILER FOR ALGOL-LIKE PROGRAMS

by Donald M. Kaplan

ABSTRACT: A compiling algorithm is given which maps a class of Algol-like programs into a class of machine language programs. The semantics, i.e., the effect of execution, of each class if specified, and recursion induction used to prove that program semantics is preserved under the mapping defined by the compiling algorithm.

The research reported here was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-183).

CORRECTNESS OF A COMPILER FOR ALGOL-LIKE PROGRAMS

by Donald M. Kaplan

1. Introduction

This paper contains a proof of correctness for a simple compiling algorithm which compiles Algol-like programs into machine language.

The tools for constructing such a proof have already been developed to some considerable degree. The concepts of abstract syntax, state vector, the use of an interpreter for defining the semantics of a programming language, and the definition of correctness of a compiler are all basically the same as in [3]. Furthermore, we use the formalisms and methods developed in [2] as a framework for deriving the main body of our results. The ultimate goal, as outlined in [1], [2] and [3], is to submit any proof that a compiler is correct to proof-checking on a computer; thus, the completeness results in [4] are also relevant here. In [5], a proof is given for an algorithm which compiles arithmetic expressions; in [6] some results are obtained for an Algol-like language, and in fact much of the material here was first described in [6] even though it is superceded by the present paper. The results presented here were obtained independently of those in [7], and in fact this paper was completed in its present form in September, 1966.

The method of proof is essentially very simple. First, the abstract analytic syntax for the source language is given. Second, the abstract synthetic syntax for the object (machine) language is given. Then, a compiling algorithm is specified which will analyze a representative source program and synthesize an object program from it.

The semantics of the source language is then specified by an interpreter which operates using the analytic syntax for source programs and which defines the effect on the program state vector of executing a source program. The abstract analytic syntax for the object language is then given, and the semantics of the object language is specified in a similar manner.

We then show that the compiling algorithm is correct. First, using the semantics, we interpretively execute a representative source program and the object program compiled from it. Second, for each variable in the source program and corresponding variable in the object program, we prove either that the components for these variables in the respective state vectors have the same value when execution of the programs is completed, or that neither program completes execution.

2. Source Language - Abstract Analytic Syntax

The abstract analytic syntax for the source language is given by the following table:

Predicate	Associated Functions	
issource[γ]	varlist[γ]	
isassignment[s]	leftside[s]	rightside[s]
isconditional[s]	premise[s]	
istransfer[s]	destination[s]	
issum[e]	s1[e]	s2[e]
isconst[e]		
isvar[e]		
isgreater[b]	left[b]	right[b]
isand[b]	b1[b]	b2[b]
isnot[b]	b3[b]	

This table is to be interpreted as follows. If isprogram [γ], then varlist[γ] is a list of all variable names appearing in the program γ, which is itself a list of statements.

There are three types of statements. If isassignment[s], then s is an Algol-like assignment statement where the variable leftside[s] is assigned the arithmetic expression rightside[s]. If isconditional[s], then we have an Algol-like conditional statement s, where the result depends on the truth or falsity of the boolean expression premise[s]. If istransfer[s], then s is an Algol-like transfer statement with an integer pointer destination[s] to a statement in γ.

There are three types of arithmetic expressions. If issum[e], then e is the sum of the arithmetic expressions s1[e] and s2[e]. For simplicity, sums only are considered; inclusion of any other binary arithmetic operation would not require any essential change in our method of proof. If isconst[e] then e is a constant, and if isvar[e] then e is a variable.

There are three types of boolean expressions. If isgreater[b], then b is the relation left[b] greater-than right[b], where left[b] and right[b] are arithmetic expressions. If isand[b], then b is the logical product of the boolean expressions b1[b] and b2[b]. If isnot[b], then b is the logical negation of the boolean expression b3[b]. Again for simplicity, logical sums and relations other than greater-than have been excluded.

Note that because we use abstract syntax, no commitment is made to a particular form for any of the constructs in the source language. Arithmetic and boolean expressions are governed by an axiom of regularity which prevents infinite descending nests of expressions and allows a simple induction principle to be applied to these expressions.

Since γ and varlist[γ] are lists, we will need certain primitive functions for manipulating lists. If a list h is not null, then first[h] gives the first element of that list, and rest[h] gives the list of remaining elements. In addition, we have the predicate null[h], which is true if and only if h is the null list. To build up lists from other lists we have the associative concatenation operation *, so that $h_1 * h_2$ is the list resulting from appending the list h_2 onto the end of list h_1. These functions and the concatenation operation are related by the following:

(2.1) $\neg$ null[h] $\supset$ h = first[h] * rest[h]

(2.2) $\neg$ null[h_1] $\wedge$ null[rest[h_1]] $\supset$ h_1 = first[h_1*h_2]

(2.3) null[h_1*h_2] $\Leftrightarrow$ null[h_1] $\wedge$ null[h_2]

Three other functions which will be useful are:

(2.4) select[n,p] = if null[p] then p else if n=1 then first[p]
else select[n-1,rest[p]]

which selects the n-th element out of a list p.

(2.5) $\underline{\text{length}}[p] = \underline{\text{if}}\ \underline{\text{null}}[p]\ \underline{\text{then}}\ 0\ \underline{\text{else}}\ 1+\underline{\text{length}}[\underline{\text{rest}}[p]]$

which gives the number of elements in the list p.

(2.6) $\underline{\text{sublist}}[n,p] = \underline{\text{if}}\ \underline{\text{null}}[p]\ \underline{\text{then}}\ p\ \underline{\text{else}}\ \underline{\text{if}}\ n=1\ \underline{\text{then}}\ \underline{\text{first}}[p]$
$\underline{\text{else}}\ \underline{\text{first}}[p]*\underline{\text{sublist}}[n-1,\underline{\text{rest}}[p]]$

which gives the sublist of p consisting of the first n elements of p. If n is not a positive integer less than or equal to the number of elements in p, then p itself is returned.

It is straightforward to show by recursion induction that

(2.7) $\underline{\text{length}}[p*q] = \underline{\text{length}}[p] + \underline{\text{length}}[q]$

(2.8) $\underline{\text{select}}[\underline{\text{length}}[p]+1,p*q] = \underline{\text{first}}[q]$

(2.9) $r = p*q \wedge \underline{\text{length}}[p] = k \supset p = \underline{\text{sublist}}[k,r]$

Note that we do not have to give a synthetic syntax for the source language since both the interpreter which defines the source language semantics and the compiler use only the analytic syntax.

3. Source Language - Semantics

The state vector for a source program γ contains one element for each of the variables in varlist[γ] plus one more element for the implicit variable α, which acts as a program counter for sequencing statement execution.

There are two functions of state vectors introduced in [3], namely:

$c(v,\xi)$ denoting the value of the element for variable v in the state vector ξ, and

$a(v,k,\xi)$ denoting the state vector obtained from the state vector ξ by changing the value of the element for variable v to k and leaving ξ otherwise unchanged.

These functions satisfy the following complete (see [4]) set of axioms:

(3.1) $c(x,a(y,k,\xi)) = \underline{\text{if}}\ x=y\ \underline{\text{then}}\ k\ \underline{\text{else}}\ c(x,\xi)$

(3.2) $a(x,k,a(y,l,\xi)) \equiv$ if x=y then $a(x,k,\xi)$ else $a(y,l,a(x,k,\xi))$

(3.3) $a(x,c(x,\xi),\xi) \equiv \xi$

Note that $\equiv$ is used to indicate equality of state vectors.

We let ξ_γ be the state vector for a program γ before any execution of γ has taken place, and we stipululate that

(3.4) $c(\alpha,\xi_\gamma) = 1$ (i.e., one)

(3.5) $c(v,\xi\gamma) = v_o$ for all variables v in varlist[γ].

These initial conditions state simply that execution starts with the first statement in γ and that the variables are initialized to some specified set of values.

We now define the semantics of the source language by defining the state vector that results when an arbitrary source program γ is executed. This resultant state vector is, in fact, sourceoutcome[γ,ξ_γ], where

(3.6) sourceoutcome[p,ξ] $\equiv$ execsource[p,1,length[p],ξ]

(3.7) execsource[p,k,n,ξ] $\equiv$ if $c(\alpha,\xi) < k \vee c(\alpha,\xi) \geq k+n$ then ξ
else execsource[p,k,n,stepsource[select[$c(\alpha,\xi)$,p],ξ]]

(3.8) stepsource[s,ξ] $\equiv$ if isassignment[s] then
$a(\alpha,1+c(\alpha,\xi),a(\text{leftside[s]},\text{value[rightside[s]},\xi],\xi))$
else if isconditional[s] then
$a(\alpha,[\text{if bvalue[premise[s]},\xi] \text{ then } 1 \text{ else } 2]+c(\alpha,\xi),\xi)$
else if istransfer[s] then $a(\alpha,\text{destination[s]},\xi)$

(3.9) value[e,ξ] = if isvar[e] then $c(e,\xi)$
else if isconst[e] then val[e]
else if issum[e] then value[s1[e],ξ] & value[s2[e],ξ]

(3.10) bvalue[b,ξ] = if isgreater[b] then
value[left[b],ξ] > value[right[b],ξ]
else if isand[b] then
[if bvalue[b1[b],ξ] then bvalue[b2[b],ξ] else false]
else if isnot[b] then
[if bvalue[b3[b],ξ] then false else true]

Remarks:

- In (3.9), val[e] gives the numerical value of an expression e representing a constant.
- The operation & in (3.9) is meant to be like addition of real numbers (i.e., like + in (3.8)), but our results do not depend on this.
- In (3.8), destination[s] gives an integer numerical value.
- In (3.7), execution of program p is restricted to that sublist of p starting at the k-th statement and extending for n statements. We denote this sublist as (k,n). If α the program counter is set to point outside this sublist, then execution terminates. This feature is used later on to execute isolated statements in a program.
- In (3.6), the sublist executed is p itself.

4. Object Language-Abstract Analytic Syntax

The abstract analytic syntax for the object language is given by the following table:

Predicate	Associated Functions
isli[t]	arg[t]
isload[t]	adr[t]
issto[t]	adr[t]
isadd[t]	adr[t]
isgth[t]	adr[t]
istra[t]	dest[t]
istmi[t]	dest[t]
ischs[t]	

There are eight types of machine instructions, and each is thought of as having two parts. The first part of an instruction indicates in some way what type of an instruction it is. The second part, given by the associated function in the above table, is a numerical value in case isli[t], the integer address of a data word in case isload[t]∨issto[t]∨isadd[t]∨isgth[t] or an integer pointer to another instruction in case istra[t]∨istmi[t]. In case ischs[t], the second part is not used.

An object program is then a list of instructions and, as such, can be processed using the list handling functions and concatenation operation described earlier.

5. Object Language - Abstract Synthetic Syntax

The synthetic syntax is used to construct object programs. There is a function for generating each of the eight types of instruction with the appropriate first and second parts. Programs are built up using the concatenation operation on lists. The synthetic syntax functions are given in the following table:

Predicate	Associated Functions
isli[t]	mkli[k]
isload[t]	mkload[x]
issto[t]	mksto[x]
isadd[t]	mkadd[x]
isgth[t]	mkgth[x]
istra[t]	mktra[n]
istmi[t]	mktmi[n]
ischs[t]	mkchs[ ]

The synthetic and analytic syntaxes for the object language are related by the following:

(5.1)	isli[mkli[k]]	k = arg[mkli[k]]
(5.2)	isload[mkload[x]]	x = adr[mkload[x]]
(5.3)	issto[mksto[x]]	x = adr[mksto[x]]
(5.4)	isadd[mkadd[x]]	x = adr[mkadd[x]]
(5.5)	isgth[mkgth[x]]	x = adr[mkgth[x]]
(5.6)	istra[mktra[n]]	n = dest[mktra[n]]
(5.7)	istmi[mktmi[n]]	n = dest[mktmi[n]]
(5.8)	ischs[mkchs[ ]]	

Note that because we use abstract syntax, no commitment is made to a particular form for machine instructions.

6. Object Language - Semantics

A machine is thought of as having a program part (i.e., the list of instructions) and a data part consisting of registers addressed by positive integers.

The state vector for a machine contains one element for each register in the data part, and we also include in the state vector an element for the accumulator register denoted by β and an element for the instruction counter register denoted by α.

The two functions of state vectors are here defined as

$c(x,\theta)$ denoting the value of the element for register x in the state vector θ, where x is α,β or an integer, and

$a(x,k,\theta)$ denoting the state vector obtained from the state vector θ by changing the value of the element for register x to k and leaving θ otherwise unchanged.

The axioms governing these functions are given in (3.1), (3.2) and (3.3).

We let θ_ω be the state vector for an object program ω before any execution of ω has taken place, and we stipulate that

(6.1) $c(\alpha,\theta_\omega) = 1$ (i.e., one)

(6.2) $c(\beta,\theta_\omega) = \beta_o$

(6.3) $c(i,\theta_\omega) = i_o$ for all data registers i in the machine.

These initial conditions state simply that execution starts with the first instruction in ω, and that the accumulator and the data registers are initialized to some specified set of values.

We now define the semantics of the object language by defining the state vector that results when an arbitrary object program ω is executed. This resultant state vector is, in fact, $\text{objectoutcome}[\omega,\theta_\omega]$, where

(6.4) $\text{objectoutcome}[p,\theta] \equiv \text{execobject}[p,1,\underline{\text{length}}[p],\theta]$

(6.5) $\text{execobject}[p,k,n,\theta] \equiv \underline{\text{if}}\ c(\alpha,\theta) < k \vee c(\alpha,\theta) \geq k+n\ \underline{\text{then}}\ \theta$
$\underline{\text{else}}\ \text{execobject}[p,k,n,\text{stepobject}[\underline{\text{select}}[c(\alpha,\theta),p],\theta]]$

(6.6) $\text{stepobject}[t,\theta] \equiv \underline{\text{if}}\ \text{istra}[t]\ \underline{\text{then}}\ a(\alpha,\text{dest}[t],\theta)$
$\underline{\text{else if}}\ \text{istmi}[t]\ \underline{\text{then}}$
$a(\alpha,[\underline{\text{if}}\ c(\beta,\theta) < 0\ \underline{\text{then}}\ \text{dest}[t]\ \underline{\text{else}}\ 1+c(\alpha,\theta)],\theta)$

```
        else a(α,1+c(α,θ),
    [if isli[t] then a(β,arg[t],θ)
    else if isload[t] then a(β,c(adr[t],θ),θ)
    else if issto[t] then a(adr[t],c(β,θ),θ)
    else if isadd[t] then a(β,c(β,θ) & c(adr[t],θ),θ)
    else if isgth[t] then
        a(β,[if c(β,θ) > c(adr[t],θ) then 1 else -1],θ)
    else if ischs[t] then a(β,-c(β,θ),θ)])
```

Remarks:

- Giving the semantics of instructions in the form of state vector expressions as in (6.6) is akin to specifying micro-programs for the central processor of a computer.
- In (6.5), execution of program p is restricted to that sublist of p starting at the k-th instruction and extending for n instructions. We denote this sublist by (k,n). If α the instruction counter is set to point outside this sublist, then execution terminates. This feature is used later on for executing isolated sections of object program.
- In (6.4), the sublist executed is p itself.
- In (6.6), the operation & is the same as that in (3.9), i.e., like addition of real numbers.

7. The Compiler

The compiler is a function which accepts a source program γ as an argument and which has as its value the corresponding object program compile[γ], where

```
(7.1)   compile[p] = comprog[p,1]
(7.2)   comprog[p,k] = if null[p] then p
            else compstate[first[p],k] * comprog[rest[p],k+1]
(7.3)   compstate[s,k] =
            if isassignment[s] then
                comparith[rightside[s],t[γ]] * mksto[loc[leftside[s]]]
            else if isconditional[s] then
                compbool[premise[s],f[k]] * mktmi[f[k+2]]]
            else if istransfer[s] then mktra[f[destination[s]]]
```

(7.4) comparith[e,i] = if isconst[e] then mkli[val[e]]
else if isvar[e] then mkload[loc[e]]
else if issum[e] then
comparith[s2[e],i]*mksto[i]*comparith[s1[e],i+1]*mkadd[i]

(7.5) compbool[b,n] = if isgreater[b] then
comparith[right[b],t[γ]] * mksto[t[γ]] *
comparith[left[b], t[γ]+1] * mkgth[t]
else if isand[b] then
compbool[b1[b],n] * mktmi[n+numbool[b]] *
compbool[b2[b],n+numbool[b1[b]]+1]
else if isnot[b] then compbool[b3[b],n] * mkchs[ ]

(7.6) t[p] = 1 + length varlist[p]

(7.7) loc[v] = locl[v,varlist[γ]]
locl[v,p] = if v=first[p] then 1 else 1+locl[v,rest[p]]

(7.8) f[k] = if k=1 then 1 else 1+fl[sublist[k-1,γ]]
fl[t] = if null[t] then 0 else numstate [first[t]]+fl[rest[t]]

(7.9) numstate[s] = if isassignment[s] then 1+numarith[rightside[s]]
else if isconditional[s] then 1+numbool[premise[s]]
else if istransfer[s] then 1

(7.10) numarith[e] = if isconst[e] ∨ isvar[e] then 1
else if issum[e] then 2+numarith[s1[e]]+numarith[s2[e]]

(7.11) numbool[b] = if isgreater[b] then
2+numarith[left[b]]+numarith[right[b]]
else if isand[b] then 1+numbool[b1[b]]+numbool[b2[b]]
else if isnot[b] then 1+numbool[b3[b]]

Remarks:

- In (7.1) through (7.11), γ is assumed to be a free variable.
- In (7.3) and (7.5), we adopt the convention that +1 and -1 represent truth and falsity respectively.
- The position of a variable in varlist[γ] determines the address loc[v] of the corresponding data register in the machine.
- The function t[γ] gives the address of the first data register available for temporary storage.

- If s is the n-th statement in γ, then f[n] gives the position in compile[γ] of the first instruction in compstate[s,n].
- In (7.5), the definition for compbool[b,n], the integer n is the position in the list compile[γ] of the first instruction of compbool[b,n].
- The function numstate[s] gives the number of instructions in compstate[s,n].

8. Correctness of the Compiler

The correctness of the compiler is stated in the following

(8.1) Theorem: For any source program γ and corresponding object program ω = compile[γ], where $c(v,\xi_\gamma) = c(loc[v],\theta_\omega)$, we have

$$c(v, sourceoutcome[\gamma,\xi_\gamma]) = c(loc[v], objectoutcome[\omega,\theta_\omega])$$

for all variables v in varlist[γ].

The remainder of this paper deals at length with the proof of this theorem; recursion induction is the method used. The reader is encouraged to pass over the details on first reading so that the main points will not become obscured. Justification for many of the proof steps is specified in the right hand margin; however, only references within this paper are given. A full explanation of the axioms of conditional expressions as utilized here is given in [3]. Several lemmas are mentioned but not proved; proofs are easily constructed, but do not bear inclusion here.

In fact, before proceeding, we state without proof the following three lemmas.

(8.2) Lemma: $execobject[\omega,k_1,n_1,\theta] \equiv$
$execobject[\omega,k_1,n_1,execobject[\omega,k_2,m_2,\theta]]$
where sublist (k_2,n_2) is itself a sublist of sublist (k_1,n_1).

(8.3) Lemma: $execobject[\omega,k_1,n_1,execobject[\omega,k_2,n_2,\theta]] \equiv$
$execobject[\omega,k_1,n_1,execobject[\omega,k_3,n_3,\theta]]$
where sublists (k_2,n_2) and (k_3,n_3) are sublists of sublist (k_1,n_1).

(8.4) Lemma: execsource[γ, k_1, n_1, ξ] ≡
execsource[γ, k_1, n_1, execsource[γ, k_2, n_2, ξ]]
where sublist (k_2, n_2) is itself a sublist of sublist (k_1, n_1).

We now proceed with the proof of (8.1).

Let ξ be a program state vector and θ a machine state vector such that

(8.5) (i) f[c(α, ξ)] = c(α, θ)
(ii) c(v, ξ) = c(loc[v], θ) for all variables v in varlist[γ].

Let sourceoutcome[γ, ξ] ≡ execsource[γ, 1, length[γ], ξ] ≡ X. (3.6)

Then,

X ≡ if c(α, ξ) < 1 ∨ c(α, ξ) ≥ length[γ]+1 then ξ else
execsource[γ, 1, length[γ], stepsource[select[c(α, ξ), γ], ξ]] (3.7)
≡ if c(α, ξ) < 1 ∨ c(α, ξ) ≥ length[γ]+1 then ξ else
execsource[γ, 1, length[γ], ψ] where (3.7)
ψ ≡ execsource[γ, c(α, ξ), 1, ξ] .

So that,

(8.6) c(v, X) = if c(α, ξ) < 1 ∨ c(α, ξ) ≥ length[γ]+1 then c(v, ξ) else
c(v, execsource[γ, 1, length[γ], ψ]) for all v in varlist[γ]

Now let objectoutcome[ω, θ] ≡ execobject[ω, 1, length[ω], θ] ≡ Y. (6.4)

Then,

Y ≡ if c(α, θ) < 1 ∨ c(α, θ) ≥ length[ω]+1 then θ else
execobject[ω, 1, length[ω], stepobject[select[c(α, θ), ω], θ]] (6.5)
≡ if c(α, θ) < 1 ∨ c(α, θ) ≥ length[ω]+1 then θ else
execobject[ω, 1, length[ω], φ] where (8.3) (6.5)
φ ≡ execobject[ω, c(α, θ), n, θ] and
n = length[compstate[select[c(α, ξ), γ], c(α, ξ)]] .

So that,

c(loc[v], Y) = if c(α, θ) < 1 ∨ c(α, θ) ≥ length[ω]+1 then c(loc[v], θ)
else c(loc[v], execobject[ω, 1, length[ω], φ])
= if f[c(α, ξ)] < 1 ∨ f[c(α, ξ)] ≥ length[ω]+1 then c(v, ξ) else
c(loc[v], execobject[ω, 1, length[ω], φ]) . (8.5)

We now make use of the following

(8.7) Lemma: $[k<1 \vee k \geq \underline{length}[\gamma]+1] \Leftrightarrow [f[k] < 1 \vee f[k] \geq \underline{length}[\omega]+1]$

In effect, this lemma states that if and when the source program γ terminates execution, so does the object program ω = compile[γ].

We do not give a proof, but only remark that (7.8) shows that if $k < 1$ or $k \geq \underline{length}[\gamma]+1$, then $\underline{sublist}[k-1,\gamma] = \gamma$, and so $f[k] = \underline{length}[\omega]+1$. Similarly, we see that $\neg f[k] < 1$ and that $\neg f[k] \leq \underline{length}[\omega]+1$ only if $k < 1$ or $k \geq \underline{length}[\gamma]+1$. More discussion of (7.8) appears in Section 9 of this paper.

Then,

(8.8) $c(loc[v],Y) = \underline{if}\ c(\alpha,\xi) < 1 \vee c(\alpha,\xi) \geq \underline{length}[\gamma]+1\ \underline{then}\ c(v,\xi)$
$\underline{else}\ c(loc[v],execobject[\omega,1,\underline{length}[\omega],\varphi)$ (8.7)
for all variables v in varlist [γ].

Comparison of (8.6) and (8.8) shows that they both have the form

$c(f_1[v],f_2[f_3[\gamma],1,\underline{length}[f_3[\gamma]],f_4])$
$= \underline{if}\ c(\alpha,\xi) < 1 \vee c(\alpha,\xi) \geq \underline{length}[\gamma]+1\ \underline{then}\ c(v,\xi)\ \underline{else}$
$c(f_1[v],f_2[f_3[\gamma],1,\underline{length}[f_3[\gamma]],f_5])$

where we have the following table of correspondences:

	(8.6)	(8.8)
$f_1[v]$	v	loc[v]
$f_2[\ldots]$	execsource[...]	execobject[...]
$f_3[\gamma]$	γ	compile[γ]
f_4	ξ	θ
f_5	ψ	φ

We need only show that the conditions in (8.5) on ξ and θ also hold for ψ and φ to make the recursion induction complete. This requirement is given by a theorem which we state here and prove in Section 9.

Note that in the case of non-terminating γ or ω, (8.7) gives us that both (8.6) and (8.8) are undefined.

(8.9) Theorem: Using the definitions for ψ and φ,

(i) $f[c(\alpha, \text{execsource}[\gamma, c(\alpha,\xi),1,\xi])]$
$= c(\alpha, \text{execobject}[\omega, c(\alpha,\theta),n,\theta])$

(ii) $c(v, \text{execsource}[\gamma, c(\alpha,\xi),1,\xi])$
$= c(\text{loc}[v], \text{execobject}[\omega, c(\alpha,\theta),n,\theta])$ for all variables v in $\text{varlist}[\gamma]$.

This is the key theorem in proving compiler correctness. It states that a statement and the instructions compiled for it each affect the variables and the flow of program control in a like manner.

Given theorem (8.9), then, we have by recursion induction that $c(v, \text{sourceoutcome}[\gamma,\xi]) = c(\text{loc}[v], \text{objectoutcome}[\omega,\theta])$ for all variables v in $\text{varlist}[\gamma]$, provided that ξ and θ satisfy the conditions in (8.5); note that we have used the defined meanings of X and Y in stating this result.

Now consider ξ_γ and θ_ω and note that

(i) $f[c(\alpha,\xi_\gamma)] = f[1] = 1 = c(\alpha,\theta_\omega)$ (3.4) (7.8) (6.1)

(ii) $c(v,\xi_\gamma) = c(\text{loc}[v],\theta_\omega)$ by the hypothesis for (8.1)

for all variables v in $\text{varlist}[\gamma]$. But these are precisely the conditions given in (8.5). Thus,

$c(v, \text{sourceoutcome}[\gamma,\xi_\gamma]) = c(\text{loc}[v], \text{objectoutcome}[\omega,\theta_\omega])$ for all variables v in $\text{varlist}[\gamma]$.

Hence, we have a proof of (8.1), correctness of the compiler, provided that we can prove (8.9). The proof of this theorem is the subject of the next section.

9. Compiler Correctness for a Single Statement - Theorem (8.9)

Before proceeding, we need a notion of partial equality for state vectors. The following description parallels that given in [5]. The notation $\delta_1 \equiv_A \delta_2$ where δ_1 and δ_2 are state vectors and A is a set of variables (or registers) means that the values of corresponding elements of δ_1 and δ_2 are equal except possibly for elements corresponding to variables in A. Symbolically, we write $x \notin A \supset c(x,\delta_1) = c(x,\delta_2)$. Partial equality satisfies the following relations:

(9.1) $\delta_1 \equiv \delta_2$ is equivalent to $\delta_1 \equiv_{\{\}} \delta_2$ where $\{\}$ denotes the empty set.

(9.2) If $A \subset B$ and $\delta_1 \equiv_A \delta_2$ then $\delta_1 \equiv_B \delta_2$

(9.3) If $\delta_1 \equiv_A \delta_2$ then $a(x,k,\delta_1) \equiv_{A-\{x\}} a(x,k,\delta_2)$

(9.4) If $x \in A$ then $a(x,k,\delta) \equiv_A \delta$

(9.5) If $\delta_1 \equiv_A \delta_2$ and $\delta_2 \equiv_B \delta_3$ then $\delta_1 \equiv_{A \cup B} \delta_3$

In the case of machine state vectors, we will make use of the following specialization of this notation:

$\theta_1 \equiv_k \theta_2$ denotes $\theta_1 \equiv_{\{x\,:\,x \geq k\}} \theta_2$

$\theta_1 \equiv_\beta \theta_2$ denotes $\theta_1 \equiv_{\{\beta\}} \theta_2$

$\theta_1 \equiv_{\beta,k} \theta_2$ denotes $\theta_1 \equiv_{\{x\,:\,x=\beta \vee x \geq k\}} \theta_2$

Let us now restate in detail the theorem we are going to prove.

Theorem: For any source program γ and object program ω = compile[γ], if ξ and θ are state vectors such that

(i) $f[c(\alpha,\xi)] = c(\alpha,\theta)$ where $1 \leq c(\alpha,\xi) <$ length[γ]+1, and

(ii) $c(v,\xi) = c(loc[v],\theta)$ for all variables v in varlist[γ],

then,

(i) $f[c(\alpha,$execsource$[\gamma,c(\alpha,\xi),1,\xi])]$
$= c(\alpha,$execobject$[\omega,c(\alpha,\theta),n,\theta])$ and

(ii) $c(v,$execsource$[\gamma,c(\alpha,\xi),1,\xi])$
$= c(loc[v],$execobject$[\omega,c(\alpha,\theta),n,\theta])$ for all variables v in varlist[γ], where

n = length[compstate[select[$c(\alpha,\xi),\gamma$],$c(\alpha,\xi)$]]] .

The intention, of course, is that the sublist $(c(\alpha,\theta),n)$ of ω should be the code compiled for the statement select[$c(\alpha,\xi),\gamma$], i.e., we want

$$(c(\alpha,\theta),n) = \text{compstate}[\text{select}[c(\alpha,\xi),\gamma],c(\alpha,\xi)] .$$

Certainly the length of $(c(\alpha,\theta),n)$, i.e. n, is the correct value, but we must still prove the following

(9.6) Lemma:

select[$c(\alpha,\theta),\omega$] = first[compstate[select[$c(\alpha,\xi),\gamma$],$c(\alpha,\xi)$]] .

Since we are given that $c(\alpha,\theta) = f[c,\alpha,\xi)]$, proof of this lemma in fact constitutes a validation of the lookup-like procedure performed by the compiler functions f[k] and fl[t] defined in (7.8). We give an outline only of a proof for this lemma.

Let s = select[$c(\alpha,\xi),\gamma$] and γ = p*s*q
where length[p] = $c(\alpha,\xi)$-1. (2.8)

Now we state without proof the following simple

(9.7) Lemma: If r = u*v, then
comprog[r,1] = comprog[u,1] * comprog[v,length[u]+1].

From this lemma, we can easily deduce that since
ω = compile[γ] = comprog[γ,1], then

(9.8) ω = comprog[p,1]*compstate[s,$c(\alpha,\xi)$]*comprog[q,$c(\alpha,\xi)$+1] .

Now consider the function definition

lengthh[comprog[p,1]]
= length[if null[p] then p else compstate[first[p],1]
*comprog[rest[p],2]] (7.2)
= if null[p] then 0 else length[compstate[first[p],1]]
+length[comprog[rest[p],2]] (2.7)
= if null[p] then 0 else numstate[first[p]]
+length[comprog[rest[p],2]]

where we have made use of a lemma whose proof is straightforward but lengthy, namely

(9.9) Lemma: numstate[select[k,γ]]=length[compstate[select[k,γ],k]]

Compare the result just obtained for length[comprog[p,1]] with the definition of fl[p] given in [7.8]. Since they have the same functional form, we have by recursion induction that

(9.10) fl[p] = length[comprog[p,1]].

Then, from (7.8) we obtain

f[$c(\alpha,\xi)$] = if $c(\alpha,\xi)$=1 then 1 else 1+fl[sublist[$c(\alpha,\xi)$-1,γ]] .

But since $\gamma = p*s*q$ and $\underline{length}[p] = c(\alpha,\xi)-1$, then

$$f[c(\alpha,\xi)] = \underline{if}\ c(\alpha,\xi)=1\ \underline{then}\ 1\ \underline{else}\ 1+fl[p] \qquad (2.9)$$

$$= \underline{if}\ c(\alpha,\xi)=1\ \underline{then}\ 1\ \underline{else}\ 1+\underline{length}[comprog[p,1]] \qquad (9.10)$$

But if $c(\alpha,\xi)=1$ then $\underline{length}[p] = 0$ and $\underline{length}[comprog[p,1]] = 0$ (7.2)

so that,

$$(9.11) \qquad f[c(\alpha,\xi)] = 1 + \underline{length}[comprog[p,1]].$$

Recall that $f[c(\alpha,\xi)] = c(\alpha,\theta)$ so that we may write for the left hand side of the statement for lemma (9.6)

$$\underline{select}[c(\alpha,\theta),\omega] = \underline{select}[f[c(\alpha,\theta)],\omega]$$

$$= \underline{select}[1+\underline{length}[comprog[p,1]],\ comprog[p,1]*compstate[s,c(\alpha,\xi)]$$

$$*comprog[q,c(\alpha,\xi)+1] \qquad (9.11)\ (9.8)$$

$$= \underline{first}[compstate[s,c(\alpha,\xi)]] \qquad (2.8)$$

$$= \underline{first}[compstate[\underline{select}[c(\alpha,\xi),\gamma],c(\alpha,\xi)]]$$

which completes the proof for lemma (9.6). Thus, we now see that $execobject[\omega,c(\alpha,\theta),n,\theta]$ is the state vector produced by executing the compiled code for the statement $\underline{select}[c(\alpha,\xi),\gamma]$. With this fact in mind, we now proceed with the main proof results of this section.

We first derive an expression for $execsource[\gamma,c(\alpha,\xi),1,\xi] \equiv X$.

$$X \equiv \underline{if}\ c(\alpha,\xi) < c(\alpha,\xi) \vee c(\alpha,\xi) \geq c(\alpha,\xi)+1\ \underline{then}\ \xi\ \underline{else}$$

$$execsource[\gamma,c(\alpha,\xi),1,stepsource[\underline{select}[c(\alpha,\xi),\gamma],\xi]] \qquad (3.7)$$

$$\equiv execsource[\gamma,c(\alpha,\xi),1,stepsource[s,\xi]] \quad \text{where}$$

$$s = \underline{select}[c(\alpha,\xi),\gamma]\ .$$

There are three possibilities: either istransfer[s] or isassignment[s] or isconditional[s]; we consider each case in turn.

Case 1: istransfer[s]. Here,

$$X \equiv execsource[\gamma,c(\alpha,\xi),1,\delta] \text{ where} \qquad (3.8)$$

$$\delta \equiv a(\alpha,destination[s],\xi).\ \text{Then,}$$

$$X \equiv \underline{if}\ c(\alpha,\delta) < c(\alpha,\xi) \vee c(\alpha,\delta) \geq c(\alpha,\xi)+1\ \underline{then}\ \delta\ \underline{else}$$

$$execsource[\gamma,c(\alpha,\xi),1,stepsource[select[c(\alpha,\delta),\gamma],\delta]] \qquad (3.7)$$

$X \equiv$ if $c(\alpha,\delta) \neq c(\alpha,\xi)$ then δ else

execsource$[\gamma, c(\alpha,\xi), 1,$ stepsource$[$select$[c(\alpha,\xi),\gamma],\xi]]$ (3.3)

$\equiv$ if destination$[s] \neq c(\alpha,\xi)$ then $a(\alpha,$destination$[s],\xi)$ else X (3.1)

Thus, X remains undefined if s is a transfer statement to itself, i.e., the source program gets caught in an infinite one statement loop.

Case 2: isassignment[s]. Here,

$X \equiv$ execsource$[\gamma, c(\alpha,\xi), 1, \delta]$ where (3.8)

$\delta \equiv a(\alpha, 1+c(\alpha,\xi), a($leftside$[s],$ value$[$rightside$[s],\xi],\xi))$.

$X \equiv c(\alpha,\delta) \neq c(\alpha,\xi)$ then δ else ... (3.7)

$\equiv$ if $1+c(\alpha,\xi) \neq c(\alpha,\xi)$ then δ else ... (3.1)

$\equiv \delta$

$\equiv a(\alpha, 1+c(\alpha,\xi), a($leftside$[s],$ value$[$rightside$[s],\xi],\xi))$.

Case 3: isconditional[s]. Here,

$X \equiv$ execsource$[\gamma, c(\alpha,\xi), 1, \delta]$ where (3.8)

$\delta \equiv a(\alpha,$[if bvalue[premise[s],ξ] then 1 else 2]$+c(\alpha,\xi),\xi)$.

$X \equiv$ if $c(\alpha,\delta) \neq c(\alpha,\xi)$ then δ else ... (3.7)

$\equiv$ if [if bvalue[premise[s],ξ] then 1 else 2]$+c(\alpha,\xi) \neq c(\alpha,\xi)$

then δ else ... (3.1)

$\equiv \delta$

$\equiv a(\alpha,$[if bvalue[premise[s],ξ] then 1 else2]$+c(\alpha,\xi),\xi)$.

These three cases may be combined to give

(9.12) execsource$[\gamma, c(\alpha,\xi), 1, \xi] \equiv$

if istransfer[s] then

[if destination$[s] \neq c(\alpha,\xi)$ then $a(\alpha,$destination$[s],\xi)$

else execsource$[\gamma, c(\alpha,\xi), 1, \xi]$]

else if isassignment[s] then

$a(\alpha, 1+c(\alpha,\xi), a($leftside$[s],$ value$[$rightside$[s],\xi],\xi))$

else if isconditional[s] then

$a(\alpha,$[if bvalue[premise[s],ξ] then 1 else 2]$+c(\alpha,\xi),\xi)$

We must now derive an expression for execobject$[\omega, c(\alpha,\theta), n, \theta] \equiv Y$.

Let ω = p*compstate$[s, c(\alpha,\xi)]$*q so that $c(\alpha,\theta)=$length$[p]+1$. (9.6) (2.8)

Again, there are three cases to consider.

Case 1: istransfer[s]. Here,

$Y \equiv$ execobject[p*mktra[f[destination[s]]*q, c(α,θ), 1, θ] (7.3)

$\equiv$ if c(α,θ)< c(α,θ) $\vee$ c(α,θ) $\geq$ c(α,θ)+1 then θ else (6.5)
execobject[ω, c(α,θ), 1, stepobject[mktra[f[destination[s]]], θ]] (9.6)

$\equiv$ execobject[ω, c(α,θ), 1, δ] where (6.6)

$\delta \equiv$ a(α, dest[mktra[f[destination[s]]]], θ)

$\equiv$ a(α, f[destination[s]], θ). Then, (5.6)

$Y \equiv$ if c(α,δ)< c(α,θ) $\vee$ c(α,δ) $\geq$ c(α,θ)+1 then δ else
execobject[ω, c(α,θ), 1, stepobject[select[c(α,δ), ω], δ]] (6.5)

$\equiv$ if c(α,δ) $\neq$ c(α,θ) then δ else
execobject[ω, c(α,θ), 1, stepobject[select[c(α,θ), ω], θ]] (3.3)

$\equiv$ if c(α,δ) $\neq$ c(α,θ) then δ else Y (6.5)

$\equiv$ if f[destination[s]]$\neq$c(α,θ) then a(α, f[destination[s]], θ) else Y (3.1)

Thus, Y remains undefined if s is a transfer statement to itself, i.e., the object program gets caught in an infinite one instruction loop.

Case 2: isassignment[s].

To obtain results for this case will require more effort than for Case 1. We start by investigating arithmetic expressions and this will lead to a discussion of assignment statements. Our first task is to prove the following

(9.13) Theorem: For any source program γ and object program ω = compile[γ], if ξ and θ are state vectors such that c(v,ξ) = c(loc[v], θ) for all variables v in varlist[γ], and if e is an arithmetic expression part of some statement in γ, and if c(α,θ) = length[p]+1 where ω = p*comparith[e,i]*q for some integer $i \geq t[\gamma]$, then

execobject[p*comparith[e,i]*q, length[p]+1, length[comparith[e,i]], θ]

$\equiv_i$ a(α, length[p*comparith[e,i]]+1, a(β, value[e,ξ], θ)).

Aside from the introduction of the instruction counter α, this theorem is essentially the one proven in [5]. The proof is accomplished

by an induction on the expression e being compiled. We prove it first for constants, then for variables, and then for sums on the induction hypothesis that it is true for the summands.

Let execobject[ω, length[p]+1, length[comparith[e,i]], θ] $\equiv$ Z.

Case (a): isconst[e]. Here,

$Z \equiv$ execobject[p*mkli[val[e]]*q, length[p]+1, 1, θ] (2.2) (2.5) (7.4)

$\equiv$ execobject[ω, length[p]+1, 1, stepobject[mkli[val[e]], θ]] (2.8)

$\equiv$ execobject[ω, length[p]+1, 1, δ] where (6.6)

$\delta \equiv a(\alpha, 1+c(\alpha,\theta), a(\beta, arg[mkli[val[e]]], \theta))$

$\equiv a(\alpha, length[p]+2, a(\beta, val[e], \theta))$. Then, (5.1)

$Z \equiv$ if $c(\alpha,\delta) <$ length[p]+1 $\vee$ $c(\alpha,\delta) \geq$ length[p]+2 then δ else ... (6.5)

$\equiv$ if length[p]+2 < length[p]+1 $\vee$ length[p]+2 $\geq$ length[p]+2 then δ else ... (3.1)

$\equiv \delta$

$\equiv a(\alpha, length[p]+2, a(\beta, val[e], \theta))$

$\equiv_i a(\alpha, length[p*comparith[e,i]]+1, a(\beta, value[e,\xi], \theta))$ (9.2)(2.7) (3.9)

Case (b): isvar[e]. Here,

$Z \equiv$ execobject[p*mkload[loc[e]]*q, length[p]+1, 1, θ] (2.2)(2.5) (7.4)

$\equiv$ execobject[ω, length[p]+1, 1, stepobject[mkload[loc[e]], θ]] (2.8)

$\equiv$ execobject[ω, length[p]+1, 1, δ] where (6.6)

$\delta \equiv a(\alpha, 1+c(\alpha,\theta), a(\beta, c(adr[mkload[loc[e]]], \theta), \theta))$

$\equiv a(\alpha, length[p]+2, a(\beta, c(loc[e], \theta), \theta))$. Then (5.2)

$Z \equiv$ if $c(\alpha,\delta) <$ length[p]+1 $\vee$ $c(\alpha,\delta) \geq$ length[p]+2 then δ else ... (6.5)

$\equiv$ if length[p]+2 < length[p]+1 $\vee$ length[p]+2 $\geq$ length[p]+2 then δ else ... (3.1)

$\equiv \delta$

$\equiv a(\alpha, length[p]+2, a(\beta, c(loc[e], \theta), \theta))$

$\equiv_i a(\alpha, length[p*comparith[e,i]]+1, a(\beta, c(loc[e], \theta), \theta))$ (2.7) (9.2)

But recall that we are given $c(loc[e], \theta) = c(e, \xi)$, so that

$Z \equiv_i a(\alpha, length[p*comparith[e,i]]+1, a(\beta, value[e,\xi], \theta))$ (3.9)

Case (c): issum[e]. Here,

$Z \equiv$ execobject[p*comparith[s2[e],i]*mksto[i]*comparith[s1[e],i+1]
*mkadd[i]*q,$\underline{\text{length}}$[p]+1,$\underline{\text{length}}$[comparith[e,i]],$\theta$] (7.4)

Let p_1 = p*comparith[s2[e],i], $p_2 = p_1$*mksto[i],
$p_3 = p_2$*comparith[s1[e],i+1], $p_4 = p_3$*mkadd[i] so that $\omega = p_4$*q,
and let m = $\underline{\text{length}}$[comparith[e,i]]. Then,

$Z \equiv$ execobject[ω,$\underline{\text{length}}$[p]+1,m,
execobject[ω,$\underline{\text{length}}$[p]+1, $\underline{\text{length}}$[comparith[s2[e],i]],$\theta$]] (8.2)

With the induction hypothesis that the theorem holds for s2[e], we have,

$Z \equiv$ execobject[ω,$\underline{\text{length}}$[p]+1,m,$\delta_1$] where
$\delta_1 \equiv_i a(\alpha,\underline{\text{length}}[p_1]+1,a(\beta,\text{value}[s2[e],\xi],\theta))$. Then,

$Z \equiv$ execobject[ω,$\underline{\text{length}}$[p]+1,m,execobject[$\omega$,$\underline{\text{length}}$[$p_1$]+1,1,$\delta_1$]] (8.2)

$\equiv$ execobject[ω,$\underline{\text{length}}$[p]+1,m,stepobject[mksto[i],δ_1]] (6.5) (2.8)

$\equiv$ execobject[ω,$\underline{\text{length}}$[p]+1,m,$\delta_2$] where
$\delta_2 \equiv a(\alpha,1+c(\alpha,\delta_1),a(\text{adr}[\text{mksto}[i]],c(\beta,\delta_1),\delta_1))$ (6.6)
$\equiv a(\alpha,\underline{\text{length}}[p_2]+1,a(i,\text{value}[s2[e],\xi],\delta_1)$ (2.7) (3.1) (3.2)
$\equiv_{i+1} a(\alpha,\underline{\text{length}}[p_2]+1,a(i,\text{value}[s2[e],\xi],$
$a(\beta,\text{value}[s2[e],\xi],\theta)))$. Then, (3.2) (9.3) (9.5)

$Z \equiv$ execobject[ω,$\underline{\text{length}}$[p]+1,m,
execobject[ω,$\underline{\text{length}}$[$p_2$]+1,$\underline{\text{length}}$[comparith[s1[e],i+1]],$\delta_2$]] (8.2)

From the expression for δ_2 we see that $c(v,\xi) = c(\text{loc}[v],\delta_2)$ for all variables v in varlist[γ]. So using the induction hypothesis that the theorem holds for s1[e], we have

$Z \equiv$ execobject[ω,$\underline{\text{length}}$[p]+1,m,$\delta_3$] where
$\delta_3 \equiv_{i+1} a(\alpha,\underline{\text{length}}[p_3]+1,a(\beta,\text{value}[s1[e],\xi],\delta_2))$
$\equiv_{i+1} a(\alpha,\underline{\text{length}}[p_3]+1,a(\beta,\text{value}[s1[e],\xi],$
$a(i,\text{value}[s2[e],\xi],\theta)))$. Then, (9.3) (9.5) (3.2)

$Z \equiv$ execobject[ω,$\underline{\text{length}}$[p]+1,m,execobject[$\omega$,$\underline{\text{length}}$[$p_3$]+1,1,$\delta_3$]] (8.2)
execobject[ω,$\underline{\text{length}}$[p]+1,m,stepobject[mkadd[i],δ_3]] (6.5) (2.8)
execobject[ω,$\underline{\text{length}}$[p]+1,m,$\delta_4$] where

$$\delta_4 \equiv a(\alpha, 1+c(\alpha,\delta_3), a(\beta, c(\beta,\delta_3)\&c(adr[mkadd[i]],\delta_3),\delta_3)) \quad (6.6)$$

$$\equiv a(\alpha, \underline{length}[p_4]+1, \quad (5.4)$$

$$a(\beta, value[s1[e],\xi]\&value[s2[e],\xi],\delta_3)) \quad (2.7)\ (3.1)\ (3.2)$$

$$\equiv_i a(\alpha, \underline{length}[p_4]+1, a(\beta, value[s1[e],\xi]\&value[s2[e],\xi],\theta)) \quad (9.5)\ (3.2)$$

Note that $\underline{length}[p_4]+1 = \underline{length}[p*comparith[e,i]]+1$

$$= \underline{length}[p]+1 + \underline{length}[comparith[e,i]] \quad (2.7)$$

$$= \underline{length}[p]+1 + m \quad \text{so that,}$$

$$Z \equiv \underline{if}\ c(\alpha,\delta_4)<1 \vee c(\alpha,\delta_4)\geq \underline{length}[p]+1+m\ \underline{then}\ \delta_4\ \underline{else}\ \ldots \quad (6.5)$$

$$\equiv \underline{if}\ \underline{length}[p_4]+1 < 1 \vee \underline{length}[p_4]+1 \geq \underline{length}[p]+1+m\ \text{then}\ \delta_4\ \underline{else}\ \ldots \quad (3.1)$$

$$\equiv \delta_4$$

$$\equiv_i a(\alpha, \underline{length}[p*comparith[e,i]]+1,$$

$$a(\beta, value[s1[e],\xi]\ \&\ value[s2[e],\xi],\theta))$$

$$\equiv_i a(\alpha, \underline{length}[p*comparith[e,i]]+1, a(\beta, value[e,\xi],\theta)) \quad (3.9)$$

This completes the proof of theorem (9.13).

Recall that we are considering the case isassignment[s] where $s = \underline{select}[c,(\alpha,\xi),\gamma]$ and we want to derive an expression for $execobject[\omega, c(\alpha,\theta), n, \theta] \equiv Y$ where $\omega = p*compstate[s,c(\alpha,\xi)]*q$, $n = \underline{length}[compstate[s,c(\alpha,\xi)]]$ and $c(\alpha,\theta) = \underline{length}[p]+1$. Then,

$$Y \equiv execobject[p*compstate[s,c(\alpha,\xi)]*q, c(\alpha,\theta), n, \theta]$$

$$\equiv execobject[p*comparith[rightside[s],t[\gamma]]*$$

$$mksto[loc[leftside[s]]]*q, c(\alpha,\theta), n, \theta] \quad (7.3)$$

$$\equiv execobject[\omega, c(\alpha,\theta), n, execobject[\omega, c(\alpha,\theta), n-1, \theta]] \text{ where} \quad (8.2)$$

$$n-1 = \underline{length}[comparith[rightside[s],t[\gamma]]]. \quad \text{Then,} \quad (2.7)$$

$$Y \equiv execobject[\omega, c(\alpha,\theta), n, \delta_1] \text{ where} \quad (2.7)$$

$$\delta_1 \equiv_{t[\gamma]} a(\alpha, length[p]+n-1+1, a(\beta, value[rightside[s],\xi],\theta)) \quad (9.13)$$

$$Y \equiv execobject[\omega, c(\alpha,\theta), n, execobject[\omega, \underline{length}[p]+n, 1, \delta_1]] \quad (8.2)$$

$$\equiv execobject[\omega, c(\alpha,\theta), n, \delta_2] \text{ where}$$

$$\delta_2 \equiv a(\alpha, 1+c(\alpha,\delta_1), \quad (6.6)$$

$$a(adr[mksto[loc[leftside[s]]]], c(\beta,\delta_1),\delta_1)) \quad (2.8)\ (6.5)$$

$$\delta_2 \equiv a(\alpha, \underline{length}[p]+n+1, \quad (5.3)$$
$$a(loc[leftside[s]], value[rightside[s], \xi], \delta_1)) \quad (3.1)\ (3.2)$$

$$\equiv_{t[\gamma],\beta} a(\alpha, \underline{length}[p]+n+1, \quad (3.2)$$
$$a(loc[leftside[s]], value[rightside[s], \xi], \theta)) \quad (9.3)\ (9.5)$$

$$Y \equiv \underline{if}\ c(\alpha,\delta_2) < c(\alpha,\theta) \vee c(\alpha,\delta_2) \geq c(\alpha,\theta)+n\ \underline{then}\ \delta_2\ \underline{else}\ \ldots \quad (6.5)$$
$$\equiv \underline{if}\ \underline{length}[p]+n+1 < \underline{length}[p]+1 \vee \underline{length}[p]+n+1 \geq \underline{length}[p]+n+1$$
$$\underline{then}\ \delta_2\ \underline{else}\ \ldots \quad (3.1)$$
$$\equiv \delta_2$$
$$\equiv_{t[\gamma],\beta} a(\alpha, \underline{length}[p*compstate[s, c(\alpha,\xi)]]+1,$$
$$a(loc[leftside[s], value[rightside[s],\xi],\theta)) \quad (2.7)$$
$$\equiv_{t[\gamma],\beta} a(\alpha, f[c(\alpha,\xi)+1], a(loc[leftside[s]], value[rightside[s],\xi],\theta)) \quad (9.11)$$

This completes the derivation of an expression for the state vector execobject[ω, c(α,θ), n, θ] $\equiv$ Y for the case isassignment[s].

Case 3: isconditional[s].

This case is treated much like Case 2. We start by investigating boolean expressions and this will lead into a discussion of conditional statements. Our first task is to prove the following

(9.14) Theorem: For any source program γ and object program ω = compile[γ], if ξ and θ are state vectors such that c(v,ξ) = c(loc[v],θ) for all variables v in varlist[γ], and if b is a boolean expression part of some statement in γ, and if c(α,θ) = $\underline{length}$[p]+1 = k where ω = p*compbool[b,k]*q, then

execobject[p*compbool[b,k]*q, $\underline{length}$[p]+1, $\underline{length}$[compbool[b,k]], θ]
$\equiv_{t[\gamma]}$ a(α, $\underline{length}$[p*compbool[b,k]]+1, a(β, [$\underline{if}$ bvalue[b,ξ]
$\underline{then}$ 1 $\underline{else}$ -1], θ))

The proof here is much like that for (9.8) and is accomplished by an induction on the expression b being compiled. We prove it first for greater-than relations and then for logical products and negations on the induction hypothesis that it is true for their operands.

Let execobject[ω, $\underline{length}$[p]+1, $\underline{length}$[compbool[b,k]], θ] $\equiv$ Z.

Case (a): isgreater[b]. Here,

$$Z \equiv \text{execobject}[p*\text{comparith}[\text{right}[b],t[\gamma]]*\text{mksto}[t[\gamma]]*\text{comparith}[\text{left}[b],t[\gamma]+1]*\text{mkgth}[t[\gamma]]*q,\underline{\text{length}}[p]+1,\underline{\text{length}}[\text{compbool}[b,k]].\theta] \quad (7.5)$$

Let $p_1 = p*\text{comparith}[\text{right}[b],t[\gamma]]$, $p_2 = p_1*\text{mksto}[t[\gamma]]$, $p_3 = p_2*\text{comparith}[\text{left}[b],t[\gamma]+1]$, $p_4 = p_3*\text{mkgth}[t[\gamma]]$ so that $\omega = p_4*q$, and let m = length[compbool[b,k]]. Then,

$$Z \equiv \text{execobject}[\omega,\underline{\text{length}}[p]+1,m,\text{execobject}[\omega,\underline{\text{length}}[p]+1,\underline{\text{length}}[\text{comparith}[\text{right}[b],t[\gamma]]],\theta] \quad (8.2)$$

$$\equiv \text{execobject}[\omega,\underline{\text{length}}[p]+1,m,\delta_1] \text{ where}$$

$$\delta_1 \equiv_{t[\gamma]} a(\alpha,\underline{\text{length}}[p_1]+1,a(\beta,\text{value}[\text{right}[b],\xi],\theta)). \text{ Then} \quad (9.13)$$

$$Z \equiv \text{execobject}[\omega,\underline{\text{length}}[p]+1,m,\text{execobject}[\omega,\underline{\text{length}}[p_1]+1,1,\delta_1]] \quad (8.2)$$

$$\equiv \text{execobject}[\omega,\underline{\text{length}}[p]+1,m,\text{stepobject}[\text{mksto}[t[\gamma]],\delta_1]] \quad (6.5)\ (2.8)$$

$$\equiv \text{execobject}[\omega,\underline{\text{length}}[p]+1,m,\delta_2] \text{ where}$$

$$\delta_2 \equiv a(\alpha,1+c(\alpha,\delta_1),a(\text{adr}[\text{mksto}[t[\gamma]]],c(\beta,\delta_1),\delta_1)) \quad (6.6)$$

$$\equiv a(\alpha,\underline{\text{length}}[p_2]+1,a(t[\gamma],\text{value}[\text{right}[b],\xi],\delta_1)) \quad (3.1)\ (3.2)$$

$$\equiv_{t[\gamma]+1} a(\alpha,\underline{\text{length}}[p_2]+1,a(t[\gamma],\text{value}[\text{right}[b],\xi], a(\beta,\text{value}[\text{right}[b],\xi],\theta))) \quad (2.7)\ (5.3)\ (9.3)\ (9.5)\ (3.2)$$

$$Z \equiv \text{execobject}[\omega,\underline{\text{length}}[p]+1,m,\text{execobject}[\omega,\underline{\text{length}}[p_2]+1,\underline{\text{length}}[\text{comparith}[\text{left}[b],t[\gamma]+1]],\delta_2]] \quad (8.2)$$

$$\equiv \text{execobject}[\omega,\underline{\text{length}}[p]+1,m,\delta_3] \text{ where}$$

$$\delta_3 \equiv a(\alpha,\underline{\text{length}}[p_3]+1,a(\beta,\text{value}[\text{left}[b],\xi],\delta_2)) \quad (9.13)$$

$$\equiv_{t[\gamma]+1} a(\alpha,\underline{\text{length}}[p_3]+1,a(\beta,\text{value}[\text{left}[b],\xi], a(t[\gamma],\text{value}[\text{right}[b],\xi],\theta))) \quad (9.3)\ (9.5)\ (3.2)$$

Note that in the application of (9.13) here, we have $c(v,\xi) = c(\text{loc}[v],\delta_2)$ for all variables v in varlist[γ] as required. The final step is then,

$$Z \equiv \text{execobject}[\omega,\underline{\text{length}}[p]+1,m,\text{execobject}[\omega,\underline{\text{length}}[p_3]+1,1,\delta_3]] \quad (8.2)$$

$$\equiv \text{execobject}[\omega,\underline{\text{length}}[p]+1,m,\text{stepobject}[\text{mkgth}[t[\gamma]],\delta_3]] \quad (6.5)\ (2.8)$$

$$\equiv \text{execobject}[\omega,\underline{\text{length}}[p]+1,m,\delta_4] \text{ where}$$

$$\delta_4 \equiv a(\alpha,1+c(\alpha,\delta_3),a(\beta,[\underline{\text{if}}\ c(\beta,\delta_3) > c(\text{adr}[\text{mkgth}[t[\gamma]]],\delta_3) \underline{\text{then}}\ 1\ \underline{\text{else}}\ -1],\delta_3)) \quad (6.6)$$

$$\delta_4 \equiv a(\alpha, \underline{length}[p_4]+1, a(\beta, [\underline{if}\ value[left[b],\xi] > value[right[b],\xi]\ \underline{then}\ 1\ \underline{else}\ -1], \delta_3)) \quad (3.1)\ (3.2)\ (2.7)\ (5.5)$$

$$\equiv_{t[\gamma]} a(\alpha, \underline{length}[p_4]+1, a(\beta, [\underline{if}\ value[left[b],\xi] > value[right[b],\xi]\ \underline{then}\ 1\ \underline{else}\ -1],)) \quad (9.3)\ (9.5)\ (3.2)$$

Note that $\underline{length}[p_4]+1 = \underline{length}[p*compbool[b,k]]+1$

$= \underline{length}[p]+1 + \underline{length}[compbool[b,k]]$ (2.7)

$= \underline{length}[p]+1 + m$ so that,

$$Z \equiv \underline{if}\ c(\alpha,\delta_4) < \underline{length}[p]+1 \vee c(\alpha,\delta_4) \geq \underline{length}[p]+1+m\ \underline{then}\ \delta_4\ \underline{else}\ \ldots$$

$$\equiv \underline{if}\ \underline{length}[p_4]+1 < \underline{length}[p]+1 \vee \underline{length}[p_4]+1 \geq \underline{length}[p]+1+m \quad (6.5)$$

$\underline{then}\ \delta_4\ \underline{else}\ \ldots$ (3.1)

$\equiv \delta_4$

$\equiv_{t[\gamma]} a(\alpha, \underline{length}[p*compbool[b,k]]+1, a(\beta, [\underline{if}\ value[left[b],\xi] > value[right[b],\xi]\ \underline{then}\ 1\ \underline{else}\ -1], \theta))$

$\equiv_{t[\gamma]} a(\alpha, \underline{length}[p*compbool[b,k]]+1, a(\beta, [\underline{if}\ bvalue[b,\xi]\ \underline{then}\ 1\ \underline{else}\ -1], \theta))$ (3.10)

Case (b): isand[b]. Here,

$Z \equiv$ execobject[p*compbool[bl[b],k]*mktmi[k+numbool[b]]
*compbool[b2[b],k+numbool[bl[b]]+1]*q, $\underline{length}$[p]+1,
$\underline{length}$[compbool[b,k]], θ] (7.5)

Let $p_1 = p*compbool[bl[b],k]$, $p_2 = p_1*mktmi[k+numbool[b]]$,
$p_3 = p_2*compbool[b2[b],k+numbool[bl[b]]+1]$ so that $= p_3*q$,
and let m = $\underline{length}$[compbool[b,k]]. Then,

$Z \equiv$ execobject[ω, length[p]+1, m,
execobject[ω, $\underline{length}$[p]+1, $\underline{length}$[compbool[bl[b],k]], θ]] (8.2)

With the induction hypothesis that the theorem holds for bl[b], we have

$Z \equiv$ execobject[ω, $\underline{length}$[p]+1, m, δ_1] where

$\delta_1 \equiv_{t[\gamma]} a(\alpha, \underline{length}[p_1]+1, a(\beta, [\underline{if}\ bvalue[bl[b],\xi]\ \underline{then}\ 1\ \underline{else}\ -1], \theta))$. Then,

$Z \equiv$ execobject[ω, $\underline{length}$[p]+1, m, execobject[ω, $\underline{length}$[p_1]+1, 1, δ_1]] (8.2)

$\equiv$ execobject[ω, $\underline{length}$[p]+1, m, stepobject[mktmi[k+numbool[b]], δ_1]] (6.5) (2.8)

$Z \equiv$ execobject[ω, length[p]+1, m, δ_2] where

$$\delta_2 \equiv a(\alpha, [\underline{\text{if}}\ c(\beta, \delta_1) < 0\ \underline{\text{then}}\ \text{dest}[\text{mktmi}[k+\text{numbool}[b]]]\ \underline{\text{else}}\ 1 + c(\alpha, \delta_1)], \delta_1) \quad (6.6)$$

$$\equiv a(\alpha, [\underline{\text{if}}[\underline{\text{if}}\ \text{bvalue}[\text{bl}[b], \xi]\ \underline{\text{then}}\ 1\ \underline{\text{else}}\ -1] < 0\ \underline{\text{then}}\ k+\text{numbool}[b]\ \underline{\text{else}}\ \text{length}[p_2]+1], \delta_1) \quad (5.7)\ (3.1)\ (3.2)\ (2.7)$$

$$\equiv a(\alpha, [\underline{\text{if}}\ \text{bvalue}[\text{bl}[b], \xi]\ \underline{\text{then}}\ [\underline{\text{if}}\ 1 < 0\ \underline{\text{then}}\ k+\text{numbool}[b]\ \underline{\text{else}}\ \text{length}[p_2]+1]\ \underline{\text{else}}\ [\underline{\text{if}}\ -1 < 0\ \underline{\text{then}}\ k+\text{numbool}[b]\ \underline{\text{else}}\ \text{length}[p_2]+1]], \delta_1)$$

$$\equiv a(\alpha, [\underline{\text{if}}\ \text{bvalue}[\text{bl}[b], \xi]\ \underline{\text{then}}\ \text{length}[p_2]+1\ \underline{\text{else}}\ k+\text{numbool}[b]], \delta_1)$$

In the proof of (9.9) the following lemma arises and is easily proved by recursion induction.

(9.15) Lemma: If b is a boolean expression, then

numbool[b] = length[compbool[b,k]].

Also recall that we are given $k = \text{length}[p]+1$, so that

$$k + \text{numbool}[b] = \text{length}[p]+1 + \text{length}[\text{compbool}[b,k]] \quad (9.15)$$

$$= \text{length}[p*\text{compbool}[b,k]]+1 \quad (2.7)$$

$$= \text{length}[p_3]+1 \quad \text{so that,}$$

$$\delta_2 \equiv a(\alpha, [\underline{\text{if}}\ \text{bvalue}[\text{bl}[b], \xi]\ \underline{\text{then}}\ \text{length}[p_2]+1\ \underline{\text{else}}\ \text{length}[p_3]+1], \delta_1)$$

$$\equiv_{t[\gamma]} a(\alpha, [\underline{\text{if}}\ \text{bvalue}[\text{bl}[b], \xi]\ \underline{\text{then}}\ \text{length}[p_2]+1\ \underline{\text{else}}\ \text{length}[p_3]+1], a(\beta, [\underline{\text{if}}\ \text{bvalue}[\text{bl}[b], \xi]\ \underline{\text{then}}\ 1\ \underline{\text{else}}\ -1], \theta)).$$ Then, (9.3) (9.5) (3.2)

$$Z \equiv \underline{\text{if}}\ c(\alpha, \delta_2) < \text{length}[p]+1 \vee c(\alpha, \delta_2) \geq \text{length}[p]+1+m\ \underline{\text{then}}\ \delta_2\ \underline{\text{else}}\ \text{execobject}[\omega, \text{length}[p]+1, m, \text{stepobject}[\text{select}[c(\alpha, \delta_2), \omega], \delta_2]] \quad (6.5)$$

Note that $\text{length}[p_3]+1 = \text{length}[p] + \text{length}[\text{compbool}[b,k]] + 1 = \text{length}[p]+1 + m$, so that (2.7)

$$Z \equiv \underline{\text{if}}\ [\underline{\text{if}}\ \text{bvalue}[\text{bl}[b], \xi]\ \underline{\text{then}}\ \text{length}[p_2]+1\ \underline{\text{else}}\ \text{length}[p_3]+1] < \text{length}[p]+1$$
$$\vee\ [\underline{\text{if}}\ \text{bvalue}[\text{bl}[b], \xi]\ \underline{\text{then}}\ \text{length}[p_2]+1\ \underline{\text{else}}\ \text{length}[p_3]+1] \geq \text{length}[p_3]+1\ \underline{\text{then}}\ \delta_2\ \underline{\text{else}}$$

execobject[ω,$\underline{\text{length}}$[p]+1,m,stepobject[$\underline{\text{select}}$

[$\underline{\text{if}}$ bvalue[bl[b],ξ] $\underline{\text{then}}$ $\underline{\text{length}}$[$p_2$]+1 $\underline{\text{else}}$ $\underline{\text{length}}$[$p_3$]+1],$\omega$]$\delta_2$]] (3.1)

Z $\equiv$ $\underline{\text{if}}$ bvalue[bl[b],ξ] $\underline{\text{then}}$

[$\underline{\text{if}}$ $\underline{\text{length}}$[$p_2$]+1 < $\underline{\text{length}}$[p]+1 $\vee$ $\underline{\text{length}}$[$p_2$]+1 $\geq$ $\underline{\text{length}}$[$p_3$]+1

$\underline{\text{then}}$ δ_2 $\underline{\text{else}}$

execobject[ω,$\underline{\text{length}}$[p]+1,m,stepobject[$\underline{\text{select}}$[$\underline{\text{length}}$[$p_2$]+1,$\omega$],$\delta_2$]]]

$\underline{\text{else}}$

[$\underline{\text{if}}$ $\underline{\text{length}}$[$p_3$]+1 < $\underline{\text{length}}$[p]+1 $\vee$ $\underline{\text{length}}$[$p_3$]+1 $\geq$ $\underline{\text{length}}$[p]+1

$\underline{\text{then}}$ δ_2 $\underline{\text{else}}$ ...]

$\equiv$ $\underline{\text{if}}$ bvalue[bl[b],ξ] $\underline{\text{then}}$ execobject[ω,$\underline{\text{length}}$[p]+1,m,

execobject[ω,$\underline{\text{length}}$[$p_2$]+1,1,$\delta_2$]] $\underline{\text{else}}$ δ_2 (6.5)

$\equiv$ $\underline{\text{if}}$ bvalue[bl[b],ξ] $\underline{\text{then}}$ W $\underline{\text{else}}$ δ_2 where

W $\equiv$ execobject[ω,$\underline{\text{length}}$[p]+1,m,

execobject[ω,$\underline{\text{length}}$[$p_2$]+1,numbool[b2[b]],$\delta_2$]] (9.15) (8.3)

At this point we want to make the induction hypothesis that the theorem holds for b2[b]. But, according to the statement for theorem (9.14) we must first verify that $\underline{\text{length}}$[$p_2$]+1=numbool[bl[b]]+1.

$\underline{\text{length}}$[$p_2$]+1 = $\underline{\text{length}}$[p] + $\underline{\text{length}}$[compbool[bl[b],k]] + 1 + 1 (2.7)

= $\underline{\text{length}}$[p]+1 + numbool[bl[b]] + 1 (9.15)

= k + numbool[bl[b]] + 1 as required.

As well, the statement for theorem (9.14) also requires that c(v,ξ) = c(loc[v],δ_2) for all variables v in varlist [γ]. This result follows immediately from (9.4) applied to the expression for δ_2 above. Thus, we can use the induction hypothesis that theorem (9.14) holds for b2[b].

Note that if $\neg$ bvalue[bl[b],ξ] then c(α,δ_2) is set so that execution of compbool[b2[b],...] will be bypassed. This is an efficient way to process the logical product of two boolean quantities when the first is found to be false.

W $\equiv$ execobject[ω,$\underline{\text{length}}$[p]+1,m,$\delta_3$] where

δ_3 $\equiv_{t[\gamma]}$ a(α,$\underline{\text{length}}$[$p_3$]+1,

a(β,[$\underline{\text{if}}$ bvalue[b2[b],ξ] $\underline{\text{then}}$ 1 $\underline{\text{else}}$ -1],δ_2))

$W \equiv$ if $c(\alpha,\delta_3) <$ length[p]+1 $\vee$ $c(\alpha,\delta_3) \geq$ length[p]+1+m then δ_3 else ...

$\equiv$ if length[p_3]+1 < length[p]+1 $\vee$ length[p_3]+1 $\geq$ length[p_3]+1 (6.5)

then δ_3 else ... (3.1)

$\equiv \delta_3$ so that we have,

$Z \equiv$ if bvalue[bl[b],ξ] then δ_3 else δ_2

$\equiv_{t[\gamma]}$ if bvalue[bl[b],ξ] then

a(α,length[p_3]+1,a(β,[if bvalue[b2[b],ξ] then 1 else -1],δ_2))

else

a(α,[if bvalue[bl[b],ξ] then length[p_2]+1 else length[p_3]+1,

a(β,[if bvalue[bl[b],ξ] then 1 else -1],θ)) (9.3) (9.5)

$\equiv_{t[\gamma]}$ if bvalue[bl[b],ξ] then

a(α,length[p_3]+1,a(β,[if bvalue[b2[b],ξ] then 1 else -1],θ))

else a(α,length[p_3]+1,a(β,-1,θ)) (3.2)

$\equiv_{t[\gamma]}$ a(α,length[p_3]+1,a(β,[if bvalue[bl[b],ξ] then

[if bvalue[b2[b],ξ] then 1 else -1],θ))

$\equiv_{t[\gamma]}$ a(α,length[p_3]+1,a(β,[if bvalue[b,ξ] then 1 else -1],θ)) (3.10)

$\equiv_{t[\gamma]}$ a(α,length[p*compbool[b,k]]+1,a(β,[if bvalue[b,ξ]

then 1 else -1],θ))

Case (c): isnot[b]. Here,

$Z \equiv$ execobject[p*compbool[b3[b],k]*mkchs[ ]*q,length[p]+1,

length[compbool[b,k]],θ] (7.5)

Let p_1 = p*compbool[b3[b],k], $p_2 = p_1$*mkchs[ ] so that $\omega = p_2$*q,
and let m = length[compbool[b,k]]. Then,

$Z \equiv$ execobject[ω,length[p]+1,m,

execobject[ω,length[p]+1,length[compbool[b3[b],k]],θ]] (8.2)

With the induction hypothesis that the theorem holds for
b3[b], we have

$Z \equiv$ execobject[ω,length[p]+1,m,δ_1] where

$\delta_1 \equiv_{t[\gamma]}$ a(α,length[p_1]+1,a(β,[if bvalue[b3[b],ξ]

then 1 else -1],θ)). Then,

$Z \equiv$ execobject[ω,length[p]+1,m,execobject[ω,length[p_1]+1,1,δ_1]] (8.2)

$Z \equiv$ execobject[ω, $\underline{\text{length}}$[p]+1, m, stepobject[mkchs[ ], δ_1]] (2.8) (6.5)

$\equiv$ execobject[ω, $\underline{\text{length}}$[p]+1, m, δ_2] where

$\delta_2 \equiv a(\alpha, 1+c(\alpha, \delta_1), a(\beta, -c(\beta, \delta_1), \delta_1))$ (6.6)

$\equiv a(\alpha, \underline{\text{length}}[p_2]+1, a(\beta, -[\underline{\text{if}}\ \text{bvalue}[b3[b], \xi]$
$\underline{\text{then}}\ 1\ \underline{\text{else}}\ -1], \delta_1))$ (2.7) (3.1)

$\equiv_{t[\gamma]} a(\alpha, \underline{\text{length}}[p_2]+1, a(\beta, [\underline{\text{if}}\ \text{bvalue}[b3[b], \xi]$
$\underline{\text{then}}\ -1\ \underline{\text{else}}\ 1], \theta))$ (9.3) (9.5) (3.2)

Note that $\underline{\text{length}}[p_2]+1 = \underline{\text{length}}[p*\text{compbool}[b,k]]+1$

$= \underline{\text{length}}[p]+1 + \underline{\text{length}}[\text{compbool}[b,k]]$ (2.7)

$= \underline{\text{length}}[p]+1+m$ so that

$Z \equiv \underline{\text{if}}\ c(\alpha, \delta_2) < \underline{\text{length}}[p]+1 \vee c(\alpha, \delta_2) \geq \underline{\text{length}}[p]+1+m\ \underline{\text{then}}\ \delta_2\ \underline{\text{else}} \ldots$ (6.5)

$\equiv \underline{\text{if}}\ \underline{\text{length}}[p_2]+1 < \underline{\text{length}}[p]+1 \vee \underline{\text{length}}[p_2]+1 \geq \underline{\text{length}}[p_2]+1$
$\underline{\text{then}}\ \delta_2\ \underline{\text{else}} \ldots$ (3.1)

$\equiv \delta_2$

$\equiv_{t[\gamma]} a(\alpha, \underline{\text{length}}[p*\text{compbool}[b,k]]+1, a(\beta, [\underline{\text{if}}\ \text{bvalue}[b3[b], \xi]$
$\underline{\text{then}}\ -1\ \underline{\text{else}}\ 1], \theta))$

$\equiv_{t[\gamma]} a(\alpha, \underline{\text{length}}[p*\text{compbool}[b,k]]+1, a(\beta, [\underline{\text{if}}\ \text{bvalue}[b, \xi]$
$\underline{\text{then}}\ 1\ \underline{\text{else}}\ -1], \theta))$ (3.10)

This completes the proof of theorem (9.10).

Recall that we are considering the case isconditional[s], where $s = \underline{\text{select}}\ [c(\alpha, \xi), \gamma]$, and we want to derive an expression for execobject[ω, $c(\alpha,\theta)$, n, θ] $\equiv$ Y where $\omega = p*\text{compstate}[s, c(\alpha,\xi)]*q$, $n = \underline{\text{length}}[\text{compstate}[s, c(\alpha,\xi)]]$ and $c(\alpha,\theta) = \underline{\text{length}}[p]+1$. Then,

$Y \equiv$ execobject[$p*\text{compstate}[s, c(\alpha,\xi)]*q, c(\alpha,\theta), n, \theta$]

$\equiv$ execobject[$p*\text{compbool}[\text{premise}[s], f[c(\alpha,\xi)]]*\text{mktmi}[f[c(\alpha,\xi)+2]]*q,$
$c(\alpha,\theta), n, \theta$] (7.3)

$\equiv$ execobject[$\omega, c(\alpha,\theta), n,$ execobject[$\omega, c(\alpha,\theta), n-1, \theta$]] where (8.2)

$n-1 = \underline{\text{length}}[\text{compbool}[\text{premise}[s], f[c(\alpha,\xi)]]]$. Then, (2.7)

$Y \equiv$ execobject[$\omega, c(\alpha,\theta), n, \delta_1$] where

$\delta_1 \equiv_{t[\gamma]} a(\alpha, \underline{\text{length}}[p]+n-1+1, a(\beta, [\underline{\text{if}}\ \text{bvalue}[b, \xi]$
$\underline{\text{then}}\ 1\ \underline{\text{else}}\ -1], \theta))$. Then, (2.7) (9.14)

$Y \equiv$ execobject[$\omega, c(\alpha,\theta), n,$ execobject[$\omega, \underline{\text{length}}[p]+n, 1, \delta_1$]] (8.2)

$Y \equiv$ execobject[$\omega, c(\alpha,\theta), n, \delta_2$] where

$\delta_2 \equiv a(\alpha, [$ if $c(\beta,\delta_1) < 0$ then dest[mktmi[f[$c(\alpha,\xi)$+2]]]
else $1+c(\alpha,\delta_1)], \delta_1)$ (2.8) (6.5) (6.6)

$\equiv a(\alpha, [$ if [if bvalue[b,ξ] then 1 else -1] < 0 then
f[$c(\alpha,\xi)$+2] else 1+length[p]+n], δ_1) (3.1) (5.7)

$\equiv_{t[\gamma],\beta} a(\alpha, [$ if bvalue[b,ξ] then length[p]+n+1 else
f[$c(\alpha,\xi)$+2]], θ). Then, (3.2) (9.3) (9.5)

$Y \equiv$ if $c(\alpha,\delta_2) < c(\alpha,\theta) \vee c(\alpha,\delta_2) \geq c(\alpha,\theta)$+n then δ_2 else ... (6.5)

$\equiv \delta_2$ (3.1)

$\equiv_{t[\gamma],\beta} a(\alpha, [$ if bvalue[b,ξ] then length[p*compstate[s,$c(\alpha,\xi)$]]+1
else f[$c(\alpha,\xi)$+2]], θ) (2.7)

$\equiv_{t[\gamma],\beta} a(\alpha, [$ if bvalue[b,ξ] then f[$c(\alpha,\xi)$+1] else f[$c(\alpha,\xi)$+2]], θ) (9.11)

$\equiv_{t[\gamma],\beta} a(\alpha, f[[$ if bvalue[b,ξ] then 1 else 2]+$c(\alpha,\xi)$], θ)

This completes the derivation of execobject[$\omega, c(\alpha,\theta), n, \theta$] $\equiv Y$ for all three cases istransfer[s], isassignment[s] and isconditional[s]. The results for these three may be combined to give

(9.16) execobject[$\omega, c(\alpha,\theta), n, \theta$] $\equiv_{t[\gamma],\beta}$

if istransfer[s] then
[if f[destination[s]]$\neq c(\alpha,\theta)$ then $a(\alpha, f[$destination[s]], $\theta)$
else execobject[$\omega, c(\alpha,\theta), n, \theta$]]
else if isassignment[s] then
$a(\alpha, f[c(\alpha,\xi)+1], a(loc[$leftside[s]], value[rightside[s],ξ], θ))
else if isconditional[s] then
$a(\alpha, f[[$ if bvalue[b,ξ] then 1 else 2]+$c(\alpha,\xi)$], θ)

We will now use this result for execobject[$\omega, c(\alpha,\theta), n, \theta$] and the result given by (9.12) for execsource[$\gamma, c(\alpha,\xi), 1, \xi$] to carry out the final steps in the proof of parts (i) and (ii) of theorem (8.9), correctness of the compiler for a single statement.

Part (i) states that

f[$c(\alpha$, execsource[$\gamma, c(\alpha,\xi), 1, \xi$])] = $c(\alpha$, execobject[$\omega, c(\alpha,\theta), n, \theta$]).

Let us compute

f[$c(\alpha$, execsource[$\gamma, c(\alpha,\xi), 1, \xi$])] =

if istransfer[s] then
  [if f[destination[s]]≠c(α,θ) then f[destination[s]]
     else f[c(α,execsource[γ,c(α,ξ),1,ξ])]]
else if isassignment[s] then f[c(α,ξ)+1]
else if isconditional[s] then
   f[[if bvalue[b,ξ] then 1 else 2]+c(α,ξ)] (3.1) (9.12)

In deriving this result, the one-to-one nature of f[k] allows us to write: destination[s]≠c(α,ξ) ⇔ f[destination[s]]≠f[c(α,ξ)]
⇔ f[destination[s]]≠c(α,θ)
The second result follows from c(α,θ) = f[c(α,ξ)], given in the statement of theorem (8.9).

Now let us compute
c(α,execobject[ω,c(α,θ),n,θ]) =
if istransfer[s] then
  [if f[destination[s]]≠c(α,θ) then f[destination[s]]
     else c(α,execobject[ω,c(α,θ),n,θ])]
else if isassignment[s] then f[c(α,ξ)+1]
else if isconditional[s] then
   f[[if bvalue[b,ξ] then 1 else 2]+c(α,ξ)] (3.1) (9.16)

Comparing the two results just derived, we obtain part (i) of theorem (8.9). Note that if s is a transfer statement to itself, then both sides of (i) are undefined.

Part (ii) states that
c(v,execsource[γ,c(α,ξ),1,ξ]) = c(loc[v],execobject[ω,c(α,θ),n,θ])

Let us compute
c(v,execsource[γ,c(α,ξ),1,ξ]) =
if istransfer[s] then
  [if destination[s]≠c(α,ξ) then c(v,ξ)
     else c(v,execsource[γ,c(α,ξ),1,ξ])]
else if isassignment[s] then
  [if v=leftside[s] then value[rightside[s],ξ] else c(v,ξ)]
else if isconditional[s] then c(v,ξ) (3.1) (3.2) (9.12)

Now let us compute

c(loc[v],execobject[ω,c(α,θ),n,θ]) =

if istransfer[s] then

[if destination[s]≠c(α,ξ) then c(v,ξ)

else c(loc[v],execobject[ω,c(α,θ),n,θ])]

else if isassignment[s] then

[if v=leftside[s] then value[rightside[s],ξ] else c(v,ξ)]

else if isconditional[s] then c(v,ξ) (3.1) (3.2) (9.16)

In deriving this result, the one-to-one nature of f[k] allows us to write: f[destination[s]]≠c(α,θ) ⇔ destination[s]≠c(α,ξ) as discussed above. As well, the one-to-one nature of loc[v] allows us to write: loc[v]=loc[leftside[s]] ⇔ v=leftside[s]. We have also used the fact c(loc[v],θ) = c(v,ξ) given in the statement of theorem (8.9).

Comparing the two results just derived, we obtain part (ii) of theorem (8.9). Note that if s is a transfer statement to itself, then both sides of (ii) are undefined.

With theorem (8.9) proven, the correctness of the compiler as stated in theorem (8.1) is therefore proven.

10. Concluding Remarks

The rather primitive compiling algorithm considered in this paper is but a first step in the formulation of usable and useful correct compilers. Nevertheless, we have to some degree advanced toward that goal. It is easy to envision the formal framework set up in this paper serving adequately for a compiler for far more sophisticated source and object languages. Extensions to include other arithmetic and logical operations are straightforward, and the introduction of character manipulation would not be much more difficult. The semantics of more structured control statements (DO, WHILE, etc.) could be defined, not by English paragraphs, but rather in terms of the well understood primitive statements for which we already have a valid proof of correctness. In fact, if results could be obtained for a reasonably elegant source language, then the compiler itself could

be written in the source language. Compilation of this would give a machine language version of the compiler which is "correct". Thus, not only would the compiling algorithm be proven correct, but the machine language implementation would also be correct.

[1] McCarthy, J., Computer Programs for Checking Mathematical Proofs, Recursive Function Theory, Proceedings of Symposia in Pure Mathematics, Vol. 5, American Mathematical Society (1962).

[2] McCarthy, J., A Basis for a Mathematical Theory of Computation, Computer Programming and Formal Systems, Edited by P. Brattford and D. Hershberg, North-Holland, Amsterdam (1963).

[3] McCarthy, J., Towards a Mathematical Theory of Computation, Proceedings of 1962 International Congress on Information Processing.

[4] Kaplan, D. M., Some Completeness Results in the Mathematical Theory of Computation, Artificial Intelligence Memo No. 45, Stanford University, 1966.

[5] McCarthy, J., and Painter, J., Correctness of a Compiler for Arithmetic Expressions, Artificial Intelligence Memo No. 40, Stanford University, 1966.

[6] Kaplan, D. M., On the Notion of Correct Compilers, CS224 Term Research Paper, Stanford University, 1966.

[7] Painter, J., Semantic Correctness of a Compiler for an ALGOL-Like Language, Artificial Intelligence Memo No. 44, Stanford University, March 10, 1967.