Errata K. Snidvongs Ph.D. Thesis

Page 3 line 13 should read

6.C GOTO Statement

Page 12 line 19 should read

P2. S(Sabc)bd = Sab(Scbd)

Page 16 line 18 should read

$a \cong b$ may be written as S(STab)T(STba)

Page 65 line 1 should read

§ 3.C The Application of S to Boolean Algebra

Page 72 line 4 should read

..... and range $R \subseteq V$

Page 72 line 13 should read

Let X denote an unordered set of ordered pairs of

Page 73 line 1 insert

For $i \neq j$ and $n_i \neq n_j$, v_i and v_j are not necessarily distinct.

Page 87 line 12 should read

$= S\mu[kakb, kckd, kekb, kekd]$

Page 93 line 6 should read

(i) $\{a_1, a_2, \ldots, a_n\} \cup \{x_1, x_2, \ldots, x_n\} \subseteq \mu \cap \eta$

Page 120 line 18 should read

... such that $E \cap \{T, F\} = \emptyset$

Page 129 last line insert

This is considered in § 6.E .

Page 140 line 8 should start

$= S\mu[\,[ka_1kx_1 ; \ldots..$

Page 142 last line should read

are syntactically acceptable.

Page 149 line 16 should read

$S\mu[\rho Z ; \rho X]$ or $S\mu[\rho Z ; \rho Y]$ or $S\mu[\rho Z ; \omega]$

Page157 line 23 should read

L: if $x > 0$ then ....

Page 196 line 16 should read

....[(7b & 7c), page 184] ....

Page198 line 22 should read

... lattice theoretic models ....

Page 199 line 6 should read

... lattice theoretic consideration.

Page 203 line 6 should read

... Hartley, D. F. , ...

AN INVESTIGATION INTO AN OPERATOR BASIS FOR MATHEMATICAL SEMANTICS

by

Kamontip Snidvongs

Thesis submitted for the degree of Doctor of Philosophy of the University of London.

August 1975

ABSTRACT

In this thesis we develop an algebraic approach to the problems of formal semantics of programming languages and program equivalence.

Taking Wesselkamper's work [41] on a Mathematical Model of the Computing Process in a High Level Language in terms of the S and μ operators of Nixon and Wesselkamper [31, 32] as a starting point, we show that it is possible to model the various forms which occur in programming languages.

Algebraic theorems for the transformation and reduction of these forms are derived and a canonical representation of a program is defined.

Notions of equivalence in terms of these " canonical forms" are briefly investigated.

CONTENTS

		ACKNOWLEDGEMENT	4
CHAPTER 1		INTRODUCTION	5
CHAPTER 2		THE OPERATOR S OVER FINITE DISJOINT SPACES	11
	2.A	Introduction and Definitions	11
	2.B	The Operator S over a Two-Valued Space	12
	2.C	The Substitution operator S and the Propositional Calculus	15
	2.D	The Extension of S to a Generalised Finite-Valued Space	22
CHAPTER 3		THE OPERATOR S OVER FINITE NON-DISJOINT SPACES	35
	3.A	Definition and Basic Properties of S	35
	3.B	Further Properties of S	53
	3.C	The Application of S to Boolean Algebra	65
CHAPTER 4		SIMPLE PROGRAMS AND THEIR SPECIFICATIONS	72
	4.A	Names, Values and the Evaluation Operator ν	72
	4.B	A Model of a Sequence of Simple Assignment Statements	76
	4.C	A Dynamic Property of the Variables	82
	4.D	Transformation of a Dynamic into a Static Sequence	85
	4.E	Reduction of Static S-Sequences	88
	4.F	Equivalence of Sequences of Simple Assignment Statements	94
	4.G	Assignment Statements with Expressions on the Right-Hand Side	97

	4.H	Transformation of a Static into a Dynamic Sequence	102
	4.I	The Simplification of Programs	115
CHAPTER 5		MODEL OF A CONDITIONAL STATEMENT	118
	5.A	Model of a Conditional Statement	118
	5.B	Formal Properties of Conditional S-Sequences	120
	5.C	Canonical Form of Conditional S-Sequences	125
	5.D	Conditional S-Sequences with Undefined Value	130
	5.E	Conclusion	136
CHAPTER 6		ON CONTROL STRUCTURES	137
	6.A	Preamble	137
	6.B	The 'Value' of a Label	143
	6.C	GOTO Statement	146
	6.D	Conditional Statements	148
	6.E	Transformation of Programs with Control Statements	154
	6.F	Blocks	162
	6.G	Initialisation	176
	6.H	Procedures	186
CHAPTER 7		CONCLUSION	198
REFERENCES			203
APPENDIX A			
APPENDIX B			
APPENDIX C			
APPENDIX D			
APPENDIX E			
APPENDIX F			

ACKNOWLEDGEMENT

I wish to thank the British Council for their scholarship which enabled me to carry out this research.

I am most grateful to my supervisor, Mr. Eric Nixon, for his constructive comments and constant encouragement throughout the preparation of this thesis. His informal approach to supervising research students is most appreciated.

My thanks also go to Miss Jean Pontefract and Miss Isabel Anderson for the typing of this manuscript.

CHAPTER I

INTRODUCTION

During the past few years, investigations have been made into the possible utilisation of a Markov-like substitution operator S as a basis for a theory of computation.

Wesselkamper [41], in his doctoral thesis, constructed a mathematical model of the computing process using S as a primitive operator. He also showed that any computer, together with its stored programs, can be completely described in terms of this single operator.

Maile [26], in her study of the use of S as a basis for logic design, pointed out that for each logic, arithmetic or control operation the number of S gates required is always less than or equal to the number of NAND gates.

In none of this previous work, however, was any attempt made to reduce the complicated expressions which resulted to a more manageable form.

In Wesselkamper's model, the expressions arising from even a simple program are quite formidable. If such a model is to serve as a basis for the investigation of program equivalence, then one essential process is to decide whether two expressions, which have different forms, are equivalent. For such a decision procedure to be possible, some form of calculus must be devised to facilitate the necessary manipulations and possible reductions of the S forms.

The work presented herein is an attempt to formulate such a calculus.

We begin in Chapter II by giving the definition of S over a general finite-valued space of disjoint elements. From this definition it is found that, over a simple two-valued space, S has the following properties:-

1. S is commutative
2. S is associative
3. S is distributive.

From these properties, a set of reduction rules for S is obtained. Then, by considering the space {T,F} as a particular two-valued space, the propositional calculus is shown to be embedded within the S system.

Next we examine how far these properties and reduction rules apply over a general n-valued space. It is found that the commutativity and some of the distributive laws no longer, in general, hold. Nevertheless, the same set of reduction rules is still applicable.

An obvious generalisation of the results of Chapter II is to consider the properties of S over a space in which the elements are not necessarily disjoint. In Chapter III, it is shown that over such a space the associativity of S still holds unconditionally, whilst the other basic properties apply only under certain conditions. On the basis of these results, we give a set of reduction rules for S acting over a generalised space; some of these rules appear in slightly different forms from those derived for the disjoint space.

Further, by considering S in connection with the basic operators of set theory, it is found that the operator S and set-union and set-intersection are distributive over each other.

As an application of the theorems obtained, we derive the axioms of a Boolean algebra from the properties of S. It then follows from the isomorphism of Boolean algebras and the algebra of sets that this also can be completely represented in terms of S. Hence, we conclude that Lattice Theory can also be formulated within the framework of S.

Having formulated the algebraic rules for manipulating expressions involving the S operator, we next investigate the possibility of applying these rules in the study of program equivalence. This is carried out in the following chapters.

Firstly, we introduce the notion of name and value in a program. An evaluation operator μ is defined acting upon a name to give the corresponding value. We thus represent each statement in a program as a successive alteration of values, using the operator μ to yield the values which act as operands for the operator S.

In this work, we model a program in terms of a composite S-expression which represents the sequential and control properties of that program. The details of operations are suppressed so as not to obscure the approach by detail. Our primary concern is to show that the approach is feasible rather than to concentrate on elegance or completeness.

In our model, we regard each statement of a program as a modification of a 'variable space' μ, containing all the variables of the program. We start, in Chapter IV, by setting up a model of a simple program, ie. a program consisting purely of assignment statements, in the form of a composite S-expression. This expression is then reduced to a canonical form to facilitate comparison with other programs.

As the effect of the evaluation of a given S-expression is, in general, to modify the variable space μ, any following S-expression must be considered relative to the new (modified) variable space. From this, it is obvious that the variable as used in programming languages differs from that in Mathematics, being dynamic in nature whilst that of conventional Mathematics is static.

For this reason, we call an S-sequence which is obtained directly from the S-expression representing a program a 'dynamic' S-sequence. Such a sequence has to be transformed into a 'static' one before any algebraic rule can be applied to it. Some of the properties of S derived for general set operands in Chapter III are found to be useful for the reduction of the static sequence. Further results which are directly applicable to the particular case of this chapter are also developed. This enables us to show how a simple program may be transformed to the canonical form and thus a simple decision procedure for determining the equivalence of simple programs is obtained.

In order to construct a model of a more realistic program, control structures have to be considered. Among these, conditional statements play an important part. We devote Chapter V to investigations concerned with conditional statements. Firstly, we model conditional statements in terms of S-expressions, then derive two theorems which may be applied to reduce any such model to a canonical conditional form. Further theorems are then developed for the transformation of a canonical conditional S-sequence into its minimal form.

The results of Chapter V and the transformations and reductions of Chapter IV are then used, in Chapter VI, as a basis for the consideration of more realistic programs.

Any program may, in general, be considered as sequences of assignment statements separated by control statements. If we regard the whole program as a single sequence of statements, each part of the program which consists purely of assignment statements may be regarded as a subsequence of the entire sequence. Therefore, if the control statements can be modelled in terms of the operator S , the whole program can then be represented as a dynamic sequence consisting of a number of subsequences each of which may be transformed and reduced to canonical form by the results of Chapter IV. When a program is executed, the values of the variables concerned are changed by a series of assignments with control statements acting as an indication of the order in which the operations are to be carried out. Thus, to obtain the canonical form of the dynamic sequence representing a general program, we consider the whole sequence as a concatenation of a number of subsequences, the order of these subsequences being governed by the control statements, and perform the transformation by the reduction of these subsequences. Such a transformation is considered in the first section of Chapter VI.

Next, we consider the most basic form of control statement, ie. the GOTO statement. Despite current efforts to eliminate the GOTO statement from programs written in high-level languages, its effect is still present in other control statements albeit in a concealed form. As our investigation concerns the actual effect of control statements, the GOTO statement becomes an essential part. Together with GOTO statements, labels have to be considered; we specify an interpretation of the 'value of a label' and construct a model of the GOTO statement.

Then, we set up a model of a conditional statement. This leads us to the consideration of generalised assignments and loops.

So far, we have modelled the types of statement which are common features of all programming languages. At this point, we illustrate the transformation of a program containing control statements utilising the transformation rules for conditional forms which have been considered in detail in Chapter V.

In the next section we consider the concept of block structure. Our treatment involves augmentation and deletion of variables to represent the effects of variables being declared at the beginning of the block and their ceasing to exist upon block exit. The scope of variables is also characterised in the model and the question of irregular exit from the block is discussed briefly. At the end of the section, a procedure for the transformation of dynamic sequences containing blocks is given.

Two types of initialisation, namely initialisation by value and initialisation by substitution, are then investigated. This necessitates the consideration of name and value transformation, the range of such a transformation and the reduction of the transformed sequence.

Finally, a model of a procedure is established. Problems arising in the transformation of programs containing procedure calls are discussed and solutions are proposed. Using the model given, recursive procedures do not present any further problems. Examples to illustrate the transformation of programs containing procedures are given in appendices D,E,F.

CHAPTER 2

THE OPERATOR S OVER FINITE DISJOINT SPACES

§2.A Introduction and Definitions

Markov [28] defined the substitution operator Σ as a triadic operator acting upon strings.

If P,Q,R are strings and Q occurs in P, then $\Sigma(P,Q,R)$ is defined as the result of replacing the leading occurrence of Q in P by R. If Q does not occur in P, $\Sigma(P,Q,R)$ is undefined.

The substitution operator S is a generalisation of Σ. Its operands may be structures and it is always defined.

In general terms, $Sabc$ means "in a, substitute c for b".

For a,b,c belonging to an n-valued space $\{v_1, \ldots, v_n\}$, S is defined by:

$$Sabc = \begin{cases} c & \text{if } a = b \\ a & \text{if } a \neq b \,. \end{cases}$$

The terms S-expression, truth-table and equivalence used in this work are defined as follows:

Definition — An S-expression is an expression of the form $Sabc$ where a,b,c are either single variables or S-expressions.

Definition — The truth-table for an S-expression is a table giving the values of the expression for all possible assignments of values to the operands occurring in the expression.

Definition Two S-expressions Sabc and Sxyz are said to be equivalent, written Sabc = Sxyz, if they have the same truth table.

§2.B. The Operator S over a Two-Valued Space

Consider the operator S over a simple two-valued space $\{v_1, v_2\}$. From the definition

$$Sabc = \begin{cases} c & \text{if } a = b \\ a & \text{if } a \neq b \end{cases},$$

it is clear that for all $a, b \in \{v_1, v_2\}$, S has the following basic properties:

$$\left.\begin{aligned} Saba &= a \\ Saab &= b \\ Sabb &= a \end{aligned}\right\} \quad P0$$

From the above properties, it follows that over any two-valued space, S is commutative, associative and distributive.

Theorem 2.1

For $a,b,c,d,f \in \{v_1, v_2\}$, the following equations are identities:

P1. $Sabc = Scba$

P2. $S(Sabc)bd = Sab(Scbd)$

P3. $Sd(Sabc)f = S(Sdaf)(Sdbf)(Sdcf)$

P4. $Sfd(Sabc) = S(Sfda)(Sfdb)(Sfdc)$

Proof

P1. $Sabc = Scba$

$a = b \Rightarrow LHS = c = RHS$

If $a \neq b$, then either $c=a$ or $c=b$.

$a \neq b$, $c = a \Rightarrow$ LHS $= a =$ RHS

$a \neq b$, $c = b \Rightarrow$ LHS $= a =$ RHS

P2. $\quad S(Sabc)bd = Sab(Scbd)$

$a = b \Rightarrow$ LHS $= Scbd =$ RHS

$a \neq b \Rightarrow$ LHS $= Sabd = a =$ RHS

P3. $\quad Sd(Sabc)f = S(daf)(Sdbf)(Sdcf)$

$a = b \Rightarrow$ LHS $= Sdcf$

RHS $= S(Sdaf)(Sdaf)(Sdcf) = Sdcf$

If $a \neq b$, then $c = a$ or $c = b$.

$a \neq b$, $c = a$

$\Rightarrow$ LHS $= Sdaf$

RHS $= S(Sdaf)(Sdbf)(Sdaf) = Sdaf$

$a \neq b$, $c = b$

$\Rightarrow$ LHS $= Sdaf$

RHS $= S(Sdaf)(Sdbf)(Sdbf) = Sdaf$

P4. $\quad Sfd(Sabc) = S(Sfda)(Sfdb)(Sfdc)$

$f = d \Rightarrow$ LHS $= Sabc =$ RHS

$f \neq d \Rightarrow$ LHS $= f$

RHS $= Sfff = f$.

Note From the properties P1 and P4, we can deduce the other distributive property of S, namely

P5. $\quad S(Sabc)df = S(Sadf)(Sbdf)(Scdf)$

Proof By the application of P1 to the identity P4.

Reduction Rules for S-Expressions

From the properties P0 to P5, a set of reduction rules for S expressions can be derived.

Theorem 2.2

For $a,b,c \in \{v_1, v_2\}$, the following statements are valid:

R1. $S(Sabc)ab = Sbac$

$S(Sabc)cb = Sacb$

R2. $S(Sabc)bc = Sabc$

$S(Sabc)ba = Sabc$

R3. $S(Sabc)ca = a$

$S(Sabc)ac = c$

R4. $Sb(Sabc)a = b$

$Sb(Sabc)c = b$

Proof R1. $S(Sabc)ab = Sba(Sabc)$ by P1

$= S(Sbaa)(Sbab)(Sbac)$ by P4

$= Sbb(Sbac)$ by P0

$= Sbac$ by P0

$S(Sabc)cb = S(Scba)cb$ by P1

$= Sbca$ from the above result

$= Sacb$ by P1

R2. $S(Sabc)bc = Sab(Scbc)$ by P2

$= Sabc$ by P0

$S(Sabc)ba = S(Scba)ba$ by P1

$= Scba$ from the above result

$= Sabc$ by P1

R3. $S(Sabc)ca = Sac(Sabc)$ by P1

$= S(Saca)(Sacb)(Sacc)$ by P4

$= Sa(Sacb)a$ by P0

$= a$ by P0

$S(Sabc)ac = S(Scba)ac$ by P1

$= c$ from the above result

R4. $Sb(Sabc)a = S(Sbaa)(Sbba)(Sbca)$ by P3

$= Sba(Sbca)$ by P0

$= S(Sbca)ab$ by P1

$= b$ by R3

$Sb(Sabc)c = Sb(Scba)c$ by P1

$= b$ from the above result

Remark

Any complex expression which we may be required to manipulate neccessarily has one of the following structures:

(i) $S(Sabc)df$

(ii) $Sd(Sabc)f$

(iii) $Sfd(Sabc)$

where a,b,c,d and f may themselves be S expressions.

§2.C. The Substitution Operator S and the Propositional Calculus

Consider the space $\{T,F\}$ as a special case of the two-valued space $\{v_1, v_2\}$. Over such a space, the relation of S to the propositional calculus may be considered.

Notation

Throughout this section, we employ the following symbols for logical connectives:

$\sim$ for "not"

$\wedge$ for "and"

$\vee$ for "or"

$\supset$ for "implies"

$\equiv$ for "equivalent"

From the definition of S, i.e.

$$Sabc = \begin{cases} c & \text{if } a = b \\ a & \text{if } a \neq b \end{cases}$$

where $a,b,c \in \{T,F\}$, the complete two-valued propositional calculus may be formulated in terms of S and the constants T and F, since

$\sim a$	may be written as	$STaF$	(*)
$a \wedge b$	may be written as	$SaTb$	
$a \vee b$	may be written as	$SaFb$	
$a \supset b$	may be written as	$STab$	
$a \equiv b$	may be written as	$S(STab)T(STb\bar{a})$	
and $a \neq b$	may be written as	$S(SFab)F(SabF)$	

The above set of correspondents is fully justified, for the the truth table of each S-expression is the same as the truth table of its corresponding logical connective. (The associated truth tables are given in Appendix A.)

Remark

In formulating the above logical connectives, we aim to express them in the simplest possible way, ie. to use the shortest possible

S-expressions corresponding to the connectives. Except for equivalence and non-equivalence, all the connectives considered may be expressed using only one S operator. Note that the given representations of the connectives are not unique, eg. STaF is given as a representation of $\sim a$ but the commutability of S on the two-valued space implies that SFaT also has the required truth table.

The case of $a \equiv b$ gives a more complicated example of several S-expressions corresponding to the same logical connective. As illustrated in Appendix A, the expressions

S(SaTb)a(STba)

S(SaTb)(SaFb)T

may also be used to represent $a \equiv b$. The expression S(STab)T(STba) is adopted as the canonical representation of $a \equiv b$ because it can be directly constructed from the S equivalent of $(a \supset b) \wedge (b \supset a)$.

Under the above formulation of the logical connectives, we can prove that the complete propositional calculus is embedded within the definition of S.

To substantiate this claim, we show that the axioms and rules of inference for an axiom system of the propositional calculus can be obtained from S.

We adopt the axiom system for the propositional calculus given by Mendelson [27].

A formal axiomatic theory L for the propositional calculus is defined by the following conditions:

(1) The symbols of L are $\sim$, $\supset$, (,), and the letters A_i with positive integers i as subscripts:

A_1, A_2, A_3, ... The symbols $\sim$ and $\supset$ are called primitive connectives, and the letters A_i are called statement letters.

(2) a. All statement letters are well formed formulae (wffs).

b. If a and b are wffs, so are $(\sim a)$ and $(a \supset b)$.

Thus a wff of L is just a statement form built up from the statement letters A_i by the connectives $\sim$ and $\supset$.

(3) If a, b and c are wffs of L, then the following are axioms of L

(A1) $(a \supset (b \supset a))$

(A2) $((a \supset (b \supset c)) \supset ((a \supset b) \supset (a \supset c)))$

(A3) $((\sim b \supset \sim a) \supset ((\sim b \supset a) \supset b))$

(4) The only rule of inference of L is modus ponens:

b is a direct consequence of a and $a \supset b$.

Other connectives are introduced by the following definitions

(D1) $(a \wedge b)$ for $\sim (a \supset \sim b)$

(D2) $(a \vee b)$ for $(\sim a) \supset b$

(D3) $(a \equiv b)$ for $(a \supset b) \wedge (b \supset a)$

Modus ponens can then be written as

$$((a \wedge (a \supset b)) \supset b).$$

Note Each of Axiom Schemas (A1) - (A3) is independent as is proved in Mendelson [27].

In order to show that (A1), (A2), (A3) and modus ponens follow directly from the properties of S, we first write them as S-expressions according to the formulation (*), and then prove that the expressions obtained are equivalent to the constant T.

We first prove the following lemmas:

Lemma 1 $SaF(STaF) = T$

Proof $SaF(STaF) = STFa$ by P1 and R1

$= T$

Lemma 2 $ST(STaF)F = a$

Proof $ST(STaF)F = S\{SaF(STaF)\}(STaF)F$ from Lemma 1

$= Sa(STaF)F$ by R1

$= a$ by R4

Theorem 2.3 The following statements are valid

(i) $STa(STba) = T$ (A1)

(ii) $ST\{STa(STbc)\}\{ST(STab)(STac)\} = T$ (A2)

(iii) $ST\{ST(STbF)(STaF)\}\{ST(ST(STbF)a)b\} = T$ (A3)

(iv) $ST(SaT(STab))b = T$ (Modus ponens)

Proof

(i) $STa(STba) = S(STba)aT$ by P1

$= T$ by R3

(ii) $ST\{STa(STbc)\}\{ST(STab)(STac)\}$

$= ST\{S(STaT)(STab)(STac)\}\{ST(STab)(STac)\}$ by P4

$= ST\{ST(STab)(STac)\}\{ST(STab)(STac)\}$ by P0

$= T$ by P0

(iii) $ST\{ST(STbF)(STaF)\}\{ST(ST(STbF)a)b\}$

$= ST\{S(ST(STbF)T)(ST(STbF)a)(ST(STbF)F)\}\{ST(ST(STbF)a)b\}$ by P4

$= ST\{ST(ST(STbF)a)b\}\{ST(ST(STbF)a)b\}$ by P0 and Lemma 2

$= T$ by P0

(iv) $ST(SaT(STab))b$

$= ST(S(SaTT)(SaTa)(SaTb))b$ by P4

$= ST(Saa(SaTb))b$ by P0

$= ST(SaTb)b$ by PO

$= T$ by R4

From the formulation (*) of the logical connectives and the definitions (D1) and (D2), we obtain the following identities:

$$SaTb = ST(STa(STbF))F$$

and $$SaFb = ST(STaF)b$$

which could be used to simplify S-expressions.

Note that in the process of proving (iii) in theorem 2.3, we have also obtained two more reduction rules, namely

(a) $SaF(STaF) = T$ (Lemma 1)

and (b) $ST(STaF)F = a$ (Lemma 2)

The list below gives further additions to the collection of reduction rules for S-expressions:

(c) $SaT(STaF) = F$

(d) $SaF(SaTb) = a$

(e) $SaT(SaFb) = a$

(f) $ST(SaFb)F = S(STaF)T(STbF)$

(g) $ST(SaTb)F = S(STaF)F(STbF)$

Written in terms of logical connectives, the above results are:

(c) $a \wedge (\sim a) = F$

(d) $a \vee (a \wedge b) = a$ Absorption Laws

(e) $a \wedge (a \vee b) = a$

(f) $\sim (a \vee b) = (\sim a) \wedge (\sim b)$ De Morgan's Laws

(g) $\sim (a \wedge b) = (\sim a) \vee (\sim b)$

These laws are among the theorems of L , and hence derivable from the axioms (A1) - (A3).

They are, however, more easily obtained from the properties of S as shown in the following examples.

Examples

Prove that the following equations hold:

(i) $ST(SaFb)F = S(STaF)T(STbF)$

(ii) $SaF(SaTb) = a$

Proof

(i)
$$\begin{aligned} ST(SaFb)F &= S(STaF)(STFF)(STbF) && \text{by P3} \\ &= S(STaF)T(STbF) && \text{by P0} \end{aligned}$$

(ii)
$$\begin{aligned} SaF(SaTb) &= S(SaTb)Fa && \text{by P1} \\ &= S(SaTb)F(SaTT) && \text{by P0} \\ &= S(SaTb)F\{SaT(SbF(STbF))\} && \text{by (a)} \\ &= S(SaTb)F\{S(SaTb)(SaTF)(SaT(STbF))\} && \text{by P4} \\ &= S(SaTb)F\{S(SaTb)F(SaT(STbF)\} && \text{by P1 and defn.} \\ &= S\{S(SaTb)F(SaTb)\}F\{SaT(STbF)\} && \text{by P2} \\ &= S(SaTb)(SaTF)(SaT(STbF)) && \text{by P0} \\ &= SaT\{SbF(STbF)\} && \text{by P4} \\ &= SaT(STFb) && \text{by P1 \& R1} \\ &= SaTT && \text{by definition} \\ &= a && \text{by P0} \end{aligned}$$

(i) and (ii) give the proofs of (f) and (d); the proofs of (g) and (e) follow by interchanging T and F in (i) and (ii) respectively.
(c) follows from (a) in a similar manner.

§2.D The Extension of S to a Generalised Finite-Valued Space

Now consider the more general case in which the operands of S belong to a finite n-valued space$\{v_1, \ldots, v_n\}$.

Under such a generalisation, some of the properties stated in 2.B are no longer valid.

For example, let $a,b,c \in \{T,F,\omega\}$ for some $\omega \notin \{T,F\}$. For $a = T$, $b = F$ and $c = \omega$, we have

$$Sabc = STF\omega = T \quad \text{and}$$

$$Scba = S\omega FT = \omega$$

ie. $Sabc \neq Scba$.

We now examine the conditions for validity of the properties P0 to P4.

Theorem 2.4

For $a,b \in \{v_1, \ldots, v_n\}$, the following properties are valid:

$$Saba = a$$

$$Saab = b$$

and $$Sabb = a$$

Proof Obvious from the definition of S.

Theorem 2.5

For $a,b,c \in \{v_1, \ldots, v_n\}$,

$$Sabc = Scba$$

unless a,b,c are all different.

Proof $a = b \Rightarrow LHS = c$

$RHS = c$ if $b \neq c$

$= a$ if $b = c$ but then $a = b = c$

$\therefore$ RHS = c

ie. $a = b \Rightarrow LHS = RHS$.

So $a \neq b$ is a necessary condition for $Sabc \neq Scba$.

Similarly, $b = c \Rightarrow LHS = RHS = a$.

So $b \neq c$ is a necessary condition for $Sabc \neq Scba$.

$a = c \Rightarrow Sabc = Saba = Scba$

So $LHS \neq RHS$ iff $a \neq c$.

Hence $Sabc \neq Scba$ iff a,b,c are all different.

Theorem 2.6

$S(Sabc)bd = Sab(Scbd)$ for all

$$a,b,c,d \in \{v_1, \ldots, v_n\} .$$

Proof $a = b \Rightarrow LHS = Scbd = RHS$

$a \neq b \Rightarrow LHS = Sabd = a = RHS$.

Theorem 2.7

$Sd(Sabc)f = S(Sdaf)(Sdbf)(Sdcf)$ unless the following conditions hold simultaneously:

(i) a,b,c are all different

(ii) $c = d$

(iii) $d \neq f$.

Proof $a = b \Rightarrow LHS = Sdcf$

$RHS = S(Sdaf)(Saaf)(Sdcf)$

$= Sdcf$

$\therefore$ $a \neq b$ is a necessary condition for inequality

$a = c \Rightarrow LHS = Sdaf$

$RHS = S(Sdaf)(Sdbf)(Sdaf)$

$= Sdaf$

$\therefore$ $a \neq c$ is a necessary condition for inequality

$b = c \Rightarrow$ LHS = Sdaf

RHS = S(Sdaf)(Sdbf)(Sdbf)

= Sdaf

$\therefore$ $b \neq c$ is a necessary condition for inequality

$d = f \Rightarrow$ LHS = d = RHS

$\therefore$ $d \neq f$ is a necessary condition for inequality.

If a,b,c are all different and $d \neq f$, then

LHS = Sdaf

RHS = S(Sdaf)(Sdbf)(Sdcf).

Now LHS $\neq$ RHS iff

(a) Sdaf = Sdbf

ie. iff $d \neq a$, $d \neq b$

and

(b) Sdcf $\neq$ Sdaf

ie. Sdcf $\neq$ d (because $d \neq a$)

ie. iff d = c.

Thus LHS = RHS unless

(i) a,b,c are all different

(ii) c = d

(iii) $d \neq f$

Note that (i) and (ii) implies $d \neq a$, $d \neq b$.

Theorem 2.8

Sfd(Sabc) = S(Sfda)(Sfdb)(Sfdc) for all

$a,b,c,d,f \in \{v_1, \ldots, v_n\}$

Proof $f = d \Rightarrow LHS = Sabc = RHS$

$f \neq d \Rightarrow LHS = f = RHS$

Hence $LHS = RHS \ \forall \ a,b,c,d,f \in \{v_1, \ldots, v_n\}$.

Since $Sabc \neq Scba$ in general, the expressions $Sfd(Sabc)$ and $S(Sabc)df$ are no longer equivalent. Theorem 2.9 gives the distributive property of expressions of the form $S(Sabc)df$ and the conditions under which this property holds.

Theorem 2.9

$S(Sabc)df = S(Sadf)(Sbdf)(Scdf)$ except when

(i) a,b,c are all different

and either (iia) $a = d$ and $b = f$

or (iib) $a = f$ and $b = d$

Proof $d = f \Rightarrow LHS = Sabc = RHS$.

$\therefore$ $d \neq f$ is a necessary condition for $LHS \neq RHS$.

$a = b \Rightarrow LHS = Scdf = RHS$.

$\therefore$ $a \neq b$ is necessary.

$a = c \Rightarrow LHS = Sadf = RHS$.

$\therefore$ $a \neq c$ is necessary.

$b = c \Rightarrow LHS = Sadf = RHS$.

$\therefore$ $b \neq c$ is necessary.

Under the above conditions, $LHS = Sadf$

$RHS = S(Sadf)(Sbdf)(Scdf)$

$\therefore$ $LHS \neq RHS$ iff $Sadf = Sbdf$ - (1)

and $Sadf \neq Scdf$ - (2).

(1) is satisfied iff either $a = d,\ b = f$ (iia)

or $a = f,\ b = d$ (iib)

(iia) $\Rightarrow$ Sadf = f

Scdf = c $\neq$ f because a,b,c are different.

(iib) $\Rightarrow$ Sadf = f

Scdf = c $\neq$ f because a,b,c are different.

Hence (iia) and (iib) satisfy (2).

Note that the conditions "a,b,c are all different" and either "a = d, b = f" or "a = f, b = d" already include the necessary condition $d \neq f$.

Hence the property above holds except when

(i) a,b,c are all different

and (iia) a = d, b = f

or (iib) a = f, b = d

Remark The condition "a,b,c are all different", which is necessary for the properties in theorems 2.5, 2.7, and 2.9 to be invalid, cannot hold in an operand space of fewer than three values. Hence those properties are unconditionally valid in any two-valued operand space.

We next examine the validity of the reduction rules R1 to R4 in 2.B.

Theorem 2.10

For $a,b,c \in \{v_1, \ldots, v_n\}$, the following statements are valid:

R1. S(Sabc)ab = Sbac

S(Sabc)cb = Sacb

R2. S(Sabc)bc = Sabc

S(Sabc)ba = Sabc

R3. $S(Sabc)ca = a$

$S(Sabc)ac = c$

R4. $Sb(Sabc)c = b$

$Sb(Sabc)a = b$

Proof R1. $S(Sabc)ab = Sbac$

$a=b \Rightarrow LHS = Scab = c = RHS$

$a\neq b \Rightarrow LHS = Saab = b = RHS.$

$S(Sabc)cb = Sacb$

$a=b \Rightarrow LHS = Sccb = b = RHS$

$a\neq b \Rightarrow LHS = Sacb = RHS.$

R2. $S(Sabc)bc = Sab(Scbc)$ by theorem 2.6

$= Sabc$

$S(Sabc)ba = Sabc$

$a=b \Rightarrow LHS = Scba = c = RHS$

$a\neq b \Rightarrow LHS = Saba = a = RHS.$

R3. $S(Sabc)ca = a$

$a=b \Rightarrow LHS = Scca = a = RHS$

$a\neq b \Rightarrow LHS = Saca = a = RHS.$

$S(Sabc)ac = c$

$a=b \Rightarrow LHS = Scac = c = RHS$

$a\neq b \Rightarrow LHS = Saac = c = RHS.$

R4. $Sb(Sabc)c = b$

$a=b \Rightarrow LHS = Sbcc = b = RHS$

$a\neq b \Rightarrow LHS = Sbac = b = RHS.$

$Sb(Sabc)a = b$

$a=b \Rightarrow LHS = Sbca = b = RHS$

$a\neq b \Rightarrow LHS = Sbaa = b = RHS.$

Again, as $Sabc \neq Scba$ in general, we require another set of reduction rules for expressions of the forms $Sfd(Sabc)$ and $Sf(Sabc)d$. This is given in the following theorem.

Theorem 2.11

For $a,b,c \in \{v_1, \ldots, v_n\}$, the following statements are valid:

R5. $Sba(Sabc) = Sbac$

$Sbc(Sabc) = Sbca$

R6. $Scb(Sabc) = Scba$

$Sab(Sabc) = Sabc$

R7. $Sac(Sabc) = a$

$Sca(Sabc) = c$

R3. $Sa(Sabc)b = b$

Proof Using the same argument as in the proof of theorem 2.10.

We can also examine the reduction rules (a)-(g) which appear immediately after theorem 2.3 to see how far they may generalise to a space of n values.

These rules are fundamentally different from those previously derived as they imply that two values (here denoted by T and F) in the n-valued space are signalled as having preferential treatment. However, in n-valued logics, this is normally the case:

(a) For $a \in \{T,F\}$

$$SaF(STaF) = T$$

$$a \notin \{T,F\} \Rightarrow SaF(STaF) = a$$

(b) For $a \in \{T,F\}$

$$ST(STaF)F = a$$

$$a \notin \{T,F\} \Rightarrow ST(STaF)F = STTF = F$$

(c) For $a \in \{T,F\}$

$$SaT(STaF) = F$$

$$a \notin \{T,F\} \Rightarrow SaT(STaF) = a$$

(d) $a,b \in \{T,F\} \Rightarrow SaF(SaTb) = a$

$$a \notin \{T,F\} \Rightarrow SaF(SaTb) = SaFa = a$$

$$\therefore SaF(SaTb) = a \quad \forall\ a,b$$

(e) $a,b \in \{T,F\} \Rightarrow SaT(SaFb) = a$

$$a \notin \{T,F\} \Rightarrow SaT(SaFb) = SaTa = a$$

$$\therefore SaT(SaFb) = a \quad \forall\ a,b$$

(f) For $a,b \in \{T,F\}$

$$ST(SaTb)F = S(STaF)F(STbF)$$

$$a \notin \{T,F\} \Rightarrow \quad LHS = STaF = T$$

$$RHS = STF(STbF) = T$$

$$a \notin \{T,F\},\ b \notin \{T,F\}$$

$$\Rightarrow \quad LHS = \begin{cases} STbF = T & \text{if } a = T \\ STFF = T & \text{if } a = F \end{cases}$$

$$RHS = \begin{cases} SFF(STbF) = T & \text{if } a = T \\ STF(STbF) = T & \text{if } a = F \end{cases}$$

$$\therefore ST(SaTb)F = S(STaF)F(STbF) \quad \forall\ a,b$$

(g) For $a,b \in \{T,F\}$

$$ST(SaFb)F = S(STaF)T(STbF)$$

$$a \notin \{T,F\} \Rightarrow \quad LHS = STaF = T$$

$$RHS = STT(STbF) = STbF$$

$$= \begin{cases} F & \text{if } b = T \\ T & \text{if } b = F \\ T & \text{if } b \notin \{T,F\} \end{cases}$$

$a \in \{T,F\}$, $b \notin \{T,F\}$

$$\Rightarrow \quad LHS = \begin{cases} STTF = F & \text{if } a = T \\ STbF = T & \text{if } a = F \end{cases}$$

$$RHS = \begin{cases} SFTT = F & \text{if } a = T \\ STTT = T & \text{if } a = F \end{cases}$$

So (g) holds except when $b = T$, $a \notin \{T,F\}$.

But, as in a two-valued space $Sabc = Scba$, (g) can then be written as

$$SF(SaFb)T = S(SFaT)T(SFbT) \quad \text{for } a,b \in \{T,F\}.$$

We now show that the rule, written in this form, holds for all a,b.

$a \notin \{T,F\} \Rightarrow$ $LHS = SFaT = F$

$RHS = SFT(SFbT) = F$

$a \in \{T,F\}$, $b \notin \{T,F\}$

$$\Rightarrow \quad LHS = \begin{cases} SFTT = F & \text{if } a = T \\ SFbT = F & \text{if } a = F \end{cases}$$

$$RHS = \begin{cases} SFTF = F & \text{if } a = T \\ STTF = F & \text{if } a = F \end{cases}$$

$\therefore SF(SaFb)T = S(SFaT)T(SFbT) \quad \forall\ a,b$

So in a space of n values, the following laws hold

(d) $SaF(SaTb) = a$
(e) $SaT(SaFb) = a$
} Absorption laws

(f) $ST(SaTb)F = S(STaF)F(STbF)$
(g) $SF(SaFb)T = S(SFaT)T(SFbT)$
} De Morgan's laws

(a), (b), (c) however take a slightly different form.

(a) $$SaF(STaF) = \begin{cases} T & a \in \{T,F\} \\ a & a \notin \{T,F\} \end{cases}$$

(b) $$ST(STaF)F = \begin{cases} a & a \in \{T,F\} \\ F & a \notin \{T,F\} \end{cases}$$

(c) $$SaT(STaF) = \begin{cases} F & a \in \{T,F\} \\ a & a \notin \{T,F\} \end{cases}.$$

We illustrate these by giving S-expressions for Kleene's extension of the two-valued case to a three-valued space.

Kleene [20] defined the connectives $\sim$, $\wedge$, $\vee$, $\supset$ and $\equiv$ in a three-valued space $\{T,F,\omega\}$ as follows:

a	$\sim$ a
T	F
F	T
ω	ω

		b		
	a $\wedge$ b	T	F	ω
	T	T	F	ω
a	F	F	F	F
	ω	ω	F	ω

		b		
	a $\vee$ b	T	F	ω
	T	T	T	T
a	F	T	F	ω
	ω	T	ω	ω

	$a \supset b$	b: T	b: F	b: ω
	T	T	F	ω
a	F	T	T	T
	ω	T	ω	ω

	$a \equiv b$	b: T	b: F	b: ω
	T	T	F	ω
a	F	F	T	ω
	ω	ω	ω	ω

For $a,b \in \{T,F,\omega\}$, we may write

$\sim a$	as	S(SaFT)a(SaTF)
$a \wedge b$	as	S(S(SFaT)Fb)Ta
$a \vee b$	as	S(S(STaF)Tb)Fa
$a \supset b$	as	S(STab)(STbF)(SaFT)
$a \equiv b$	as	$(a \subset b) \wedge (b \supset a)$.

To justify our representation we show that each S-expression yields the same truth table as that of its corresponding logical connective.

(i) S(SaFT)a(SaTF) for $\sim a$

$a=T \Rightarrow$ S(SaFT)a(SaTF) = S(STFT)T(STTF)

= STTF

= F

$a=F \Rightarrow$ S(SaFT)a(SaTF) = S(SFFT)F(SFTF)

= STFF

$$= T$$

$$a=\omega \Rightarrow S(SaFT)a(SaTF) = S(S\omega FT)\omega(S\omega TF)$$

$$= S\omega\omega\omega$$

$$= \omega$$

(ii) $S(S(SFaT)Fb)Ta$ for $a \wedge b$

$$a=T \Rightarrow S(S(SFaT)Fb)Ta = S(S(SFTT)Fb)TT$$

$$= S(SFFb)TT$$

$$= SbTT$$

$$= b$$

$$a=F \Rightarrow S(S(SFaT)Fb)Ts = S(S(SFFT)Fb)TF$$

$$= S(STFb)TF$$

$$= STTF$$

$$= F$$

$$a=\omega \Rightarrow S(S(SFaT)Fb)Ta = S(S(SF\omega T)Fb)T\omega$$

$$= S(SFFb)T\omega$$

$$= SbT\omega$$

$$= \begin{cases} F & \text{if } b=F \\ \omega & \text{otherwise.} \end{cases}$$

(iii) $S(S(STaF)Tb)Fa$ = for $a \vee b$

$$a=T \Rightarrow S(S(STaF)Tb)Fa = S(S(STTF)Tb)FT$$

$$= S(SFTb)FT$$

$$= SFFT$$

$$= T$$

$$a=F \Rightarrow S(S(STaF)Tb)Fa = S(S(STFF)Tb)FF$$

$$= S(STTb)FF$$

$$= SbFF$$

$$= b$$

$$a=\omega \Rightarrow S(S(STaF)Tb)Fa = S(S(ST\omega F)Tb)F\omega$$
$$= S(STTb)F\omega$$
$$= SbF\omega$$
$$= \begin{cases} T & \text{if } b=T \\ \omega & \text{otherwise.} \end{cases}$$

(iv) $S(STab)(STbF)(SaFT)$ for $a \supset b$

$$a=T \Rightarrow S(STab)(STbF)(SaFT) = S(STTb)(STbF)(STFT)$$
$$= Sb(STbF)T$$
$$= b \qquad \text{by R4.}$$
$$a=F \Rightarrow S(STab)(STbF)(SaFT) = S(STFb)(STbF)(SFFT)$$
$$= ST(STbF)T$$
$$= T$$
$$a=\omega \Rightarrow S(STab)(STbF)(SaFT) = S(ST\omega b)(STbF)(S\omega FT)$$
$$= ST(STbF)\omega$$
$$= \begin{cases} T & \text{if } b=T \\ \omega & \text{otherwise.} \end{cases}$$

(v) $(a \supset b) \wedge (b \supset a)$ for $(a \equiv b)$

$a \equiv b$	a	b	$a \supset b$	$b \supset a$	$(a \supset b) \wedge (b \supset a)$
T	T	T	T	T	T
F	T	F	F	T	F
ω	T	ω	ω	T	ω
F	F	T	F	F	F
T	F	F	T	T	T
ω	F	ω	T	ω	ω
ω	ω	T	T	ω	ω
ω	ω	F	ω	T	ω
ω	ω	ω	ω	ω	ω

CHAPTER 3

THE OPERATOR S OVER FINITE NON-DISJOINT SPACES

§3.A Definition and Basic Properties of S

Consider an arbitrary, finite set U.

For any subsets a, b, c of the set U, Sabc is defined by

$$Sabc = \begin{cases} (a \cap \bar{b}) \cup c & \text{if } b \subseteq a \\ a & \text{if } b \not\subseteq a \end{cases}$$

In the case of n-valued disjoint space, this reduces to the definition of Chapter II.

Assumption

In many of the theorems in this chapter, one of the conditions of validity is that $b \subseteq a$ in the expression Sabc. Though this may appear to be restrictive, it will be seen later that the condition always holds in practice, therefore there is no loss of generality.

The basic properties of S

In Chapter II, we found that the operator S, acting upon a finite space of disjoint elements, has certain basic properties which serve as a valuable means of simplifying S-expressions. Only some of these properties hold unconditionally, the others are subject to restrictions.

In the following theorems, we investigate the validity of such properties, when a,b,c,d,f are subsets of U.

Theorem 3.1

Saba = a

Saab = b

Sabb = a for all a,b

Proof

(i) $Saba = a$

$b \subseteq a \Rightarrow LHS = a = RHS$

$b \not\subseteq a \Rightarrow LHS = (a \cap \overline{b}) \cup a$

$= a \qquad = RHS$

(ii) $Saab = b$

$LHS = (a \cap \overline{a}) \cup b$

$= b \qquad = RHS$

(iii) $Sabb = a$

$b \not\subseteq a \Rightarrow LHS = a = RHS$

$b \subseteq a \Rightarrow LHS = (a \cap \overline{b}) \cup b$

$= (a \cup b) \cap (\overline{b} \cup b)$

$= a \cap U \qquad \because b \subseteq a$

$= a = RHS$

Theorem 3.2

Given $b \subseteq a$, then

$Sabc = Scba$

iff either $b \subseteq c$ or $a \subseteq b \cup c$

Proof

$LHS = Sabc$

$= (a \cap \overline{b}) \cup c$

$$RHS = \begin{cases} (c \cap \overline{b}) \cup a & \text{if } b \subseteq c \\ c & \text{if } b \not\subseteq c \end{cases}$$

$b \subseteq c$: $LHS = (a \cup c) \cap (\overline{b} \cup c)$

$= a \cup c \qquad \because b \subseteq c \Rightarrow \overline{b} \cup c = U$

$RHS = (c \cap \overline{b}) \cup a$

$= (c \cup a) \cap (\overline{b} \cup a)$

$= c \cup a \qquad \because b \subseteq a \Rightarrow \overline{b} \cup a = U$

$\therefore \quad LHS = RHS$

$b \not\subseteq c$: LHS $= (a \cap \bar{b}) \cup c$

RHS $= c \subseteq$ LHS

$\therefore$ LHS = RHS iff LHS $\subseteq$ RHS

ie. iff $(a \cap \bar{b}) \cup c \subseteq c$

iff $(a \cap \bar{b}) \subseteq c$

iff $a \subseteq b \cup c$

Theorem 3.3

Given that $b \subseteq a$ and $b \subseteq Sabc$, then

$$S(Sabc)bd = Sab(Scbd)$$

Proof $b \subseteq a$ and $b \subseteq Sabc$

$\Rightarrow$ $b \subseteq (a \cap \bar{b}) \cup c$

ie. $b \cap (\bar{a} \cup b) \subseteq c$

$(b \cap \bar{a}) \cup (b \cap b) \subseteq c$

$b \subseteq c$

$\therefore$ $b \subseteq a$, $b \subseteq c$ implies

$LHS = [\{(a \cap \bar{b}) \cup c\} \cap \bar{b}] \cup d$

$= [\{(a \cup c) \cap (\bar{b} \cup c)\} \cap \bar{b}] \cup d$

$= \{(a \cup c) \cap \bar{b}\} \cup d$

and

$RHS = (a \cap \bar{b}) \cup (c \cap \bar{b}) \cup d$

$= \{(a \cup c) \cap \bar{b}\} \cup d$

$\therefore$ LHS = RHS.

Remark

In the above theorem, the conditions $b \subseteq a$ and $b \subseteq Sabc$ are given in accordance with our assumption. But in fact, the statement is valid even when such conditions are removed. This can be proved as follows:

Theorem 3.3*

$S(Sabc)bd = Sab(Scbd)$ for all a,b,c,d.

Proof

$b \not\subseteq a$:

$$\begin{aligned} LHS &= S(Sabc)bd \\ &= Sabd \\ &= a. \end{aligned}$$

$$\begin{aligned} RHS &= Sab(Scbd) \\ &= a \end{aligned}$$

$b \subseteq a$, $b \not\subseteq c$:

$$\begin{aligned} LHS &= S\{(a \cap \bar{b}) \cup c\}bd \\ &= (a \cap \bar{b}) \cup c \qquad \because b \not\subseteq (a \cap \bar{b}) \cup c \end{aligned}$$

$$\begin{aligned} RHS &= Sabc \qquad \because b \not\subseteq c \\ &= (a \cap \bar{b}) \cup c \end{aligned}$$

$b \subseteq a$, $b \subseteq c \Rightarrow$ LHS = RHS as proved in theorem 3.3

Hence LHS = RHS for all a,b,c,d.

To help clarify the proof of the next theorem, we first prove the following lemmas:

Lemma 3.4.1

If $b \subseteq a$ and $Sabc \subseteq d$, then

$a \subseteq d$ iff $b \subseteq d$.

Proof

$a \subseteq d$:

$b \subseteq a$ and $a \subseteq d$ trivially implies $b \subseteq d$.

$b \subseteq d$:

Now $b \subseteq a$ and $Sabc \subseteq d$

$\Rightarrow \quad (a \cap \bar{b}) \cup c \subseteq d$

ie. $\{(a \cap \bar{b}) \cup c\} \cap \bar{d} = \emptyset$

$(a \cap \bar{b} \cap \bar{d}) \cup (c \cap \bar{d}) = \emptyset$

$\therefore \quad a \cap \bar{b} \cap \bar{d} = \emptyset$ ie. $a \subseteq b \cup d$

and $c \cap \bar{d} = \emptyset$ ie. $c \subseteq d$.

Hence $b \subseteq a$ and $Sabc \subseteq d \Rightarrow a \subseteq b \cup d$ and $c \subseteq d$

But $b \subseteq d \Rightarrow \bar{b} \supseteq \bar{d} \Rightarrow \bar{b} \cap \bar{d} = \bar{d}$

So $a \cap \bar{b} \cap \bar{d} = \emptyset$

$\Rightarrow \quad a \cap \bar{d} = \emptyset$

ie. $a \subseteq d$.

Lemma 3.4.2

Given $a \subseteq d$, then $(d \cap \bar{b}) \cup f \subseteq (d \cap \bar{a}) \cup f$

iff $a \subseteq b \cup f$

Proof

$(d \cap \bar{b}) \cup f \subseteq (d \cap \bar{a}) \cup f$

iff $(d \cap \bar{b}) \subseteq (d \cap \bar{a}) \cup f$

ie. $\overline{(d \cap \bar{b}) \cap (d \cap \bar{a})} \subseteq f$

$(d \cap \bar{b}) \cap (\bar{d} \cup a) \subseteq f$

$(d \cap \bar{b} \cap \bar{d}) \cup (d \cap \bar{b} \cap a) \subseteq f$

$a \cap \bar{b} \subseteq f \qquad \therefore \ a \subseteq d$

ie. $a \subseteq b \cup f$.

Theorem 3.4

Given $b \subseteq a$ and $Sabc \subseteq d$, then

$$Sd(Sabc)f = S(Sdaf)(Sdbf)(Sdcf)$$

iff either (i) $a \subseteq d$, $a \subseteq b \cup f$

or (ii) $a \subseteq d$, $b \subseteq c \cup f$, $c \subseteq a \cup f$

or (iii) $a \not\subseteq d$, $a \subseteq b \cup c \cup f$.

Proof

From proof of lemma 3.4.1

$$b \subseteq a \text{ and } Sabc \subseteq d \Rightarrow b \subseteq a \subseteq b \cup d \text{ and } c \subseteq d \qquad (1)$$

$\therefore$ There are two possibilities to be considered:

(a) $a \not\subseteq d$

(b) $a \subseteq d$

(a) $a \not\subseteq d$:

Under the given conditions (1), $a \not\subseteq d \Rightarrow b \not\subseteq d$ by lemma 3.4.1

So
$$\begin{aligned} LHS &= Sd\{(a \cap \bar{b}) \cup c\}f && \because b \subseteq a \\ &= [d \cap \overline{\{(a \cap \bar{b}) \cup c\}}] \cup f && \because Sabc \subseteq d \\ &= \{d \cap (\bar{a} \cup b) \cap \bar{c}\} \cup f \\ &= \{(d \cap \bar{c}) \cap (\bar{a} \cup b)\} \cup f \\ RHS &= Sdd\{(d \cap \bar{c}) \cup f\} && \because a, b \not\subseteq d,\ c \subseteq d \\ &= (d \cap \bar{c}) \cup f \end{aligned}$$

$\therefore$ $LHS \subseteq RHS$

So $LHS = RHS$ iff $RHS \subseteq LHS$

ie. $(d \cap \bar{c}) \cup f \subseteq \{(d \cap \bar{c}) \cap (\bar{a} \cup b)\} \cup f$

$(d \cap \bar{c}) \subseteq \{(d \cap \bar{c}) \cap (\bar{a} \cup b)\} \cup f$

$\overline{(d \cap \bar{c}) \cap \{(d \cap \bar{c}) \cap (\bar{a} \cup b)\}} \subseteq f$

$(d \cap \bar{c}) \cap (a \cap \bar{b}) \subseteq f \qquad (2)$

But $a \subseteq b \cup d \Rightarrow a \cap \bar{b} \subseteq d \Rightarrow (a \cap \bar{b}) \cap d = a \cap \bar{b}$

$\therefore$ (2) $\Rightarrow$ $a \cap \bar{b} \cap \bar{c} \subseteq f$

ie. $a \subseteq b \cup c \cup f$

Thus for $a \not\subseteq d$, LHS = RHS iff $a \subseteq b \cup c \cup f$.

(b) $a \subseteq d$:

Under the given conditions, $a \subseteq d \Rightarrow b \subseteq d$ by lemma 3.4.1

Again, $\overline{\text{LHS}} = d \cap \{(a \cap \bar{b}) \cup c\} \cup f$

$= \{d \cap (\bar{a} \cup b) \cap \bar{c}\} \cup f$ (3)

$= [\{(d \cap \bar{a}) \cup (d \cap b)\} \cap \bar{c}] \cup f$

$= [\{(d \cap \bar{a}) \cup b\} \cap \bar{c}] \cup f$ $\quad \therefore$ $b \subseteq d$ (4)

$\overline{\text{RHS}} = S\{(d \cap \bar{a}) \cup f\}\{(d \cap \bar{b}) \cup f\}\{(d \cap \bar{c}) \cup f\}$

For $a \subseteq d$, $(d \cap \bar{b}) \cup f \subseteq (d \cap \bar{a}) \cup f$ iff $a \subseteq b \cup f$ by lemma 3.4.2

Hence, we consider the two cases:

(b1) $a \not\subseteq b \cup f$

(b2) $a \subseteq b \cup f$.

(b1) $a \not\subseteq b \cup f$:

then $\overline{\text{RHS}} = (d \cap \bar{a}) \cup f$

and LHS = RHS iff

$[\{(d \cap \bar{a}) \cup b\} \cap \bar{c}] \cup f = (d \cap \bar{a}) \cup f$ from (4)

ie. iff $[\{(d \cap \bar{a}) \cup b\} \cap \bar{c}] \cup f \subseteq (d \cap \bar{a}) \cup f$ (5)

and $(d \cap \bar{a}) \cup f \subseteq [\{(d \cap \bar{a}) \cup b\} \cap \bar{c}] \cup f$ (6)

(5) is equivalent to

$\{(d \cap \bar{a}) \cup b\} \cap \bar{c} \subseteq (d \cap \bar{a}) \cup f$

ie. $\{(d \cap \bar{a}) \cup b\} \cap \overline{(d \cap \bar{a})} \subseteq c \cup f$

$b \cap (\bar{d} \cup a) \subseteq c \cup f$

$(b \cap \bar{d}) \cup (b \cap a) \subseteq c \cup f$

$b \subseteq c \cup f$ $\quad \therefore$ $b \subseteq d$, $b \subseteq a$

(6) is equivalent to

$$d \cap \bar{a} \subseteq [\{(d \cap \bar{a}) \cup b\} \cap \bar{c}] \cup f$$

$$\text{ie. } \overline{(d \cap \bar{a}) \cap [\{(d \cap \bar{a}) \cup b\} \cap \bar{c}]} \subseteq f$$

$$\overline{(d \cap \bar{a}) \cap [\{(d \cap \bar{a}) \cap \bar{b}\} \cup c]} \subseteq f$$

$$d \cap \bar{a} \cap c \subseteq f$$

$$\bar{a} \cap c \subseteq f \qquad \because\ c \subseteq d$$

$$c \subseteq a \cup f.$$

Hence when $a \subseteq d$, $a \not\subseteq b \cup f$, LHS = RHS iff $b \subseteq c \cup f$ and $c \subseteq a \cup f$

(b2) $a \subseteq b \cup f$:

Then $$\text{RHS} = [\{(d \cap \bar{a}) \cup f\} \cap \overline{\{(d \cap \bar{b}) \cup f\}}] \cup \{(d \cap \bar{c}) \cup f\}$$

$$= [\{(d \cap \bar{a}) \cup f\} \cap (\bar{d} \cup b) \cap \bar{f}] \cup (d \cap \bar{c}) \cup f$$

$$= \{(d \cap \bar{a}) \cap (\bar{d} \cup b) \cap \bar{f}\} \cup f \cup (d \cap \bar{c})$$

$$= \{\bar{a} \cap d \cap (\bar{d} \cup b) \cup f \cup (d \cap \bar{c})$$

$$= (\bar{a} \cap d \cap b) \cup f \cup (d \cap \bar{c})$$

$$= f \cup (d \cap \bar{c}) \qquad \because\ b \subseteq a \Rightarrow b \cap \bar{a} = \emptyset$$

From (3), $$\text{LHS} = \{d \cap (\bar{a} \cup b) \cap \bar{c}) \cup f$$

$$= \{(d \cap \bar{c}) \cup f\} \cap \{(\bar{a} \cup b) \cup f\}$$

But $a \subseteq b \cup f$, so $\bar{a} \cup b \cup f = U$

Hence $$\text{LHS} = (d \cap \bar{c}) \cup f$$

$$= \text{RHS}.$$

Theorem 3.5

Given $b \subseteq a$ and $d \subseteq f$, then

$$Sfd(Sabc) = S(Sfda)(Sfdb)(Sfdc)$$

Proof

$$\begin{aligned}
\text{LHS} &= Sfd[(a \cap \bar{b}) \cup c] && \because\ b \subseteq a \\
&= (f \cap \bar{d}) \cup (a \cap \bar{b}) \cup c && \because\ d \subseteq f \\
\text{RHS} &= S\{(f \cap \bar{d}) \cup a\}\{(f \cap \bar{d}) \cup b\}\{f \cap \bar{d}) \cup c\} && \because\ d \subseteq f \\
&= [\{(f \cap \bar{d}) \cup a\} \cap \overline{\{(f \cap \bar{d}) \cup b\}}] \cup \{(f \cap \bar{d}) \cup c] && \because\ b \subseteq a \\
&= [\{(f \cap \bar{d}) \cup a\} \cap \overline{(f \cap \bar{d})} \cap \bar{b}] \cup (f \cap \bar{d}) \cup c \\
&= [(a \cap \bar{b}) \cap \overline{(f \cap \bar{d})}] \cup (f \cap \bar{d}) \cup c \\
&= (a \cap \bar{b}) \cup (f \cap \bar{d}) \cup c \\
&= \text{LHS}.
\end{aligned}$$

Remark

In the foregoing theorem, it is clear that $d \not\subseteq f$ is a sufficient condition for LHS = RHS.

Because $d \not\subseteq f \Rightarrow \text{RHS} = Sfff = f = \text{LHS}$.

In order to find the conditions of validity for P5, we first prove the following lemmas:

Lemma 3.6.1

Given $b \subseteq a$, then $b \subseteq (a \cap \bar{d}) \cup f$

iff $b \cap d \subseteq f$

Proof

$$\begin{aligned}
& b \subseteq (a \cap \bar{d}) \cup f \\
\text{iff}\quad & b \cap \overline{(a \cap \bar{d})} \subseteq f \\
& b \cap (\bar{a} \cup d) \subseteq f \\
& (b \cap \bar{a}) \cup (b \cap d) \subseteq f \\
& b \cap d \subseteq f && \because\ b \subseteq a \Rightarrow b \cap \bar{a} = \emptyset
\end{aligned}$$

Lemma 3.6.2

Given $b \subseteq a$, $d \subseteq a,b,c$ then

$S(Sabc)df = S(Sadf)(Sbdf)(Scdf)$

Proof

$$LHS = S\{(a \cap \bar{b}) \cup c\}df$$

$$= [\{(a \cap \bar{b}) \cup c\} \cap \bar{d}] \cup f \qquad \because\ d \subseteq c$$

$$= (a \cap \bar{b} \cap \bar{d}) \cup (c \cap \bar{d}) \cup f$$

$$RHS = S\{(a \cap \bar{d}) \cup f\}\{(b \cap \bar{d}) \cup f\}\{c \cap \bar{d}) \cup f\}$$

$$= [\{(a \cap \bar{d}) \cup f\} \cap \overline{\{(b \cap \bar{d}) \cup f\}}] \cup (c \cap \bar{d}) \cup f \qquad \because\ b \subseteq a$$

$$= [\{(a \cap \bar{d}) \cup f\} \cap (\bar{b} \cup d) \cap \bar{f}] \cup (c \cap \bar{d}) \cup f$$

$$= \{(a \cap \bar{d}) \cap (\bar{b} \cup d) \cap \bar{f}\} \cup f \cup (c \cap \bar{d})$$

$$= (a \cap \bar{d} \cap \bar{b} \cap \bar{f}) \cup f \cup (c \cap \bar{d})$$

$$= (a \cap \bar{d} \cap \bar{b}) \cup f \cup (c \cap \bar{d})$$

$$= LHS$$

Lemma 3.6.3

Given $b \subseteq a$, $d \subseteq a$, $d \subseteq c$, $d \nsubseteq b$ then

$S(Sabc)df = S(Sadf)(Sbdf)(Scdf)$

iff either (i) $b \cap d \subseteq f$

or (ii) $b \subseteq c \cup f$ and $c \subseteq a \cup f$.

Proof $b \subseteq a$ and $d \subseteq c$

$$\Rightarrow LHS = [\{(a \cap \bar{b}) \cup c\} \cap \bar{d}] \cup f$$

and $RHS = S\{(a \cap \bar{d}) \cup f\}b\{(c \cap \bar{d}) \cup f\}$ $\because\ d \subseteq a$

From lemma 3.6.1, $b \subseteq (a \cap \bar{d}) \cup f$ iff $b \cap d \subseteq f$.

So we consider the two possibilities:

(a) $b \cap d \nsubseteq f$

and (b) $b \cap d \subseteq f$.

(a) $b \cap d \not\subseteq f$:

$\Rightarrow$ RHS $= (a \cap \bar{d}) \cup f$.

$\therefore$ LHS = RHS iff

$[\{(a \cap \bar{b}) \cup c\} \cap \bar{d}] \cup f = (a \cap \bar{d}) \cup f$ i.e.

iff (a1) $[\{(a \cap \bar{b}) \cup c\} \cap \bar{d}] \cup f \subseteq (a \cap \bar{d}) \cup f$

and (a2) $(a \cap \bar{d}) \cup f \subseteq [\{(a \cap \bar{b}) \cup c\} \cap \bar{d}] \cup f$

(a1) is equivalent to

$$\{(a \cap \bar{b}) \cup c\} \cap \bar{d} \subseteq (a \cap \bar{d}) \cup f.$$

i.e. $$[\{(a \cap \bar{d}) \cap \bar{b}\} \cup (c \cap \bar{d})] \cap \overline{(a \cap \bar{d})} \subseteq f$$

$$c \cap \bar{d} \cap (\bar{a} \cup d) \subseteq f$$

$$c \cap \bar{d} \cap \bar{a} \subseteq f$$

$$c \cap \bar{a} \subseteq f \quad \because d \subseteq a \Rightarrow \bar{d} \cap \bar{a} = \bar{a}$$

i.e. $$c \subseteq a \cup f.$$

(a2) is equivalent to

$$(a \cap \bar{d}) \subseteq [\{(a \cap \bar{b}) \cup c\} \cap \bar{d}] \cup f$$

i.e. $$(a \cap \bar{d}) \cap [\{\overline{(a \cap \bar{d})} \cup b\} \cap (\bar{c} \cup d) \subseteq f$$

$$a \cap \bar{d} \cap b \cap (\bar{c} \cup d) \subseteq f$$

$$a \cap b \cap \bar{d} \cap \bar{c} \subseteq f$$

$$b \cap \bar{c} \subseteq f \quad \because b \subseteq a \text{ and } d \subseteq c$$

i.e. $$b \subseteq c \cup f$$

$\therefore$ When $b \cap d \not\subseteq f$, LHS = RHS iff $b \subseteq c \cup f$ and $c \subseteq a \cup f$. (ii)

(b) $b \cap d \subseteq f$:

$\Rightarrow$ RHS $= [\{(a \cap \bar{d}) \cup f\} \cap \bar{b}] \cup \{(c \cap \bar{d}) \cup f\}$

$= (a \cap \bar{d} \cap \bar{b}) \cup (\bar{b} \cap f) \cup f \cup (c \cap \bar{d})$

$= (a \cap \bar{d} \cap \bar{b}) \cup f \cup (c \cap \bar{d})$

$= [\{(a \cap \bar{b}) \cup c\} \cap \bar{d}] \cup f$

$=$ LHS

Hence $b \cap d \subseteq f \Rightarrow$ LHS = RHS. (i)

Lemma 3.6.4

Given $b \subseteq a$, $d \subseteq a$, $d \not\subseteq b,c$ then

$S(Sabc)df = S(Sadf)(Sbdf)(Scdf)$

iff one of the following sets of conditions hold

(i) $b \cap d \not\subseteq c$, $b \cap d \not\subseteq f$, $b \subseteq d \cup f$, $c \subseteq a \cup f$, $d \subseteq b \cup f$, $f \subseteq a \cup c$, $b \cap f \subseteq c$ and $c \cap d \subseteq f$

(ii) $b \cap d \not\subseteq c$, $b \cap d \subseteq f$, $d \subseteq b \cup c \cup f$, $f \subseteq a \cup c$

(iii) $b \cap d \subseteq c$, $b \cap d \not\subseteq f$, $c \subseteq a \cup f$, $b \subseteq c \cup d \cup f$

(iv) $b \cap d \subseteq c$, $b \cap d \subseteq f$, $b \cap f \subseteq c$, $c \cap d \subseteq f$.

Proof

$$LHS = S\{(a \cap \bar{b}) \cup c\}df \qquad \because b \subseteq a$$

Now $d \subseteq a \Rightarrow d \subseteq (a \cap \bar{b}) \cup c$ iff $b \cap d \subseteq c$ by lemma 3.6.1

Hence

$$LHS = \begin{cases} (a \cap \bar{b}) \cup c & \text{if } b \cap d \not\subseteq c \\ [\{(a \cap \bar{b}) \cup c\} \cap \bar{d}] \cup f & \text{if } b \cap d \subseteq c \end{cases}$$

$$RHS = S\{(a \cap \bar{d}) \cup f\}bc \qquad \because d \subseteq a,\ d \not\subseteq b,c$$

By lemma 3.6.1, $b \subseteq a \Rightarrow b \subseteq (a \cap \bar{d}) \cup f$ iff $b \cap d \subseteq f$.

Hence

$$RHS = \begin{cases} (a \cap \bar{d}) \cup f & \text{if } b \cap d \not\subseteq f \\ [\{(a \cap \bar{d}) \cup f\} \cap \bar{b}] \cup c & \text{if } b \cap d \subseteq f. \end{cases}$$

So we consider the following possibilities:-

(a) $b \cap d \not\subseteq c$, $b \cap d \not\subseteq f$

(b) $b \cap d \not\subseteq c$, $b \cap d \subseteq f$

(c) $b \cap d \subseteq c$, $b \cap d \not\subseteq f$

(d) $b \cap d \subseteq c$, $b \cap d \subseteq f$.

(a) $b \cap d \not\subseteq c$, $b \cap d \not\supseteq f$

LHS = RHS

iff $(a \cap \bar{b}) \cup c = (a \cap \bar{d}) \cup f$

iff (a1) $\{(a \cap \bar{b}) \cup c\} \cap \overline{\{(a \cap \bar{d}) \cup f\}} = \emptyset$

and (a2) $\overline{\{(a \cap \bar{b}) \cup c\}} \cap \{(a \cap \bar{d}) \cup f\} = \emptyset$

(a1) is equivalent to

$$\{(a \cap \bar{b}) \cup c\} \cap (\bar{a} \cup d) \cap \bar{f} = \emptyset$$

$$\{(a \cap \bar{b}) \cap (\bar{a} \cup d) \cap \bar{f}\} \cup \{c \cap (\bar{a} \cup d) \cap \bar{f}\} = \emptyset$$

$$(a \cap \bar{b} \cap d \cap \bar{f}) \cup (c \cap \bar{a} \cap \bar{f}) \cup (c \cap d \cap \bar{f}) = \emptyset$$

ie. $a \cap d \cap (\bar{b} \cap \bar{f}) = \emptyset \Rightarrow d \subseteq b \cup f$

and $c \cap \bar{a} \cap \bar{f} = \emptyset \Rightarrow c \subseteq a \cup f$

and $c \cap d \cap \bar{f} = \emptyset \Rightarrow c \cap d \subseteq f$

Similarly, (a2) is equivalent to

$b \subseteq d \cup f$, $f \subseteq a \cup c$ and $f \cap b \subseteq c$.

Hence when $b \cap d \not\subseteq c$, $b \cap d \not\supseteq f$, LHS = RHS iff

$b \subseteq d \cup f$, $c \subseteq a \cup f$, $d \subseteq b \cup f$, $f \subseteq a \cup c$, $b \cap f \subseteq c$ and $c \cap d \subseteq f$.

(b) $b \cap d \not\subseteq c$, $b \cap d \not\supseteq f$

LHS = RHS

iff $(a \cap \bar{b}) \cup c = [\{(a \cap \bar{d}) \cup f\} \cap \bar{b}] \cup c$

iff (b1) $(a \cap \bar{b}) \cup c \subseteq [\{(a \cap \bar{d}) \cup f\} \cap \bar{b}] \cup c$

and (b2) $[\{(a \cap \bar{d}) \cup f\} \cap \bar{b}] \cup c \subseteq (a \cap \bar{b}) \cup c$

(b1) is equivalent to

$$(a \cap \bar{b}) \subseteq [\{(a \cap \bar{d}) \cup f\} \cap \bar{b}] \cup c$$

ie. $(a \cap \bar{b}) \cap [\{\overline{(a \cap \bar{d})} \cap \bar{f}\} \cup b] \subseteq c$

$$(a \cap \bar{b}) \cap [\{(\bar{a} \cup d) \cap \bar{f}\} \cup b] \subseteq c$$

$$(a \cap \bar{b}) \cap (\bar{a} \cup d) \cap \bar{f} \subseteq c$$

$$a \cap \bar{b} \cap d \cap \bar{f} \subseteq c$$

$$d \cap \bar{b} \cap \bar{f} \subseteq c \qquad \because\ d \subseteq a$$

$$d \subseteq b \cup c \cup f$$

(b2) is equivalent to

$$\{(a \cap \bar{d}) \cup f\} \cap \bar{b} \subseteq (a \cap \bar{b}) \cup c$$

$$[\{(a \cap \bar{d}) \cap \bar{b}\} \cup (f \cap \bar{b})] \cap \overline{(a \cap \bar{b})} \subseteq c$$

$$(f \cap \bar{b}) \cap (\bar{a} \cup b) \subseteq c$$

$$f \cap \bar{b} \cap \bar{a} \subseteq c$$

$$f \cap \bar{a} \subseteq c \qquad \because\ b \subseteq a$$

$$f \subseteq a \cup c \ .$$

$\therefore$ When $b \cap d \not\subseteq c$, $b \cap d \subseteq f$, LHS = RHS iff $d \subseteq b \cup c \cup f$ and $f \subseteq a \cup c$.

(c) $b \cap d \subseteq c$, $b \cap d \not\subseteq f$

LHS = RHS

iff $[\{(a \cap \bar{b}) \cup c\} \cap \bar{d}] \cup f = (a \cap \bar{d}) \cup f$

iff (c1) $[\{(a \cap \bar{b}) \cup c\} \cap \bar{d}] \cup f \subseteq (a \cap \bar{d}) \cup f$

and (c2) $(a \cap \bar{d}) \cup f \subseteq [\{(a \cap \bar{b}) \cup c\} \cap \bar{d}] \cup f$

From the proof of lemma 3.6.3, (c1) holds iff $c \subseteq a \cup f$,

and (c2) is equivalent to

$$a \cap b \cap \bar{d} \cap \bar{c} \subseteq f$$

$$b \cap \bar{d} \cap \bar{c} \subseteq f \qquad \because\ b \subseteq a$$

$$b \subseteq c \cup d \cup f \ .$$

$\therefore$ When $b \cap d \subseteq c$, $b \cap d \not\subseteq f$, LHS = RHS iff $c \subseteq a \cup f$ and $b \subseteq c \cup d \cup f$.

(d) $b \cap d \subseteq c$, $b \; d \subseteq f$

$\Rightarrow$ RHS $= [\{(a \cap \bar{d}) \cup f\} \cap \bar{b}] \cup c$

$= (a \cap \bar{d} \cap \bar{b}) \cup (f \cap \bar{b}) \cup c$

and LHS $= (a \cap \bar{b} \cap \bar{d}) \cup (c \cap \bar{d}) \cup f$

$\therefore$ LHS = RHS iff

(d1) $(a \cap \bar{b} \cap \bar{d}) \cup (c \cap \bar{d}) \cup f \subseteq (a \cap \bar{b} \cap \bar{d}) \cup (f \cap \bar{b}) \cup c$

and (d2) $(a \cap \bar{b} \cap \bar{d}) \cup c \cup (f \cap \bar{b}) \subseteq (a \cap \bar{b} \cap \bar{d}) \cup (c \cap \bar{d}) \cup f$.

(d1) is equivalent to

$$f \subseteq (a \cap \bar{b} \cap \bar{d}) \cup (f \cap \bar{b}) \cup c$$

i.e. $f \cap (\bar{f} \cup b) \cap (\bar{a} \cup b \cup d) \subseteq c$

$(f \cap b) \cap (\bar{a} \cup b \cup d) \subseteq c$

$$f \cap b \subseteq c$$

and (d2) is equivalent to

$$c \subseteq (a \cap \bar{b} \cap \bar{d}) \cup (c \cap \bar{d}) \cup f$$

i.e. $c \cap (\bar{c} \cup d) \cap (\bar{a} \cup b \cup d) \subseteq f$

$c \cap d \cap (\bar{a} \cup b \cup d) \subseteq f$

$$c \cap d \subseteq f$$

$\therefore$ When $b \cap d \subseteq c$, $b \cap d \subseteq f$, LHS = RHS iff $b \cap f \subseteq c$ and $c \cap d \subseteq f$.

Lemma 3.6.5

Given $b \subseteq a$, $d \subseteq c$, $d \nsubseteq a,b$ then

$S(Sabc)df = S(Sadf)\,(Sbdf)\,(Scdf)$

iff $a \cap d \subseteq b \cup f$

Proof

LHS $= S\{(a \cap \bar{b}) \cup c\}df$ $\quad \because b \subseteq a$

$= [\{(a \cap \bar{b}) \cup c\} \cap \bar{d}] \cup f$ $\quad \because d \subseteq c$

$= (a \cap \bar{b} \cap \bar{d}) \cup (c \cap \bar{d}) \cup f$

$$\text{RHS} = Sab\{(c \cap \bar{d}) \cup f\} \qquad \because\ d \subseteq c$$

$$= (a \cap \bar{b}) \cup (c \cap \bar{d}) \cup f \qquad \because\ b \subseteq a$$

$\therefore$ LHS $\subseteq$ RHS and so LHS = RHS iff

$$\text{RHS} \subseteq \text{LHS}$$

$$\text{i.e. } (a \cap \bar{b}) \cup (c \cap \bar{d}) \cup f \subseteq (a \cap \bar{b} \cap \bar{d}) \cup (c \cap \bar{d}) \cup f$$

$$a \cap \bar{b} \subseteq (a \cap \bar{b} \cap \bar{d}) \cup (c \cap \bar{d}) \cup f$$

$$\text{i.e. } (a \cap \bar{b}) \cap \{\overline{(a \cap \bar{b})} \cup d\} \cap (\bar{c} \cup d) \subseteq f$$

$$a \cap \bar{b} \cap d \cap (\bar{c} \cup d) \subseteq f$$

$$\text{i.e. } a \cap \bar{b} \cap d \subseteq f$$

$$a \cap d \subseteq b \cup f$$

$\therefore$ LHS = RHS iff $a \cap d \subseteq b \cup f$.

Lemma 3.6.6

Given $b \subseteq a$, then $Sabc = a$ iff

$$b \subseteq c \subseteq a$$

Proof

$$b \subseteq a \Rightarrow Sabc = (a \cap \bar{b}) \cup c = a$$

iff $a = (a \cap \bar{b}) \cup c$

$$\text{i.e. } [a \cap \{\overline{(a \cap \bar{b}) \cup c}\}] \cup [\{(a \cap \bar{b}) \cup c\} \cap \bar{a}] = \emptyset$$

$$\{a \cap (\bar{a} \cup b) \cap \bar{c}\} \cup \{(a \cap \bar{b} \cap \bar{a}) \cup (c \cap \bar{a})\} = \emptyset$$

$$(a \cap b \cap \bar{c}) \cup (c \cap \bar{a}) = \emptyset$$

$$(b \cap \bar{c}) \cup (c \cap \bar{a}) = \emptyset \qquad \because\ b \subseteq a$$

i.e. $b \subseteq c \subseteq a$

Lemma 3.6.7

Given $b \subseteq a$, $d \notin a,b,c$, $d \subseteq Sabc$ then

$$S(Sabc)df = S(Sadf)(Sbdf)(Scdf)$$

iff $d \subseteq f \subseteq a \cup c$ and $f \cap b \subseteq c$

Proof $d \notin a, b, c \Rightarrow RHS = Sabc$.

$d \subseteq Sabc$, so LHS = RHS iff $S(Sabc)df = Sabc$

ie. iff $d \subseteq f \subseteq Sabc$ by lemma 3.6.6

$f \subseteq Sabc$ and $b \subseteq a \Rightarrow f \subseteq (a \cap \bar{b}) \cup c$

ie. $f \cap (\bar{a} \cup b) \cap \bar{c} = \emptyset$

$(f \cap \bar{a} \cap \bar{c}) \cup (f \cap b \cap \bar{c}) = \emptyset$

$\therefore$ $f \subseteq a \cup c$ and $f \cap b \subseteq c$.

Theorem 3.6

Given $b \subseteq a$ and $d \subseteq Sabc$, then

$S(Sabc)df = S(Sadf)(Sbdf)(Scdf)$

iff one of the following sets of conditions is satisfied:

(i) $d \subseteq b$

(ii) $d \subseteq a$, $d \subseteq c$ and either $b \cap d \subseteq f$ or $b \subseteq c \cup f$, $c \subseteq a \cup f$

(iii) $d \subseteq a$, $d \notin c$ and

(a) $b \cap d \notin c$, $b \cap d \notin f$, $b \subseteq d \cup f$, $c \subseteq a \cup f$, $d \subseteq b \cup f$, $f \subseteq a \cup c$, $b \cap f \subseteq c$ and $c \cap d \subseteq f$

or (b) $b \cap d \notin c$, $b \cap d \subseteq f$, $d \subseteq b \cup c \cup f$, $f \subseteq a \cup c$

or (c) $b \cap d \subseteq c$, $b \cap d \notin f$, $c \subseteq a \cup f$, $b \subseteq c \cup d \cup f$

or (d) $b \cap d \subseteq c$, $b \cap d \subseteq f$, $b \cap f \subseteq c$, $c \cap d \subseteq f$

(iv) $d \notin a$, $d \subseteq c$, $a \cap d \subseteq b \cup f$

(v) $d \notin a$, $d \notin c$, $d \subseteq f \subseteq a \cup c$, $b \cap f \subseteq c$.

Proof

$b \subseteq a$ and $d \subseteq Sabc$

$\Rightarrow$ $d \subseteq (a \cap \bar{b}) \cup c$

ie. $d \cap (\bar{a} \cup b) \cap \bar{c} = \emptyset$

$(d \cap \bar{a} \cap \bar{c}) \cup (d \cap b \cap \bar{c}) = \emptyset$

ie. $d \subseteq a \cup c$ and $d \cap b \subseteq c$.

If $d \subseteq b$, then $b \subseteq a \Rightarrow d \subseteq a$

and $d \cap b \subseteq c \Rightarrow d \subseteq c$

Hence, given $b \subseteq a$, $d \subseteq a \cup c$ and $d \cap b \subseteq c$, we have to consider the following possibilities:

(i) $d \subseteq b$

(ii) $d \nsubseteq b$, $d \subseteq a$, $d \subseteq c$

(iii) $d \nsubseteq b$, $d \subseteq a$, $d \nsubseteq c$

(iv) $d \nsubseteq b$, $d \nsubseteq a$, $d \subseteq c$

and (v) $d \nsubseteq a$, $d \nsubseteq b$, $d \nsubseteq c$.

The conditions under which LHS = RHS for each of the above possibilities are given in lemma 3.6.2 to lemma 3.6.5 and lemma 3.6.7 respectively. Hence, the theorem is proved.

Reduction rules for S-expressions

In a finite space of disjoint elements considered in Chapter II, we found that expressions of the forms

$$S(Sabc)xy$$

$$Sx(Sabc)y$$

and $$Sxy(Sabc) \quad \text{where} \quad x,y \in \{a,b,c\}, \qquad (*)$$

can be simplified.

Here we consider the simplication of this type of S-expression when a,b,c are subsets of U.

It is found that, under our assumption, most of the expressions under consideration can be reduced to single variables.

The following is the list of expressions of the forms (*) and their simplified versions, together with the assumption made in each case.

R1. Given $b \subseteq a$, $b \subseteq c$ then

$S(Sabc)ab = Scab$

iff either $a \subseteq c$ or $a \cap c \subseteq b$.

R2. Given $b \subseteq a$ then

$S(Sabc)cb = Sacb$

iff either $c \subseteq a$ or $a \cap c \subseteq b$.

R3. Given $b \subseteq a$, $b \subseteq c$ then $S(Sabc)bc = Sabc$

R4. Given $b \subseteq a$, $b \subseteq c$ then $S(Sabc)ba = Sabc$

R5. Given $b \subseteq a$ then $S(Sabc)ca = a$

R6. Given $b \subseteq a$, $b \subseteq c$ then $S(Sabc)ac = c$

R7. Given $b \subseteq a$, $c \subseteq a$ then $Sa(Sabc)b = b$

R8. Given $b \subseteq a$, $c \subseteq a$ then $Sa(Sabc)c = b \cup c$

R9. Given $c \subseteq a = b$ then $Sb(Sabc)c = b$

R10. Given $c \subseteq a = b$ then $Sb(Sabc)a = b$

R11. Given $b \subseteq a \subseteq b \cup c$ then $Sc(Sabc)a = a$

R12. Given $b \subseteq a \subseteq b \cup c$ then $Sc(Sabc)b = b$

R13. Given $b \subseteq a$ then $Sab(Sabc) = Sabc$

R14. Given $b \subseteq a$, $c \subseteq a$ then $Sac(Sabc) = a$

R15. Given $c \subseteq b \subseteq a$ then $Sbc(Sabc) = a$

R16. Given $a = b$ then $Sba(Sabc) = c$

R17. Given $b \subseteq a \subseteq c$ then $Sca(Sabc) = c$

R18. Given $b \subseteq a$, $b \subseteq c$ then $Scb(Sabc) = Sabc$.

Thus, whenever the given conditions are satisfied, the above equalities can be used as reduction rules.

The proofs of these rules are given as follows:

R1. $S(Sabc)ab = Scab$ iff either $a \subseteq c$ or $a \cap c \subseteq b$.

Assumption: $b \subseteq a$ and $a \subseteq Sabc$

i.e. $a \subseteq (a \cap \bar{b}) \cup c$

$a \cap (\bar{a} \cup b) \subseteq c$

$a \cap b \subseteq c$ i.e. $b \subseteq c$

$S(Sabc)ab = [\{(a \cap \bar{b}) \cup c\} \cap \bar{a}] \cup b$

$= (c \cap \bar{a}) \cup b$

$$Scab = \begin{cases} (c \cap \bar{a}) \cup b & \text{if } a \subseteq c \\ c & \text{if } a \nsubseteq c \end{cases}$$

When $a \nsubseteq c$, LHS = RHS iff

$(c \cap \bar{a}) \cup b = c$

i.e. $[\{(c \cap \bar{a}) \cup b\} \cap \bar{c}] \cup [c \cap (\bar{c} \cup a) \cap \bar{b}] = \emptyset$

$(b \cap \bar{c}) \cup (c \cap a \cap \bar{b}) = \emptyset$

$(c \cap a \cap \bar{b}) = \emptyset$ $\quad \because b \subseteq c$

i.e. $a \cap c \subseteq b$

R2. $S(Sabc)cb = Sacb$ iff either $c \subseteq a$ or $a \cap c \subseteq b$

Assumption: $b \subseteq a$

So $S(Sabc)cb = S\{(a \cap \bar{b}) \cup c\}cb$

$= [\{(a \cap \bar{b}) \cup c\} \cap \bar{c}] \cup b$

$= (a \cap \bar{b} \cap \bar{c}) \cup b$

$= (a \cap \bar{c}) \cup b$

$$Sacb = \begin{cases} (a \cap \bar{c}) \cup b & \text{if } c \subseteq a \\ a & \text{if } c \nsubseteq a \end{cases}$$

When $c \nsubseteq a$, LHS = RHS

iff $a \cap c \subseteq b$

by the same argument as in R1.

R3. $S(Sabc)bc = Sabc$

Assumption: $b \subseteq a$ and $b \subseteq Sabc$

$\therefore S(Sabc)bc = [\{(a \cap \bar{b}) \cup c\} \cap \bar{b}] \cup c$ by assumption

$= (a \cap \bar{b}) \cup (c \cap \bar{b}) \cup c$

$= (a \cap \bar{b}) \cup c$

$= Sabc$ $\quad \because b \subseteq a$

R4. $S(Sabc)ba = Sabc$

Assumption: $b \subseteq a$, $b \subseteq Sabc$

i.e. $b \subseteq (a \cap \bar{b}) \cup c$

$b \cap (\bar{a} \cup b) \subseteq c$

i.e. $b \subseteq c$ $\quad \because\ b \subseteq a \Rightarrow b \cap \bar{a} = \emptyset$

$\therefore\ S(Sabc)ba = [\{(a \cap \bar{b}) \cup c\} \cap \bar{b}] \cup a$ by assumption

$= (a \cap \bar{b}) \cup (c \cap \bar{b}) \cup a$

$= (c \cap \bar{b}) \cup a$

$= Scba$ $\quad \because\ b \subseteq c$

R5. $S(Sabc)ca = a$

Assumption: $b \subseteq a$

So $S(Sabc)ca = S\{(a \cap \bar{b}) \cup c\}ca$

$= [\{(a \cap \bar{b}) \cup c\} \cap \bar{c}] \cup a$

$= (a \cap \bar{b} \cap \bar{c}) \cup a$

$= a$

R6. $S(Sabc)ac = c$

Assumption: $b \subseteq a$, $a \subseteq Sabc$

$S(Sabc)ac = [\{(a \cap \bar{b}) \cup c\} \cap \bar{a}] \cup c$

$= (c \cap \bar{a}) \cup c$

$= c$

R7. $Sa(Sabc)b = b$

Assumption: $b \subseteq a$, $Sabc \subseteq a$

i.e. $(a \cap \bar{b}) \cup c \subseteq a$

$\{(a \cap \bar{b}) \cup c\} \cap \bar{a} = \emptyset$

$c \cap \bar{a} = \emptyset$

$\therefore\ c \subseteq a$

$Sa(Sabc)b = [a \cap \overline{\{(a \cap \bar{b}) \cup c\}}] \cup b$ by assumption

$= \{a \cap (\bar{a} \cup b) \cap \bar{c}\} \cup b$

$= (a \cap b \cap \bar{c}) \cup b$

$= b$

R8. $Sa(Sabc)c = b \cup c$

Assumption: $b \subseteq a$, $Sabc \subseteq a$

i.e. $b \subseteq a$, $c \subseteq a$ from (R7)

$\therefore Sa(Sabc)c = (a \cap b \cap \bar{c}) \cup c$

$= (b \cap \bar{c}) \cup c$ $\quad \because b \subseteq a$

$= b \cup c$

R9. $Sb(Sabc)c = b$

Assumption: $b \subseteq a$, $Sabc \subseteq b$

i.e. $(a \cap \bar{b}) \cup c \subseteq b$

$\{(a \cap \bar{b}) \cup c\} \cap \bar{b} = \emptyset$

$(a \cap \bar{b}) \cup (c \cap \bar{b}) = \emptyset$ $\quad$ i.e. $a \subseteq b$ and $c \subseteq b$

So the assumption is $c \subseteq a = b$

$\therefore Sb(Sabc)c = Sbcc$

$= b$

R10. $Sb(Sabc)a = b$

Assumption: $b \subseteq a$, $Sabc \subseteq b$

i.e. $c \subseteq a = b$ from (R9)

$\therefore Sb(Sabc)a = Sbcb$

$= b$

R11. $Sc(Sabc)a = a$

Assumption: $b \subseteq a$, $Sabc \subseteq c$

i.e. $(a \cap \bar{b}) \cup c \subseteq c$

i.e. $a \subseteq b \cup c$

$Sc(Sabc)a = [c \cap \overline{\{(a \cap \bar{b}) \cup c\}}] \cup a$ by assumption

$= \{c \cap (\bar{a} \cup b) \cap \bar{c}\} \cup a$

$= a$

R12. $Sc(Sabc)b = b$

Assumption: $b \subseteq a$, $Sabc \subseteq c$

$\therefore Sc(Sabc)b = \{c \cap (\bar{a} \cup b) \cap \bar{c}\} \cup b$

$= b$

R13. $Sab(Sabc) = Sabc$

Assumption: $b \subseteq a$

$\therefore Sab(Sabc) = (a \cap \overline{b}) \cup (a \cap \overline{b}) \cup c$

$= (a \cap \overline{b}) \cup c$

$= Sabc \qquad \because b \subseteq a$

R14. $Sac(Sabc) = a$

Assumption: $b \subseteq a,\ c \subseteq a$

$\therefore Sac(Sabc) = (a \cap \overline{c}) \cup (a \cap \overline{b}) \cup c$

$= (a \cap (\overline{c} \cup \overline{b})\} \cup c$

$= a$

R15. $Sbc(Sabc) = a$

Assumption: $b \subseteq a,\ c \subseteq b$

$\therefore Sbc(Sabc) = (b \cap \overline{c}) \cup (a \cap \overline{b}) \cup c$

$= (b \cup c) \cup (a \cap \overline{b})$

$= c \cup b \cup a$

$= a$

R16 $Sba(Sabc) = c$

Assumption: $b \subseteq a,\ a \subseteq b$ i.e. $a = b$

$\therefore Sba(Sabc) = c$

R17. $Sca(Sabc) = c$

Assumption: $b \subseteq a \subseteq c$

$\therefore Sca(Sabc) = (c \cap \overline{a}) \cup (a \cap \overline{b}) \cup c$

$= (a \cap \overline{b}) \cup c$

$= c \qquad \because a \subseteq c$

R18. $Scb(Sabc) = Sabc$

Assumption: $b \subseteq a,\ b \subseteq c$

$$\therefore\ Scb(Sabc) = (c \cap \overline{b}) \cup (a \cap \overline{b}) \cup c$$
$$= (a \cap \overline{b}) \cup c$$
$$= Sabc$$

§3.B Further properties of S

In this section, we focus our attention on expressions $Sabc$ in which a,b,c are of the form $x \cup y$ or $x \cap y$, where x,y are subsets of U.

In what follows, we show that, under certain conditions, the operator S and the set operations $\cap, \cup$ are distributive over each other.

Theorem 3.7

Given $c \subseteq a \cap b$, then

$$S(a \cap b)cd = Sacd \cap Sbcd.$$

Proof

$$LHS = \{(a \cap b) \cap \overline{c}\} \cup d \qquad \because\ c \subseteq a \cap b$$

$c \subseteq a \cap b \Rightarrow c \subseteq a$ and $c \subseteq b$

$$\therefore\ RHS = \{(a \cap \overline{c}) \cup d\} \cap \{(b \cap \overline{c}) \cup d\}$$
$$= \{(a \cap \overline{c}) \cap (b \cap \overline{c})\} \cup d$$
$$= (a \cap b \cap \overline{c}) \cup d$$
$$= LHS$$

Theorem 3.8

Given $c \subseteq a \cup b$, then

$$S(a \cup b)cd = Sacd \cup Sbcd$$

iff one of the following holds:

(i) $c \subseteq a \cap b$

(ii) $c \subseteq a$, $c \nsubseteq b$, $b \cap c \subseteq d$

(iii) $c \nsubseteq a$, $c \subseteq b$, $a \cap c \subseteq d$

(iv) $c \nsubseteq a$, $c \nsubseteq b$, $c \subseteq d \subseteq a \cup b$

Proof

$LHS = \{(a \cup b) \cap \overline{c}\} \cup d$ $\quad \because c \subseteq a \cup b$

$= (a \cap \overline{c}) \cup (b \cap \overline{c}) \cup d$

For RHS, we consider the four possibilities:

(i) $c \subseteq a$, $c \subseteq b$:

$\Rightarrow RHS = (a \cap \overline{c}) \cup d \cup (b \cap \overline{c}) \cup d$

$= (a \cap \overline{c}) \cup (b \cap \overline{c}) \cup d$

$= LHS$

(ii) $c \subseteq a$, $c \nsubseteq b$:

$\Rightarrow RHS = (a \cap \overline{c}) \cup d \cup b$

$\supseteq LHS$

$\therefore$ LHS = RHS iff RHS $\subseteq$ LHS

i.e. $(a \cap \overline{c}) \cup d \cup b \subseteq (a \cap \overline{c}) \cup (b \cap \overline{c}) \cup d$

$b \subseteq (a \cap \overline{c}) \cup (b \cap \overline{c}) \cup d$

$b \cap (\overline{a} \cup c) \cap (\overline{b} \cup c) \subseteq d$

$(\overline{a} \cup c) \cap (b \cap c) \subseteq d$

$b \cap c \subseteq d$

Hence if $c \subseteq a$ and $c \nsubseteq b$, then LHS = RHS iff $b \cap c \subseteq d$

(iii) $c \nsubseteq a$, $c \subseteq b$:

$\Rightarrow RHS = a \cup (b \cap \overline{c}) \cup d$

$\supseteq LHS$

$\therefore$ LHS = RHS iff RHS $\subseteq$ LHS

i.e. $a \cup (b \cap \bar{c}) \cup d \subseteq (a \cap \bar{c}) \cup (b \cap \bar{c}) \cup d$

$a \subseteq (a \cap \bar{c}) \cup (b \cap \bar{c}) \cup d$

By interchanging a and b in (ii), we see that for $c \nsubseteq a$ and $c \subseteq b$, LHS = RHS iff $a \cap c \subseteq d$.

(iv) $c \nsubseteq a$, $c \nsubseteq b$:

$\Rightarrow$ RHS = $a \cup b$

$\therefore$ LHS = RHS iff

$\{(a \cup b) \cap \bar{c}\} \cup d = a \cup b$

i.e. iff $[\{(a \cup b) \cap \bar{c}\} \cup d] \cap \overline{(a \cup b)} = \emptyset$

$d \cap \bar{a} \cap \bar{b} = \emptyset$

$d \subseteq a \cup b$

and $(a \cup b) \cap [\overline{\{(a \cup b) \cap \bar{c}\}} \cup d] = \emptyset$

$(a \cup b) \cap [\{\overline{(a \cup b)} \cup c\} \cap \bar{d}] = \emptyset$

$(a \cup b) \cap c \cap \bar{d} = \emptyset$

$c \cap \bar{d} = \emptyset$ $\quad \because c \subseteq (a \cup b)$

$c \subseteq d$

$\therefore$ For $c \nsubseteq a$ and $c \nsubseteq b$, LHS = RHS iff $c \subseteq d \subseteq a \cup b$.

Theorem 3.9

$Sab(c \cap d) = Sabc \cap Sabd$

Proof

$b \nsubseteq a \Rightarrow$ LHS $= a$ and RHS $= a \cap a = a$

$b \subseteq a \Rightarrow$

LHS $= (a \cap \overline{b}) \cup (c \cap d)$

RHS $= \{(a \cap \overline{b}) \cup c\} \cap \{(a \cap \overline{b}) \cup d\}$

$= (a \cap \overline{b}) \cup (c \cap d)$

$=$ LHS.

Theorem 3.10

$Sab(c \cup d) = Sabc \cup Sabd$.

Proof

$b \nsubseteq a \Rightarrow$ RHS $= a \cup a = a =$ LHS

$b \subseteq a \Rightarrow$

LHS $= (a \cap \overline{b}) \cup (c \cup d)$

RHS $= (a \cap \overline{b}) \cup c \cup (a \cap \overline{b}) \cup d$

$= (a \cap \overline{b}) \cup (c \cup d)$

$=$ LHS.

Theorem 3.11

Given $b \cap c \subseteq a$, then

$Sa(b \cap c)d = Sabd \cup Sacd$

iff one of the following holds:

(i) $b \subseteq a$, $c \subseteq a$

(ii) $b \subseteq a$, $c \nsubseteq a$, $b \cap c \subseteq d$

(iii) $b \nsubseteq a$, $c \subseteq a$, $b \cap c \subseteq d$

(iv) $b \nsubseteq a$, $c \nsubseteq a$, $b \cap c \subseteq d \subseteq a$

Proof

$$\text{LHS} = \{a \cap \overline{(b \cap c)}\} \cup d$$

$$= \{a \cap (\bar{b} \cup \bar{c})\} \cup d$$

$$= (a \cap \bar{b}) \cup (a \cap \bar{c}) \cup d$$

Given $b \cap c \subseteq a$, there are four possible cases to be considered:

(i) $b \subseteq a$, $c \subseteq a$:

$$\Rightarrow \text{RHS} = \{(a \cap \bar{b}) \cup d\} \cup \{(a \cap \bar{c}) \cup d\}$$

$$= (a \cap \bar{b}) \cup (a \cap \bar{c}) \cup d$$

$$= \text{LHS}$$

(ii) $b \subseteq a$, $c \nsubseteq a$:

$$\Rightarrow \text{RHS} = (a \cap \bar{b}) \cup d \cup a$$

$$\supseteq \text{LHS}$$

$\therefore$ LHS = RHS iff RHS $\subseteq$ LHS

i.e. $(a \cap \bar{b}) \cup d \cup a \subseteq (a \cap \bar{b}) \cup (a \cap \bar{c}) \cup d$

$$a \subseteq (a \cap \bar{b}) \cup (a \cap \bar{c}) \cup d$$

$$a \cap (\bar{a} \cup b) \cap (\bar{a} \cup c) \subseteq d$$

$$a \cap b \cap (\bar{a} \cup c) \subseteq d$$

$$a \cap b \cap c \subseteq d$$

i.e. $b \cap c \subseteq d$ $\quad \because b \cap c \subseteq a$

$\therefore$ When $b \subseteq a$, $c \nsubseteq a$, LHS = RHS iff $b \cap c \subseteq d$.

(iii) $b \nsubseteq a$, $c \subseteq a$:

$$\Rightarrow \text{RHS} = a \cup (a \cap \bar{c}) \cup d$$

So by the same argument as in (ii), LHS = RHS iff $b \cap c \subseteq d$.

(iv) $b \nsubseteq a$, $c \nsubseteq a$

$$\Rightarrow \text{RHS} = a \cup a$$

$$= a$$

LHS = RHS

iff $Sa(b \cap c)d = a$

i.e. iff $b \cap c \subseteq d \subseteq a$ by lemma 3.6.6.

Note: The condition $b \cap c \not\subseteq a$ is sufficient for the above result to hold.

$\therefore$ $b \cap c \not\subseteq a \Rightarrow b \not\subseteq a$ and $c \not\subseteq a$

$\therefore$ LHS = $Sa(b \cap c)d = a$

and RHS = $Sabd \cup Sacd = a \cup a = a$

Hence Theorem 3.11 may be stated as follows:

Theorem 3.11

$Sa(b \cap c)d = Sabd \cup Sacd$

iff one of the following holds.

(i) $b \cap c \not\subseteq a$

(ii) $b \cup c \subseteq a$

(iii) $b \subseteq a$, $c \not\subseteq a$, $b \cap c \subseteq d$

(iv) $b \not\subseteq a$, $c \subseteq a$, $b \cap c \subseteq d$

(v) $b \not\subseteq a$, $c \not\subseteq a$, $b \cap c \subseteq d \subseteq a$

Theorem 3.12

Given $b \cup c \subseteq a$, then

$Sa(b \cup c)d = Sabd \cap Sacd$

Proof

LHS = $\{a \cap \overline{(b \cup c)}\} \cup d$

$= (a \cap \bar{b} \cap \bar{c}) \cup d$

$b \cup c \subseteq a \Rightarrow b \subseteq a$ and $c \subseteq a$

$\therefore$ RHS $= \{(a \cap \overline{b}) \cup d\} \cap \{(a \cap \overline{c}) \cup d\}$

$= (a \cap \overline{b} \cap \overline{c}) \cup d$

$=$ LHS.

Theorem 3.13

Given $c \subseteq b$, then

$a \cap Sbcd = S(a \cap b)(a \cap c)(a \cap d)$

Proof

LHS $= a \cap \{(b \cap \overline{c}) \cup d\}$ $\quad \because \ c \subseteq b$

$= (a \cap b \cap \overline{c}) \cup (a \cap d)$

Now $c \subseteq b \Rightarrow a \cap c \subseteq a \cap b$

$\therefore$ RHS $= \{(a \cap b) \cap \overline{(a \cap c)}\} \cup (a \cap d)$

$= \{(a \cap b) \cap (\overline{a} \cup \overline{c})\} \cup (a \cap d)$

$= (a \cap b \cap \overline{c}) \cup (a \cap d)$

$=$ LHS.

Theorem 3.14

Given $c \subseteq b$, then

$a \cup Sbcd = S(a \cup b)(a \cup c)(a \cup d)$

Proof

LHS $= a \cup (b \cap \overline{c}) \cup d$ $\quad \because \ c \subseteq b$

Now $c \subseteq b \Rightarrow a \cup c \subseteq a \cup b$

$\therefore$ RHS $= \{(a \cup b) \cap \overline{(a \cup c)}\} \cup (a \cup d)$

$= \{(a \cup b) \cap (\overline{a} \cap \overline{c})\} \cup (a \cup d)$

$= (b \cap \overline{a} \cap \overline{c}) \cup a \cup d$

$= (b \cap \overline{c}) \cup a \cup d$

$=$ LHS.

§3.C The application of S to Boolean Algebras

Formulation of set operations by S-expressions

Let a,b be subsets of the universal set U. Then the operations $\bar{a}$, $a \cup b$ and $a \cap b$ may be written in terms of the operator S as follows:

$\bar{a}$	is represented by	$SUa\emptyset$	(3.15)
$a \cup b$	is represented by	$Sa\emptyset b$	(3.16)
$a \cap b$	is represented by	$SU\{S(SUa\emptyset)\emptyset(SUb\emptyset)\}\emptyset$	(3.17)
	or	$SU\{SUa(SUb\emptyset)\}\emptyset$	(3.18)

This we verify by referring to the definition of S given in 3.A

$$\text{i.e.} \quad Sabc = \begin{cases} (a \cap \bar{b}) \cup c & \text{if } b \subseteq a \\ a & \text{if } b \nsubseteq a \end{cases}$$

where $a,b,c \subseteq U$.

That is,

$$\begin{aligned} SUa\emptyset &= (U \cap \bar{a}) \cup \emptyset && \because a \subseteq U \\ &= \bar{a} \\ Sa\emptyset b &= (a \cap \bar{\emptyset}) \cup b && \because \emptyset \subseteq a \\ &= (a \cap U) \cup b \\ &= a \cup b \end{aligned}$$

$$\begin{aligned} SU\{S(SUa\emptyset)\emptyset(SUb\emptyset)\}\emptyset &= SU\{Sa\bar{\emptyset}b\}\emptyset && \text{by (3.15)} \\ &= SU(\bar{a} \cup \bar{b})\emptyset && \text{by (3.16)} \\ &= \overline{(\bar{a} \cup \bar{b})} && \text{by (3.15)} \\ &= a \cap b \end{aligned}$$

$$\begin{aligned} \text{and } SU\{SUa(SUb\emptyset)\}\emptyset &= SU\{SUab\}\emptyset && \text{by (3.15)} \\ &= SU\{(U \cap \bar{a}) \cup \bar{b}\}\emptyset && \because a \subseteq U \\ &= SU(\bar{a} \cup \bar{b})\emptyset \\ &= a \cap b && \text{from above} \end{aligned}$$

Note

1. Two representations, (3.17) and (3.18), are given for $a \cap b$.

   The S-expression in (3.18) has the advantage of being shorter; but in several cases, the formulation given by (3.17) lends itself more readily to manipulation.

   In what follows, the two representations will be treated as interchangeable.

2. The S-expression given in (3.15) - (3.18) have the following properties:

   Either a) The first operand is the universal set U,

   or b) The second operand is the empty set $\emptyset$.

   This means that every S-expression under consideration, being a composition of the expression in (3.15) - (3.18) conforms to the assumption made in 3.A that in every expression Sabc, $b \subseteq a$.

A formulation of the theory of Boolean algebras

In this work, we adopt a natural formulation of the theory of Boolean algebras as given in Stoll [38] Chapter 6, which is quoted below:

A Boolean algebra is a 6-tuple $\langle B, \vee, \wedge, ', 0, 1 \rangle$ where

B is a set,

$\vee$ and $\wedge$ are binary operations,

$'$ is a binary relation in B having B as its domain,

0 and 1 are distinct elements of B,

and the following axioms are satisfied for all $a, b, c \in B$.

(i) Each operation is associative:

$$a \vee (b \vee c) = (a \vee b) \vee c$$

and $$a \wedge (b \wedge c) = (a \wedge b) \wedge c$$

(ii) Each operation is commutative:

$$a \vee b = b \vee a$$

and $$a \wedge b = b \wedge a$$

(iii) Each operation distributes over the other:

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

and $$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$

(iv) $$a \vee 0 = a$$

and $$a \wedge 1 = a$$

(v) For each $a \in B$, $\exists$ a related element a' such that

$$a \vee a' = 1$$

and $$a \wedge a' = 0$$

By identifying

B with $P(U)$, the power set of U

$\vee$ and $\wedge$ with set-union ($\cup$) and set-intersection ($\cap$)

0 and 1 with $\emptyset$ and U,

the above set of axioms may be derived from the theorems of S as shown in the next section.

Derivation of the axioms from the properties of S

Each axiom (i) - (v) consists of a pair of statements. We show that the axioms can be derived from the properties of S in ten separate theorems as follows:

Theorem 3.19 $a \cup (b \cup c) = (a \cup b) \cup c$

Proof

$a \cup (b \cup c) = Sa\emptyset(b \cup c)$ by (3.16)

$= Sa\emptyset(Sb\emptyset c)$ by (3.16)

$= S(Sa\emptyset b)\emptyset c$ by theorem 3.3*

$= S(a \cup b)\emptyset c$ by (3.16)

$= (a \cup b) \cup c$ by (3.16)

Theorem 3.20 $a \cap (b \cap c) = (a \cap b) \cap c$

Proof

$a \cap (b \cap c) = SU[S(SUa\emptyset)\emptyset(SU(b \cap c)\emptyset)]\emptyset$ by (3.17)

$= SU[S(SUa\emptyset)\emptyset\{SUb\emptyset \cup SUc\emptyset\}]\emptyset$ by theorem 3.11

$= SU[S(SUa\emptyset)\emptyset\{S(SUb\emptyset)\emptyset(SUc\emptyset)\}]\emptyset$ by (3.16)

$= SU[S\{S(SUa\emptyset)\emptyset(SU_b\emptyset)\}\emptyset(SUc\emptyset)]\emptyset$ by theorem 3.3*

$= SU[S\{SUa\emptyset \cup SUb\emptyset\}\emptyset(SUc\emptyset)]\emptyset$ by (3.16)

$= SU[S(SU(a \cap b)\emptyset)\emptyset(SUc\emptyset)]\emptyset$ by theorem 3.11

$= (a \cap b) \cap c$ by (3.17)

Theorem 3.21 $a \cup b = b \cup a$

Proof

$a \cup b = Sa\emptyset b$ by (3.16)

$= Sb\emptyset a$ by theorem 3.2

$= b \cup a$ by (3.16)

Theorem 3.22 $a \cap b = b \cap a$

Proof

$a \cap b = SU[S(SUa\emptyset)\emptyset(SUb\emptyset)]\emptyset$ by (3.17)

$= SU[S(SUb\emptyset)\emptyset(SUa\emptyset)]\emptyset$ by theorem 3.2

$= b \cap a$ by (3.17)

Theorem 3.23 $a \cup (b \cap c) = (a \cup b) \cap (a \cup c)$

Proof

$a \cup (b \cap c) = Sa\emptyset(b \cap c)$ by (3.16)

$= (Sa\emptyset b) \cap (Sa\emptyset c)$ by theorem 3.9

$= (a \cup b) \cap (a \cup c)$ by (3.16)

Theorem 3.24 $a \cap (b \cup c) = (a \cap b) \cup (a \cap c)$

Proof

$a \cap (b \cup c) = SU[S(SUa\emptyset)\emptyset(SU(b \cup c)\emptyset)]\emptyset$ by (3.17)

$= SU[S(SUa\emptyset)\emptyset\{(SUb\emptyset) \cap (SUc\emptyset)\}]\emptyset$ by theorem 3.12

$= SU[\{S(SUa\emptyset)\emptyset(SUb\emptyset)\} \cap \{S(SUa\emptyset)\emptyset(SUc\emptyset)\}]\emptyset$ by theorem 3.9

$= [SU\{S(SUa\emptyset)\emptyset(SUb\emptyset)\}\emptyset] \cup [SU\{S(SUa\emptyset)\emptyset (SUc\emptyset)\}\emptyset]$ by theorem 3.11

$= (a \cap b) \cup (a \cap c)$ by (3.17)

Theorem 3.25 $a \cup \emptyset = a$

Proof

$a \cup \emptyset = Sa\emptyset\emptyset$ by (3.16)

$= a$ by theorem 3.1

Theorem 3.26 $a \cap U = a$

Proof

$a \cap U = SU[SUa(SUU\emptyset)]\emptyset$		by (3.18)
$= SU[SUa\emptyset]\emptyset$		by theorem 3.1
$= a \cup \emptyset$		by R8
$= a$		by theorem 3.25

Theorem 3.27 $a \cup \bar{a} = U$

Proof

$a \cup \bar{a} = a \cup (SUa\emptyset)$		by (3.15)
$= Sa\emptyset(SUa\emptyset)$		by (3.16)
$= U$		by R15

Theorem 3.28 $a \cap \bar{a} = \emptyset$

Proof

$a \cap \bar{a} = a \cap (SUa\emptyset)$		by(3.15)
$= SU[SUa\{SU(SUa\emptyset)\emptyset\}]\emptyset$		by (3.18)
$= SU[SUaa]\emptyset$	as in theorem 3.26	
$= SUU\emptyset$		by theorem 3.1
$= \emptyset$		

Conclusion

We have shown that the set of axioms of a Boolean algebra can be derived from the properties of S.

Hence, from the isomorphism of Boolean algebras and the algebra of sets, we can conclude the following:

That the theory of Boolean algebras and the algebra of sets can be completely represented within the framework of S.

Moreover, by virtue of theorems 3.19 to 3.23 and theorem 3.24 following, we observe that the algebra of $\mathcal{S}$ is also a lattice [33].

Theorem 3.24 $\quad a \cap a = a \cup a = a$

Proof

$a \cap a$	$=$	$SU[S(SUa\emptyset)\emptyset(SUa\emptyset)]\emptyset$	by (3.17)
	$=$	$SU(SUa\emptyset)\emptyset$	by theorem 3.1
	$=$	$Sa\emptyset\emptyset$	by R8
	$=$	a	by theorem 3.1
$a \cup a$	$=$	$Sa\emptyset a$	by (3.16)
	$=$	a	by theorem 3.1

CHAPTER 4

SIMPLE PROGRAMS AND THEIR SPECIFICATIONS

§4.A Names, Values and the Evaluation Operator $\varkappa$

Let N be a set of names and V be a set of values which includes the boolean values T,F and the unassigned value ω.

Then define $\varkappa$, a function with domain $N \cup V$ and range $R \subset V$, as follows:

Definition 4.1

The function $\varkappa: N \cup V \to V$ is such that

$$\varkappa x = \alpha$$

where $\alpha \in V$ is the value corresponding to $x \in N$, and

$$\varkappa\beta = \beta \quad \forall\, \beta \in V.$$

Intuitively, '$\varkappa x$' may be regarded as 'the value of x' and the name of a constant is the constant itself.

Let X denote a set whose elements are an alternate sequence of corresponding names and values.

X is formally defined by

Definition 4.2

Let p_i be a pair

$$p_i = \{n_i, v_i\} \quad n_i \in N \cup V,\ v_i \in V$$

such that $\varkappa n_i = v_i$.

Then

$$X = \bigcup_i p_i, \quad i \in \mathbb{N}$$

which, for simplicity, we shall always write as

$$X = \{n_1, v_1, n_2, v_2, \ldots\}$$

where $n_i \in N \cup V$, $v_i \in V$ for $i=1,2,\ldots$.

Definition 4.3

Let $P(X)$ denote the power set of X .

Then $P^*(X) \subseteq P(X)$

is such that $Y \in P^*(X)$

iff $n_i \in Y \Leftrightarrow v_i \in Y$.

Examples

Let $a,b,c \in N$, then

(i) $\{a,2,b,5,c,7\} \in P^*(X)$

with $\nu a = 2$, $\nu b = 5$, $\nu c = 7$.

(ii) $\{a,2,b,5,c,2\} \in P^*(X)$

with $\nu a = 2$, $\nu b = 5$, $\nu c = 2$.

(iii) $\{a,2,3,3,c,5\} \in P^*(X)$

with $\nu a = 2$, $\nu 3 = 3$, $\nu c = 5$.

(iv) $\{a,2,b,3,5,c,7\} \notin P^*(X)$.

Note

In the set X , although $n_i \neq n_j$ for $i \neq j$, the v_i's are not necessarily distinct.

It should be observed here that the ordering relationship in the set X is such as to preserve the relationship ν between a name and its value. No restriction is imposed upon the order of appearance of the name-value pairs.

Thus, if A and B are elements of $P^*(X)$ where

$$A = \{n_1, v_1, n_2, v_2, \ldots\ldots, n_r, v_r\} \quad n_i \in N,\ v_i \in V$$

$$B = \{m_1, u_1, m_2, u_2, \ldots\ldots, m_s, u_s\} \quad m_i \in N,\ u_i \in V\ ,$$

the relations of inclusion and equality of subsets of X are defined as follows:

Definition 4.4

A is included in B , written $A \subseteq B$,

iff

(i) $\{n_1, n_2, \ldots\ldots, n_r\} \subseteq \{m_1, m_2, \ldots\ldots, m_s\}$

ie. the set of names in A is included in the set of names in B,

and

(ii) if $n_i = m_j$ then $v_i = u_j$.

Definition 4.5

The sets A and B are said to be equal, $A = B$ iff $A \subseteq B$ and $B \subseteq A$.

eg. If $A = \{a,1,b,2,c,3\}$

$B = \{a,3,b,2,c,1\}$

$C = \{c,3,a,1,b,2\}$

$D = \{a,3,c,1\}$

then $D \subsetneq B$, $A = C$ but $A \neq B$.

Moreover, we need to define more precisely what is meant by the empty set of X and the operations union and intersection on X .

Definition 4.6

The intersection of A and B , $A \cap B$., is the set

$$A \cap B = \{x_1, y_1, x_2, y_2, \ldots, x_t, y_t\}$$

such that

(i) $\{x_1, \ldots, x_t\} = \{n_1, \ldots, n_r\} \cap \{m_1, \ldots, m_s\}$

(ii) For each i, $\exists$ j,l such that

$$x_i = n_j = m_l$$

$$\text{and } y_i = \begin{cases} v_j = u_l & \text{if } v_j = u_l \\ \omega & \text{otherwise.} \end{cases}$$

It follows that $A \cap B = \emptyset$

iff $\{n_1, n_2, \ldots, n_r\} \cap \{m_1, m_2, \ldots, m_s\} = \emptyset$.

eg. If

$A = \{a,3,b,2,c,4\}$

$B = \{b,2,c,5,d,4\}$

then

$A \cap B = \{b,2,c,\omega\}$.

Definition 4.7

The union of A and B, $A \cup B$, is the set

$$A \cup B = \{x_1, y_1, x_2, y_2, \ldots, x_t, y_t\}$$

such that

(i) $\{x_1, \ldots, x_t\} = \{n_1, \ldots, n_r\} \cup \{m_1, \ldots, m_s\}$

(ii) If $x_i \notin \{n_1, \ldots, n_r\} \cap \{m_1, \ldots, m_s\}$

then either $x_i = n_j$ and $y_i = v_j$ for some j

or $x_i = m_l$ and $y_i = u_l$ for some l

(iii) If $x_i \in \{n_1,\ldots,n_r\} \cap \{m_1,\ldots,m_s\}$,

let $x_i = n_j = m_l$, then

$$y_i = \begin{cases} v_j = u_l & \text{if } v_j = u_l \\ \omega & \text{otherwise} \end{cases}$$

eg. From the previous example,

$$A \cup B = \{a,3,b,2,c,\omega,d,4\}$$

Having defined the sets of our particular interest, we now examine the operation of the substitution operator S on the set $P^*(X)$ and set up a model of a sequence of assignments in terms of S.

§4.B A Model of a Sequence of Simple Assignment Statements

An assignment statement

$$x := y$$

causes the value of the variable y to be assigned as the value of the variable x.

The same effect may be achieved by the action of the substitution operator S upon suitable operands.

For any subsets a,b,c of an arbitrary, finite set U, $Sabc$ is defined in Chapter III by

$$Sabc = \begin{cases} (a \wedge \bar{b}) \vee c & \text{if } b \subseteq a \\ a & \text{if } b \not\subseteq a. \end{cases}$$

For the purpose of modelling an assignment statement, we shall from now on consider the first operand a as an element of $P^*(X)$ and b,c as members of the value set.

Let $a \in P^*(X)$ be the set

$$a = \{n_1, v_1, n_2, v_2, \ldots, n_l, v_l\}$$

where $n_i \in N$, $v_i \in V$ for all $i = 1, \ldots, l$.

Consider the following two cases:-

(i) If $b = \varkappa n_i \quad n_i \in N$

$c = \varkappa n_j \quad n_j \in N$, then

$$Sabc = S\{n_1, v_1, n_2, v_2, \ldots, n_l, v_l\}\ \varkappa n_i\ \varkappa n_j$$

$$= \begin{cases} \{n_1, v_1, n_2, v_2, \ldots, n_i, v_j, \ldots, n_j, v_j, \ldots, n_l, v_l\} & \text{if } n_i, n_j \in a \\ a & \text{otherwise} \end{cases}$$

(ii) If $b = \varkappa n_i \quad n_i \in N$

$c = \varkappa\alpha \quad \alpha \in V$ then

$$Sabc = S\{n_1, v_1, n_2, v_2, \ldots, n_l, v_l\}\ \varkappa n_i\ \varkappa\alpha$$

$$= \begin{cases} \{n_1, v_1, n_2, v_2, \ldots, n_i, \alpha, \ldots, n_l, v_l\} & \text{if } n_i \in a \\ a & \text{otherwise} \end{cases}$$

eg.

(i) $a = \{x, 1, y, 3, z, 5\}$, $b = \varkappa x$, $c = \varkappa z$

then

$$Sabc = S\{x, 1, y, 3, z, 5\}\ \varkappa x\ \varkappa z$$

$$= \{x, 5, y, 3, z, 5\}$$

(ii) a = {x,1,y,3,z,5}, b = νy, c = ν8

then

Sabc = S{x,1,y,3,z,5} νy ν8

= {x,1,y,8,z,5}

(iii) a = {x,1,y,3,z,5}, b = νt, c = νz

then

Sabc = S{x,1,y,3,z,5} νt νz

= {x,1,y,3,z,5}.

Remark

If the first operand of S is a set in which some names have the same value, the above definition can still be used without ambiguity. As even though two values νx, νz are the same, the symbols νx and νz refer to different elements of the set, ie. νx refers to the value immediately following the name x and νz to the value immediately following the name z. This is illustrated in the following example:

a = {x,1,y,2,z,1}, b = νx, c = νy

then

Sabc = S{x,1,y,2,z,1} νx νy

= {x,2,y,2,z,1}

From the definition of S given above, a simple assignment statement of the form

x := y

may be represented by the S-expression

$$S\{x,\theta\}\ {}_{\kappa}x\ {}_{\kappa}y$$

where θ denotes any value including the unassigned value ω and $x, y \in N$.

Sequence of assignment statements

To establish a model of a sequence of assignment statements, we first define what is meant by the 'variable space' of a program.

Definition 4.8

The variable space μ of a program is a set $\in P^*(X)$ consisting of all the variables in the program together with their values.

At the beginning of each program, all the variables in μ are defined to have an unassigned value

ie. $\mu = \{n_1,\omega,n_2,\omega,\ldots,n_l,\omega\}$

where $n_1,\ldots,n_l \in N$ are the variables contained in the program.

Given a program with variable space μ, each step in the program may be regarded as an alteration of μ.

Thus, for $x,y \in \mu$, the model of

$$x := y$$

is defined to be

$$S\,\mu\ {}_{\kappa}x{}_{\kappa}y \qquad (4.9)$$

Note

Though, according to definition 4.1, $\varkappa x \in V$ means that x could belong to either N or V, the validity of the assignment

$$x := y$$

implies that x must be an element of N.

Now consider two successive assignment statements

$$x := y$$
$$y := z .$$

According to (4.9), these are modelled by two S-expressions

$$S_{\mu\varkappa x \varkappa y} \tag{4.10}$$
$$S_{\mu\varkappa y \varkappa z} . \tag{4.11}$$

It is important to note here that μ in the expressions (4.10) and (4.11) refers to different states of the variable space.

For suppose that in (4.10)

$$\mu = \mu_1 = \{x,\omega,y,\alpha,z,\beta\} \quad x,y,z \in N$$
$$\alpha,\beta,\omega \in V$$

then, after the execution of (4.10), μ takes the value μ_2 where

$$\mu_2 = S_{\mu_1 \varkappa x \varkappa y} = \{x,\alpha,y,\alpha,z,\beta\} .$$

Therefore, in (4.11), μ is in the state μ_2 which is different from μ_1.

Thus, strictly speaking, the expression (4.11) is

$$S(S_{\mu n x n y) n y n z} \cdot$$

In general, each line in a sequence

$$\begin{array}{l} S_{\mu n x_1 n y_1} \\ S_{\mu n x_2 n y_2} \\ \quad \vdots \\ S_{\mu n x_n n y_n} \end{array} \qquad (4.12)$$

represents a successive alteration of the variable space μ. Furthermore, μ in an expression refers to the new state of μ pertaining after the operation of the previous line.

Therefore, the sequence (4.12) is in fact a sequence of nested S-expressions:

$$\begin{array}{l} S_{\mu n x_1 n y_1} \\ S(S_{\mu n x_1 n y_1) n x_2 n y_2} \\ \quad \vdots \\ S(\ldots S(S_{\mu n x_1 n y_1) n x_2 n y_2 \ldots) n x_n n y_n} \end{array}$$

where the last expression

$$S(\ldots S(S_{\mu n x_1 n y_1) n x_2 n y_2 \ldots) n x_n n y_n}$$

represents the final state of the variable space μ.

This expression corresponds to the effect of a sequence of assignment statements

$$x_1 := y_1$$

$$x_2 := y_2$$

$$\vdots$$

$$x_n := y_n$$

acting upon the variables $x_1,\ldots,x_n$, $y_1,\ldots,y_n$.

§4.C A Dynamic Property of the Variables

Consider the following pair of statements:

$$\begin{aligned} x &:= y \\ z &:= x \end{aligned} \tag{4.13}$$

From the last section, these are modelled by

$$S\mu\kappa x\kappa y$$

$$S\mu\kappa z\kappa x$$

which, as we have seen, leads to the expression

$$S(S\mu\kappa x\kappa y)\kappa z\kappa x \tag{4.14}$$

representing the sequence of assignments of (4.13).

The expression (4.14) is of the form

$$S(Sabc)db.$$

In previous treatments of S-expressions in Chapter II and III, all occurences of the same variable were regarded as exactly the same, ie. having the same value. Such is not the case in computations.

This is due to a conceptual difference between variables as used in mathematics and those used in a programming language. The difference being that in mathematics the variables have values which are fixed throughout any mathematical argument whereas the value of a variable in a program varies at different stages of its execution.

Thus in an expression arising from statements of a program such as (4.14) the two occurences of νx need not, in general, refer to the same value.

For example, suppose that the variable space μ just before the execution of (4.13) is

$$\mu = \{x,3,y,2,z,5\}.$$

Then, $\nu x = 3$, $\nu y = 2$, $\nu z = 5$ and

$$S\mu\nu x\nu y = S\{x,3,y,2,z,5\}\ \nu x\nu y$$
$$= \{x,2,y,2,z,5\}$$

ie. just after the execution of $S\mu\nu x\nu y$,

$$\mu = \{x,2,y,2,z,5\} \quad \text{giving} \quad \nu x = 2 .$$

Therefore, in the expression

$$S(S\mu\nu x\nu y)\nu z\nu x ,$$

the operand νx in the inner S-expression refers to the value 3, while the other occurence of νx refers to the value 2.

In order to differentiate between the two types of operand, the following notation will be employed:

Write $S(\ldots S(S\mu a_1 b_1)a_2 b_2 \ldots)a_n b_n$

as $S\mu[a_1 b_1, a_2 b_2, \ldots, a_n b_n]$

if the operands a_i, b_i have fixed values, and

as $S\mu[a_1 b_1; a_2 b_2; \ldots; a_n b_n]$

if the operands a_i, b_i do not have fixed values.

The first kind of notation will be called a static S-sequence and the second one a dynamic S-sequence.

Each pair $a_i b_i$ in the sequence is called a term of the sequence with a_i as the first component and b_i as the second component of the term.

From the above definition, any S-sequence obtained as the model of a set of assignment statements is a dynamic one.

Given a set of assignment statements, such a sequence can be obtained directly.

eg.
```
a := b
b := c
c := a
b := d
d := c
```

is equivalent to the dynamic S-sequence

$$S\mu[\iota a_\iota b;\ \iota b_\iota c;\ \iota c_\iota a;\ \iota b_\iota d;\ \iota d_\iota c]$$

where $\mu \in P^*(X)$ is the corresponding variable space.

§4.D Transformation of a Dynamic into a Static Sequence

Now algebraic manipulation can only be performed on an S-sequence of static type as shown in Chapter II and III. Therefore, a dynamic S-sequence obtained to model a sequence of assignments must be transformed into a corresponding static sequence before any reduction is attempted.

Consider the following sequences:

(a) $T_1 = S_\mu[{}_\kappa a_\eta b;\ {}_\kappa a_\eta c]$

(b) $T_2 = S_\mu[{}_\kappa a_\eta b;\ {}_\kappa c_\eta a]$.

In both cases, the operand ${}_\kappa a$ in the second term appears as the first component of the first term.

Let $\mu = \{a,\alpha,b,\beta,c,\gamma,\ldots\}$ be the variable space with $a,b,c \in N$, $\alpha,\beta,\gamma \in V$, then

$$S_{\mu\kappa}a_\eta b = S\{a,\alpha,b,\beta,c,\gamma,\ldots\}\ {}_\kappa a_\eta b$$
$$= \{a,\beta,b,\beta,c,\gamma,\ldots\}$$

So
$$T_1 = S\{a,\beta,b,\beta,c,\gamma,\ldots\}\ {}_\kappa a_\eta c \qquad (4.15)$$
$$= \{a,\gamma,b,\beta,c,\gamma,\ldots\}$$

and
$$T_2 = S\{a,\beta.b,\beta,c,\gamma,\ldots\}\ {}_\kappa c_\eta a \qquad (4.16)$$
$$= \{a,\beta,b,\beta,c,\beta,\ldots\} .$$

There is a difference between ${}_\kappa a$ in (4.15) and in (4.16)

In the first case, ${}_\kappa a$ merely refers to the element immediately following the name a . The actual value of that element makes no difference to the resulting set. ${}_\kappa a$ here indicates the position of the element to be replaced by ${}_\kappa c$.

In the second case, however, the situation is quite different. The contents of the final set very much depends on the actual value of κa . In this particular example, κa has changed from α to β as the result of the first step.

Thus, the term $\kappa a \kappa c$ in the dynamic sequence

$$S\mu[\ldots; \kappa a \kappa c; \ldots]$$

may be interpreted as:

"In μ , replace the element immediately following a by the current value of c "

Therefore, in the conversion of a dynamic S-sequence into the corresponding static sequence, it is the current value of each variable that has to be considered.

That is, in the transformation of sequences, only the second component of each term is subject to alteration.

The above observation leads to the following general rule of transformation:

Consider the sequence term by term.

If the second component, κa , of the jth term appears in the sequence as the first component, say as $\kappa a \kappa b$, in the ith term where $i < j$, then replace κa in the jth term by κb.

Otherwise, the term remains unchanged.

Note

Having considered the jth term for any necessary change, the separator ';' preceding this term is replaced by ',' to indicate how far the dynamic sequence has been partially transformed to static.

Example

The term-by-term transformation of

$$S\mu[\varkappa a\varkappa b;\ \varkappa c\varkappa d;\ \varkappa e\varkappa a;\ \varkappa a\varkappa c]$$

$$S\mu[\varkappa a\varkappa b;\ \varkappa c\varkappa d;\ \varkappa e\varkappa a;\ \varkappa a\varkappa c]$$
$$= S\mu[\varkappa a\varkappa b,\ \varkappa c\varkappa d;\ \varkappa e\varkappa a;\ \varkappa a\varkappa c]$$
$$= S\mu[\varkappa a\varkappa b,\ \varkappa c\varkappa d,\ \varkappa e\varkappa b;\ \varkappa a\varkappa c]$$
$$= S\mu[\varkappa a\varkappa b,\ \varkappa c\varkappa d,\ \varkappa e\varkappa b,\ \varkappa a\varkappa d]$$

The above rule of transformation, however, does not take into account cases in which the second component of a term appears as the first component in more than one earlier term.

For instance, in the sequence

$$S\mu[\varkappa a\varkappa b;\ \varkappa a\varkappa c;\ \varkappa d\varkappa a],$$

$\varkappa a$ appears as the first component of both the first and the second term.

What, then, is the change to be made on $\varkappa a$ in the third term?

With reference to the intuitive meaning of the terms $\varkappa a\varkappa b$ and $\varkappa a\varkappa c$, it is clear that the effect of the first term is overridden by that of the second term. Hence the transformation of the above expression yields

$$S\mu[\varkappa a\varkappa b,\ \varkappa a\varkappa c,\ \varkappa d\varkappa c].$$

Thus, the rule of transformation is:

Consider the sequence term by term.

If the second component, $_{\kappa}a$, of the jth term appears in the sequence as the first component in terms $i_1, i_2, \ldots, i_n$ where $i_1 < i_2 < \ldots < i_n < j$, then replace $_{\kappa}a$ by the second component of the i_nth term.

Otherwise, the term remains unchanged. (4.17)

Once a dynamic sequence which is a representation of a set of assignment statements has been transformed to static, it can be manipulated by the rules of algebra. In particular, it can be considered for transformation and reduction by the algebraic rules developed for S-expressions in Chapter III.

Apart from the rules mentioned above which apply to S-expressions with general set operands, there are further results which are particularly relevant to the case of our present interest, ie. the case in which the first operand of S is an element of $P^*(X)$ and the second and third operands are values. These results will be developed in the theorems of the next section.

§4.E Reduction of Static S-Sequences

A static S-sequence

$$S_{\mu}[{}_{\kappa}a_1\,{}_{\kappa}b_1, {}_{\kappa}a_2\,{}_{\kappa}b_2, \ldots, {}_{\kappa}a_n\,{}_{\kappa}b_n]$$

may be reduceable with the help of the following theorems:

Theorem 4.18

(i) $S_\mu[\varkappa a_\varkappa a] = \mu$

and

(ii) $S_\mu[\varkappa a_\varkappa b] = \mu$ iff $\varkappa a = \varkappa b$.

Proof

Obvious from the definition of S .

Theorem 4.19

$$S_\mu[\varkappa a_\varkappa b, \varkappa a_\varkappa c] = S_\mu[\varkappa a_\varkappa c]$$

except when $a \in \mu$, $b \in \mu$, $c \notin \mu$.

Proof

If either $a = b$ or $\varkappa a = \varkappa b$ then theorem 4.18 applies and the equality trivially holds.

So assume $\varkappa a \neq \varkappa b$.

In which case, the possibilities are:-

(i) $a \notin \mu$ $\Rightarrow$ LHS = RHS = μ

(ii) $a \in \mu$, $b \notin \mu$ $\Rightarrow$ LHS = RHS = $S_\mu[\varkappa a_\varkappa c]$

(iii) $a \in \mu$, $b \in \mu$, $c \notin \mu$ $\Rightarrow$ LHS = $S_\mu[\varkappa a_\varkappa b]$, RHS = μ

But $\varkappa a \neq \varkappa b$, so LHS $\neq$ RHS by theorem 4.18

(iv) $a,b,c \in \mu$, suppose

$$\mu = \{a,\alpha,b,\beta,c,\gamma,\ldots\}, \quad \text{then}$$

$$\text{LHS} = S_\mu[\varkappa a_\varkappa b, \varkappa a_\varkappa c]$$

$$= S\{a,\alpha,b,\beta,c,\gamma,\ldots\}[\varkappa a_\varkappa b, \varkappa a_\varkappa c]$$

$$= S\{a,\beta,b,\beta,c,\gamma,\ldots\}\ [\varkappa a_\varkappa c]$$

$$= \{a,\gamma,b,\beta,c,\gamma,\ldots\}$$

$$\text{RHS} = S_\mu[\varkappa a_\varkappa c]$$

$$= S\{a,\alpha,b,\beta,c,\gamma,\ldots\}\ [\varkappa a_\varkappa c]$$

$$= \{a,\gamma,b,\beta,c,\gamma,\ldots\}$$

$\therefore$ LHS = RHS by definition 4.5.

Theorem 4.20

Given $a,b,c,d \in \mu$, then

$$S_\mu[\varkappa a_\varkappa b,\ \varkappa c_\varkappa d] = S_\mu[\varkappa c_\varkappa d,\ \varkappa a_\varkappa b]$$

if either $a \neq c$ or $a=c$ and $\varkappa b \neq \varkappa d$.

Proof

By the same argument as in theorem 4.19, assume $a \neq b$, $c \neq d$, $\varkappa a \neq \varkappa b$ and $\varkappa c \neq \varkappa d$.

Let

$$\mu = \{a,\alpha,b,\beta,c,\gamma,d,\delta,\ldots\}$$

so $\varkappa a = \alpha$, $\varkappa b = \beta$, $\varkappa c = \gamma$, $\varkappa d = \delta$.

(i) Suppose $a = c$,

then $\varkappa b = \varkappa d$ leads to triviality

assume $\varkappa b \neq \varkappa d$

$\therefore$ LHS $= S_\mu[\varkappa a_\varkappa b, \varkappa a_\varkappa d]$

$= S_\mu[\varkappa a_\varkappa d]$ by theorem 4.19

$$RHS = S\mu[\kappa a_\kappa d, \kappa a_\kappa b]$$

$$= S\mu[\kappa a_\kappa b] \qquad \text{by theorem 4.19}$$

$\kappa b \neq \kappa d \Rightarrow LHS \neq RHS$.

Hence $LHS = RHS \Rightarrow a \neq c$.

(ii) Given $a \neq c$, so

$$LHS = S\mu[\kappa a_\kappa b, \kappa c_\kappa d]$$

$$RHS = S\mu[\kappa c_\kappa d, \kappa a_\kappa b] .$$

In LHS , α is replaced by κb

then γ is replaced by κd .

In RHS , γ is replaced by κd

then α is replaced by κb .

In both cases, the two elements α and γ are replaced by the same values. Even though the replacements are in reversed order, the final results are the same.

$\therefore$ $a \neq c \Rightarrow LHS = RHS$.

Note

Whether $a = d$ or not makes no difference to the result of this theorem. If $b = d$ then the theorem always holds.

The results proved in Theorem 4.18 to Theorem 4.20 can be used in the reduction of static S-sequences.

Example

Reduce $S_\mu[\kappa a \kappa b, \kappa b \kappa c, \kappa c \kappa b, \kappa a \kappa c]$

$S_\mu[\kappa a \kappa b, \kappa b \kappa c, \kappa c \kappa b, \kappa a \kappa c]$

$= S_\mu[\kappa a \kappa b, \kappa b \kappa c, \kappa a \kappa c, \kappa c \kappa b]$ by theorem 4.20

$= S_\mu[\kappa a \kappa b, \kappa a \kappa c, \kappa b \kappa c, \kappa c \kappa b]$ by theorem 4.20

$= S_\mu[\kappa a \kappa c, \kappa b \kappa c, \kappa c \kappa b]$ by theorem 4.19

Remark

From the theorems, it is clear that the minimal form of a static sequence is

$$S_\mu[\kappa a_1 \kappa b_1, \kappa a_2 \kappa b_2, \ldots, \kappa a_n \kappa b_n]$$

where the a_i's are all different and n is always less than or equal to the number of variables in μ.

Now theorem 4.20 implies that in a static sequence containing terms with distinct first components, the order of the terms is immaterial. Therefore in a sequence

$$S_\mu[\ldots,\ldots,\ldots\ldots]$$

with such a property, the terms in [...] can be regarded as the elements of a set.

The equivalence of minimal S-sequences can, thus, be defined from the equality of sets.

Definition 4.21

Two minimal static S-sequences

$$T_1 = S\mu[a_1b_1, a_2b_2, \ldots, a_nb_n]$$

$$T_2 = S\eta[x_1y_1, x_2y_2, \ldots, x_my_m]$$

are said to be equivalent, denoted $T_1 = T_2$, iff

(i) $\mu = \eta$ from definition 4.5

(ii) $\{a_1, a_2, \ldots, a_n\} = \{x_1, x_2, \ldots, x_m\}$

(iii) $a_i = x_j \Rightarrow b_i = y_j \quad \forall i,j$

eg. $S\mu[n x_n a, n y_n b, n z_n c] = S\mu[n z_n c, n x_n a, n y_n b]$

The conditions of equivalence in the above definition may be slightly relaxed to give the equivalence of minimal static sequences over a particular set of variables.

Definition 4.22

Two minimal static S-sequences

$$T_1 = S\mu[a_1b_1, a_2b_2, \ldots, a_nb_n]$$

$$T_2 = S\eta[x_1y_1, x_2y_2, \ldots, x_ny_n]$$

are said to be equivalent over the set $\{n_1, n_2, \ldots, n_l\}$ of names, written as

$$T_1 \underset{\{n_1, \ldots, n_l\}}{=} T_2$$

iff the following conditions hold:

(i) $\mu \cap \eta = \nu \neq \emptyset$

Let $\nu = \{n_1, v_1, n_2, v_2, \ldots, n_l, v_l\}$, $v_1, \ldots, v_l \in V$

(ii) If $\theta = \{\kappa n_1, \kappa n_2, \ldots, \kappa n_l\}$, then

$\theta \subseteq \{a_1, \ldots, a_n\}$, say $\theta = \{a_r, \ldots, a_s\}$

and $\theta \subseteq \{x_1, \ldots, x_m\}$, say $\theta = \{x_p, \ldots, x_q\}$

where $\{r, \ldots, s\} \subseteq \{1, \ldots, n\}$ and $\{p, \ldots, q\} \subseteq \{1, \ldots, m\}$

(iii) $a_i = x_j \Rightarrow b_i = y_j \quad \forall\, i \in \{r, \ldots, s\}$

$j \in \{p, \ldots, q\}$.

eg. If $T_1 = S\{a, \alpha, b, \beta, c, \gamma\}\ [\kappa a \kappa b, \kappa b \kappa a, \kappa c \kappa a]$

$T_2 = S\{a, \alpha, b, \beta\}\ [\kappa a \kappa b, \kappa b \kappa a]$

then $T_1 \underset{\{a,b\}}{=} T_2$

§4.F Equivalence of Sequences of Simple Assignment Statements

Given two sequences of assignment statements, it is possible to show, using the results of the previous section, whether or not they are equivalent.

Firstly, the concept of equivalence is defined.

Definition 4.23

If Θ and Φ are two sequences of assignment statements involving the variables $x_1, \ldots, x_n$, then Θ and Φ are said to be equivalent

$$\Theta \simeq \Phi$$

iff the final values of $x_1, \ldots, x_n$ are the same.

Θ and Φ are equivalent over the set $\{x_r,\ldots,x_s\} \subset \{x_1,\ldots,x_n\}$

$$\Theta \underset{\{x_r,\ldots,x_s\}}{\simeq} \Phi$$

iff the final values of $\{x_r,\ldots,x_s\}$ are the same for any valid initial values.

eg.

(i) If $x = 3$, $y = 5$, $z = 2$ as the result of both Θ and Φ, then

$$\Theta \simeq \Phi$$

(ii) If Θ yields

$$x = 3,\ y = 5,\ z = 2$$

and Φ yields

$$x = 3,\ y = 7,\ z = 2$$

then $\Theta \underset{\{x,z\}}{\simeq} \Phi$

To consider the equivalence of two sequences of assignment statements Θ and Φ, the following procedure is used:

1. Obtain the dynamic S-sequence corresponding to Θ and Φ [§4.C].
2. Transform the dynamic sequences into static sequences [§4.D].
3. Reduce the static sequences to their minimal forms by the application of theorems 4.18 to 4.20 [§4.E].
4. Compare the minimal sequences according to the definition of equivalences [Definitions 4.21 and 4.22].

In what follows we shall assume that all the programs are valid in the sense that a variable cannot appear as the right-hand side of an assignment statement unless it has been initialized before the execution of that statement.

Example

Consider the following sets of assignment statements:

Θ :	Φ :
d := b	d := a
b := a	a := b
a := d	b := c
d := b	c := a
b := c	a := d
c := a	
a := d	

1. Dynamic S-sequences corresponding to Θ and Φ are:-

$$T_\theta = S\mu[\kappa d\kappa b;\ \kappa b\kappa a;\ \kappa a\kappa d;\ \kappa d\kappa b;\ \kappa b\kappa c;\ \kappa c\kappa a;\ \kappa a\kappa d]$$

and $$T_\Phi = S\mu[\kappa d\kappa a;\ \kappa a\kappa b;\ \kappa b\kappa c;\ \kappa c\kappa a;\ \kappa a\kappa d]$$

2. Transform T_θ and T_Φ into static sequences

$$T_\theta = S\mu[\kappa d\kappa b,\kappa b\kappa a,\kappa a\kappa b,\kappa d\kappa a,\kappa b\kappa c,\kappa c\kappa b,\kappa a\kappa a]$$

$$T_\Phi = S\mu[\kappa d\kappa a,\kappa a\kappa b,\kappa b\kappa c,\kappa c\kappa b,\kappa a\kappa a]$$

3. Reduce the sequences to minimal forms

$$T_\theta = S\mu[\kappa d\kappa b,\kappa d\kappa a,\kappa b\kappa a,\kappa b\kappa c,\kappa a\kappa b,\kappa a\kappa a,\kappa c\kappa b]$$

by repeated application of theorem 4.20.

$$= S\mu[\kappa d\kappa a,\kappa b\kappa c,\kappa a\kappa a,\kappa c\kappa b]$$

by theorem 4.19.

$\therefore$ $T_\theta = S_\mu[\kappa d_\kappa a, \kappa b_\kappa c, \kappa c_\kappa b]$ by theorem 4.18

$T_\Phi = S_\mu[\kappa d_\kappa a, \kappa a_\kappa b, \kappa a_\kappa a, \kappa b_\kappa c, \kappa c_\kappa b]$ by theorem 4.20

$= S_\mu[\kappa d_\kappa a, \kappa a_\kappa a, \kappa b_\kappa c, \kappa c_\kappa b]$ by theorem 4.19

ie. $T_\Phi = S_\mu[\kappa d_\kappa a, \kappa b_\kappa c, \kappa c_\kappa b]$ by theorem 4.18

4. $T_\theta = T_\Phi$ by definition 4.21

Hence

$\Theta \simeq \Phi$

Note

The concept of equivalence given here is the same as that used by de Bakker [1] and Igarashi [16], both of whom adopted an axiomatic approach for their consideration of an equivalence theory of programs.

§4.G Assignment Statements with Expressions on the Right-Hand Sides

So far, only simple assignments of the form

x := y

have been considered.

In this section, we extend our consideration to include cases in which the right-hand side of an assignment statement involves operations upon the variables such as

x := y + z

In general, consider

$$x := y \ \underline{op} \ z \qquad (4.24)$$

and $x := \underline{op}\ y$

where $\underline{op}$ denotes an arithmetic or a boolean operator.

As before, statements (4.24) are modelled by

$$S_{\mu\ \kappa x \kappa}(y \ \underline{op} \ z)$$

and $S_{\mu\ \kappa x \kappa}(\underline{op}\ y)$

where μ is the variable space.

Now κx , where x is a single variable, has been defined in §4.A. In order to consider cases in which the operand of κ is an expression, we first define a new function $\kappa\ \underline{op}$.

Definition 4.25

The function $\kappa\ \underline{op} : V \times V \to V$ is such that if $v_1, v_2 \in V$ then

$$\kappa\ \underline{op}\ (v_1, v_2)$$

is the result of applying a process corresponding to the operation $\underline{op}$ to the values v_1 and v_2.

eg. If $a, b \in V$, then

$$\kappa+ \ (a,b)$$

gives the value obtained from a, b by the process corresponding to addition.

When $\underline{op}$ is a monadic operation, $\varkappa\ \underline{op}$ is similarly defined to be the function $\varkappa\ \underline{op} : V \to V$ such that

$$\varkappa\ \underline{op}(v) \quad \text{for} \quad v \in V$$

is the result of applying a process corresponding to the operation $\underline{op}$ to the value v.

Definition 4.26

For $x,y \in N$, $\varkappa(x\ \underline{op}\ y)$ is defined to be

$$\varkappa\ \underline{op}\ (\varkappa x,\varkappa y)$$

Similarly, in the case of monadic operator

$$\varkappa\ (\underline{op}\ x) = \varkappa\ \underline{op}\ (\varkappa x)$$

eg. $\varkappa(x \times y) = \varkappa\times(\varkappa x,\varkappa y)$

$\varkappa(-x) = \varkappa-(\varkappa x)$

$\varkappa(x \vee y) = \varkappa\vee(\varkappa x,\varkappa y)$

In what follows, the usual laws of arithmetic and boolean algebra will be assumed.

Thus, the function $\varkappa$ with more complicated forms of operand may be considered in terms of $\varkappa\ \underline{op}$.

eg. $\varkappa(y \times z - x) = \varkappa-(\varkappa\times(\varkappa y,\varkappa z),\varkappa x)$

$\varkappa(x \vee (\sim y \wedge z)) = \varkappa\vee(\varkappa x,\varkappa\wedge(\varkappa\sim(\varkappa y),\varkappa z))$

Suppose the dynamic sequence obtained from a program is of the form

$$T = S_u[\ldots;\ \varkappa n_i \varkappa n_r;\ \ldots;\ \varkappa n_s \varkappa(n_i\ \underline{op}\ n_j);\ \ldots]$$

where the subsequence between the two terms does not contain any term with $\varkappa n_i$ as the first component.

From definition 4.26, T can be written as

$$T = S\mu[\ldots; \varkappa n_i \varkappa n_r; \ldots; \varkappa n_s \varkappa \underline{op}\ (\varkappa n_i, \varkappa n_j); \ldots] .$$

To transform T into a static sequence, the transformation rule (4.17) applies. Suppose $\varkappa n_i$ appear as a first component for the first time in the term $\varkappa n_i \varkappa n_r$, then

$$T = S\mu[\ldots, \varkappa n_i \varkappa n_r, \ldots, \varkappa n_s \varkappa \underline{op}\ (\varkappa n_r, \varkappa n_j), \ldots] .$$

That is,

$$T = S\mu[\ldots, \varkappa n_i \varkappa n_r, \ldots, \varkappa n_s \varkappa (n_r \underline{op}\ n_j), \ldots]$$

by definition 4.26.

Hence, we have just proved the following:

Lemma 4.27

Let T be a dynamic sequence

$$T = S\mu[\ldots; \varkappa n_i \varkappa n_r; \ldots; \varkappa n_s \varkappa (n_i \underline{op}\ n_j); \ldots]$$

where $\varkappa n_i \varkappa n_r$ is the only term in the subsequence up to the term $\varkappa n_s \varkappa (n_i \underline{op}\ n_j)$ which contains $\varkappa n_i$ as the first component.

Then, the transformation of T into a static sequence gives

$$T = S\mu[\ldots, \varkappa n_i \varkappa n_r, \ldots, \varkappa n_s \varkappa (n_r \underline{op}\ n_j), \ldots]$$

The same argument applies where op is a monadic operator and we have

Lemma 4.28

Let T be a dynamic sequence

$$T = S_{\mu}[\ldots;\ {}_{\kappa}n_i{}_{\kappa}n_r;\ \ldots;\ {}_{\kappa}n_s{}_{\kappa}(\underline{op}\ n_i);\ \ldots]$$

where ${}_{\kappa}n_i{}_{\kappa}n_r$ is the only term in the subsequence up to the term ${}_{\kappa}n_s{}_{\kappa}(\underline{op}\ n_i)$ which contains ${}_{\kappa}n_i$ as the first component.

Then, the transformation of T into a static sequence gives

$$T = S_{\mu}[\ldots,{}_{\kappa}n_i{}_{\kappa}n_r,\ldots,{}_{\kappa}n_s{}_{\kappa}(\underline{op}\ n_r),\ \ldots]$$

Thus, a dynamic S-sequence obtained from a program involving operations on the variables can be simplified as in the case of simple assignments.

For this purpose, the procedure of §4.F is slightly extended as follows:

1. Obtain the dynamic S-sequence corresponding to the given set of assignments.

2. Transform the dynamic into static sequence using the transformation rule 4.17 and lemmas 4.27 and 4.28.

3. Perform all possible simplification of the expressions in the second components.

4. Reduce the sequence to its minimal form.

Example 4.29

Reduction of the sequence

a := a+b

b := a-b

a := a-b

1. The dynamic sequence corresponding to the above program is

$$T = S_\mu[{}_\kappa a_\kappa(a+b);\ {}_\kappa b_\kappa(a-b);\ {}_\kappa a_\kappa(a-b)]$$

2. Transform T into a static sequence

$$T = S_\mu[{}_\kappa a_\kappa(a+b), {}_\kappa b_\kappa(a+b-b), {}_\kappa a_\kappa(a+b-a-b+b)]$$

3. Simplify the expressions in the second components

$$T = S_\mu[{}_\kappa a_\kappa(a+b), {}_\kappa b_\kappa a, {}_\kappa a_\kappa b]$$

4. Reduce T to its minimal form

$$T = S_\mu[{}_\kappa a_\kappa b, {}_\kappa b_\kappa a] \ .$$

§4.H Transformation of a Static into a Dynamic Sequence

We have shown that, given a dynamic S-sequence corresponding to a program, it can be transformed into a static sequence and then reduced to its minimal form.

The minimal static sequence obtained gives the final values of the variables involved and may, therefore, be regarded as a set of parallel assignments.

For instance, a minimal sequence such as

$$S_{\mu}[{}_{\mathcal{H}}b_{\mathcal{H}}d,{}_{\mathcal{H}}a_{\mathcal{H}}c,{}_{\mathcal{H}}d_{\mathcal{H}}a,{}_{\mathcal{H}}c_{\mathcal{H}}a]$$

may be written as a multiple assignment statement

```
b,a,d,c := d,c,a,a
```

in a programming language which allows parallel assignments such as CPL [2].

In a sequential programming language, however, the sequence of assignments

```
b := d
a := c
d := a
c := a
```

obtained directly from the minimal S-sequence does not yield the final values as required.

Thus, in order to obtain a set of assignments corresponding to a given static sequence, the sequence must be transformed into a dynamic sequence which is a direct representation of a valid program.

There are three main points to be considered in the conversion:-

1. The order of assignments
2. The current values of the variables involved
3. The information lost during the reduction process.

We shall now consider the above problems in detail.

P1. The order of assignments

It is clear that, in a sequential programming langauge, the order of assignments plays a very important part. Thus, in the transformation of a static into a dynamic sequence, attention must be paid to the order of the terms in the dynamic sequence.

The first question is, given a minimal static sequence, how to choose the first term of what is to be the corresponding dynamic sequence.

The solution lies in the following observation:

In a program, the first assignment statement is such that either

(a) the expression on the RHS of the assignment does not contain the variable on the LHS

eg. x := y , a := b+c

or

(b) the expression on the RHS of the assignment contains the variable on the LHS

eg. x := x+y

If a program contains the first statement of type (a), then the value of the variable on the LHS of the first statement cannot appear in the final stage, ie. from the above examples $_{\kappa}x$, $_{\kappa}a$ cannot appear as a second component in the minimal static sequences.

A program with the first statement of type (b), however, may retain the value of the variable on the LHS of its first assignment in the final stage.

Thus, the procedure for selecting the first term of the dynamic sequence from a given static sequence

$$T = S_{\mu}[\varkappa x_1 \varkappa y_1, \varkappa x_2 \varkappa y_2, \cdots, \varkappa x_n \varkappa y_n]$$

is as follows:

P1.1 If $\exists$ a term $\varkappa x_i \varkappa y_i$ such that $\varkappa x_i$ does not appear in the second component of a term in T, then choose $\varkappa x_i \varkappa y_i$ as the first term of the dynamic sequence.

P1.2 Otherwise, choose the first term to be $\varkappa x_i \varkappa y_i$ such that y_i is a function of x_i.

eg. From the minimal static sequence

(i) $S_{\mu}[\varkappa a_{\varkappa} b, \varkappa b_{\varkappa} a, \varkappa c_{\varkappa} a]$, the first term is chosen to be $\varkappa c_{\varkappa} a$.

(ii) $S_{\mu}[\varkappa a_{\varkappa} c, \varkappa b_{\varkappa} (b+c), \varkappa c_{\varkappa} a]$, the first term is $\varkappa b_{\varkappa} (b+c)$.

(iii) $S_{\mu}[\varkappa a_{\varkappa} (a+b), \varkappa b_{\varkappa} (c+a), \varkappa c_{\varkappa} b, \varkappa d_{\varkappa} c]$, the first term is $\varkappa d_{\varkappa} c$.

If a minimal static sequence under consideration contains more then one term with such properties, then a term with the shortest second component is chosen on the grounds that it normally leads to a simpler dynamic sequence.

Note

The case in which neither P1.1 nor P1.2 holds will be treated later [P3].

P2. The current values of the variables

Suppose the first term of the dynamic sequence corresponding to T is chosen to be $\varkappa x_i \varkappa y_i$.

Such a dynamic sequence is a representation of a set of assignments starting with the statement

$$x_i := y_i \ .$$

Therefore, from now on, whenever the value νy_i is required it is accessible through the variable x_i. Thus, a change is made on the rest of the terms in the static sequence to adjust the current values of the variables as follows:

P2.1

If $\nu x_i \nu y_i$ is the first term, then whenever νy_i appears as a second component in the remaining terms, it is replaced by νx_i.

eg. $S\mu[\nu a \nu b, \nu b \nu a, \nu c \nu a]$

with $\nu c \nu a$ as the first term becomes

$$S\mu[\nu c \nu a;\ \nu a \nu b, \nu b \nu c] \ .$$

Note that this process is the reverse of the transformation (4.17).

For cases in which the second components are expressions, the updating of the current values is performed by the application of the following lemmas:

Lemma 4.30

If $\nu n_i \nu n_j$ is chosen to be the first term of the dynamic sequence corresponding to

$$T = S\mu[\ldots, \nu n_i \nu n_j, \ldots, \nu n_r \nu (n_j \ \underline{op} \ n_s), \ldots] \ ,$$

then the updated sequence is

$$T = S_\mu[\varkappa n_i \varkappa n_j;\ \ldots, \varkappa n_r \varkappa (n_i \underline{\text{op}}\ n_s), \ldots] \ .$$

Proof

$$T = S_\mu[\ldots, \varkappa n_i \varkappa n_j, \ldots, \varkappa n_r \varkappa (n_j \underline{\text{op}}\ n_s), \ldots]$$

$$= S_\mu[\ldots, \varkappa n_i \varkappa n_j, \ldots, \varkappa n_r \varkappa \ \underline{\text{op}}\ (\varkappa n_j, \varkappa n_s), \ldots] \quad \text{by definition 4.26.}$$

With $\varkappa n_i \varkappa n_j$ as the first term,

$$T = S_\mu[\varkappa n_i \varkappa n_j;\ \ldots, \varkappa n_r \varkappa \ \underline{\text{op}}\ (\varkappa n_i, \varkappa n_s), \ldots] \qquad \text{by P2.1}$$

$$= S_\mu[\varkappa n_i \varkappa n_j;\ \ldots, \varkappa n_r \varkappa (n_i \underline{\text{op}}\ n_s), \ldots] \ .$$

Lemma 4.31

If $\varkappa n_i \varkappa n_j$ is chosen to be the first term of the dynamic sequence corresponding to

$$T = S_\mu[\ldots, \varkappa n_i \varkappa n_j, \ldots, \varkappa n_r \varkappa (\underline{\text{op}}\ n_j), \ldots] \ ,$$

then the updated sequence is

$$T = S_\mu[\varkappa n_i \varkappa n_j;\ \ldots, \varkappa n_r \varkappa (\underline{\text{op}}\ n_i), \ldots] \ .$$

Proof as in the previous lemma.

eg. $S_\mu[\varkappa b \varkappa c, \varkappa a \varkappa (a+b), \varkappa c \varkappa (b+a+c)]$

with $\varkappa a \varkappa (a+b)$ as the first term is updated to

$$S_\mu[\varkappa a \varkappa (a+b);\ \varkappa b \varkappa c, \varkappa c \varkappa (a+c)] \ .$$

Now, from the given static sequence

$$T = S_\mu[\varkappa x_1 \varkappa y_1, \varkappa x_2, \varkappa y_2, \ldots, \varkappa x_n \varkappa y_n] \ ,$$

the updating of T with respect to the first term $\kappa x_i \kappa y_i$ yields the sequence

$$T = S_U[\kappa x_i \kappa y_i;\ \kappa x_1 \kappa y_1', \kappa x_2 \kappa y_2', \ldots, \kappa x_n \kappa y_n']$$

where $y_1', y_2', \ldots, y_n'$ denote the updated second components. Note that $y_1', \ldots, y_n'$ do not contain y_i, but y_i may be among $x_1, \ldots, x_n$.

The above sequence could be written as

$$\begin{aligned} T &= S(S_\mu \kappa x_i \kappa y_i)[\kappa x_1 \kappa y_1', \ldots, \kappa x_n \kappa y_n'] \\ &= S_{\mu'}[\kappa x_1 \kappa y_1', \ldots, \kappa x_n \kappa y_n'] . \end{aligned}$$

Thus, the remaining n-1 terms $[\kappa x_1 \kappa y_1', \ldots, \kappa x_n \kappa y_r']$ can now be regarded as a minimal static sequence on its own. This implies that the second term of the dynamic sequence can be selected and the subsequence updated by the method used for the first term. This process is repeated until the terms in the original static sequence are exhausted and hence the required dynamic sequence is obtained.

eg.
$$\begin{aligned} & S_\mu[\kappa d \kappa c;\ \kappa a \kappa d, \kappa c \kappa (d-c), \kappa b \kappa a] \\ &= S_\mu[\kappa b \kappa a;\ \kappa d \kappa c, \kappa a \kappa d, \kappa c \kappa (d-c)] \\ &= S_\mu[\kappa b \kappa a;\ \kappa a \kappa d;\ \kappa d \kappa c, \kappa c \kappa (a-c)] \\ &= S_\mu[\kappa b \kappa a;\ \kappa a \kappa d;\ \kappa d \kappa c;\ \kappa c \kappa (a-d)] \end{aligned}$$

Hence, a dynamic sequence can be derived from any given static sequence. That the sequence obtained does correspond to the given one may be checked by applying the transformation (4.17) to the final result to obtain the original static sequence.

P3 The information lost during the reduction process

In P1 a procedure is given by which a next term in the dynamic sequence is selected from a given minimal sequence. Two alternatives P1.1 and P1.2 are considered. It is, however, possible that neither of the cases holds.

For example, the minimal sequence might be

$$T = S_{\mu}[{}_{\mu}a_{\mu}b, {}_{\mu}b_{\mu}a]$$

which could have emerged from the program

```
a := a+b
b := a-b                                   (4.32)
a := a-b
```

with the variable space containing only a and b . (Example 4.29)

Due to the information lost during the reduction process, a program such as (4.32) cannot be derived from the given sequence T by any process.

In such cases, we introduce more variables into the program as follows:

P3.1

If neither P.1.1 nor P.1.2 holds, then introduce a new variable $x_j \notin \mu$ into the sequence. The variable space is augmented to become $\mu \cup \{x_j, \omega)$ and the first term is chosen to be ${}_{\mu}x_j{}_{\mu}x_i$ where $x_i \in \{x_1, \ldots, x_n\}$.

The choice of x_i among the $x_1, \ldots, x_n$ could be made, strictly speaking, at random. But for the same reason as before, x_i is chosen such that the term ${}_{\mu}x_i{}_{\mu}y_i$ has the shortest second component.

eg. From $T = S\mu[{}_{\kappa}a_{\kappa}b, {}_{\kappa}b_{\kappa}a]$, a new variable c is introduced into the sequence - and the first term of the dynamic sequence is ${}_{\kappa}c_{\kappa}a$.

The variable space μ is augmented by $\{c,\omega\}$.

Let $\mu' = \mu \cup \{c,\omega\}$

The sequence obtained is, then

$$T_1 = S\mu'[{}_{\kappa}c_{\kappa}a;\ {}_{\kappa}a_{\kappa}b, {}_{\kappa}b_{\kappa}c]$$
$$= S\mu'[{}_{\kappa}c_{\kappa}a;\ {}_{\kappa}a_{\kappa}b;\ {}_{\kappa}b_{\kappa}c] \ .$$

Note that the dynamic sequence T_1 obtained after augmentation is equivalent to the given sequence T over the original set of variables

ie. $T_{1\{a,b\}} = T$.

Now consider the adjustment for the current values of variables discussed in P2.

If no structural information is lost during the reduction process, each replacement of a second component ${}_{\kappa}y_i$ by its corresponding first component ${}_{\kappa}x_i$ should yield a sequence with the following properties:

(a) ${}_{\kappa}y_i$ does not appear in any of the remaining second components.

(b) ${}_{\kappa}x_i$ appears in a second component only as a replacement of ${}_{\kappa}y_i$.

Property (a) always holds as it is a direct consequence of the adjustment. It is, however, possible that the updated sequence does not have the second property.

eg. The sequence

$$S_\mu[{}_\kappa c_\kappa b, {}_\kappa a_\kappa (a+c), {}_\kappa b_\kappa a]$$

with ${}_\kappa a_\kappa (a+c)$ chosen as its first term [P1.2] is updated to

$$S_\mu[{}_\kappa a_\kappa (a+c);\ {}_\kappa c_\kappa b, {}_\kappa b_\kappa a] \ .$$

Here the value of (a+c) has been assigned to the variable a and yet ${}_\kappa a$ appears in one of the remaining second components.

This is caused by the loss of information during the reduction. It indicates that the value of a has to be stored elsewhere before the assignment

$$a := a+c$$

is made.

Thus, a new variable is augmented to the sequence, the value of a is assigned to it and the sequence is updated accordingly.

Note that the violation of (b) could only happen when the next term is to be ${}_\kappa x_i {}_\kappa y_i$ where y_i is a function of x_i. Hence, before such a term is selected, a check must be made as to whether an augmentation is necessary.

This implies that P1.2 of the selection procedure has to be extended as follows:

P3.2

Else, if $\exists$ a term $\kappa x_i \kappa y_i$ such that y_i is a function of x_i then

(i) If x_i appears in the remaining second components, then augment the variable space by $\{x_j, \omega\}$ where $x_j \notin \mu$ and choose the first term to be $\kappa x_j \kappa x_i$.

(ii) Otherwise choose $\kappa x_i \kappa y_i$ as the first term.

eg. From the sequence

$$T = S_\mu[\kappa c \kappa b, \kappa a \kappa (a+c), \kappa b \kappa a] ,$$

introduce a new variable, say d .

The augmented variable space is $\mu' = \mu \cup \{d, \omega\}$ and the resulting sequence T_1 where

$$T_1 = {}^{T}_{\{a,b,c\}}$$

is such that

$$\begin{aligned} T_1 &= S_{\mu'}[\kappa d \kappa a;\ \kappa c \kappa b, \kappa a \kappa (d+c), \kappa b \kappa d] \\ &= S_{\mu'}[\kappa d \kappa a;\ \kappa a \kappa (d+c);\ \kappa c \kappa b, \kappa b \kappa d] \\ &= S_{\mu'}[\kappa d \kappa a;\ \kappa a \kappa (d+c);\ \kappa c \kappa b;\ \kappa b \kappa d] . \end{aligned}$$

Remark

In this section, more variables are added to the original variable space μ because of the structural information lost during the reduction process. The number of extra variables can be kept to the minimum by checking for

(i) a variable originally in μ which retains its own value in the final stage

(ii) a variable already introduced which is not going to be required later

and make use of such a variable whenever possible.

Summary

The transformation of an n-term minimal static sequence

$$S_\mu[\kappa x_1 \kappa y_1, \kappa x_2 \kappa y_2, \cdots, \kappa x_n \kappa y_n]$$

into a dynamic sequence may be summarised in the following algorithm

1. The number of terms in the static sequence is n.
   Let l be the number of terms in the dynamic sequence.
   t_l denotes the lth term of the dynamic sequence.
   At the beginning of the transformation, $l = 0$.

2. $l = l+1$.

3. Consider the n-term static sequence.
   If $\exists$ at least one term $\kappa x_i \kappa y_i$ such that κx_i does not appear in any of the second component then choose the one with the shortest second component and let $t_l = \kappa x_i \kappa y_i$.
   Goto 6.

4. If $\exists$ at least one $\kappa x_i \kappa y_i$ such that y_i is a function of x_i then select the one with the shortest second component.

Check whether x_i appears among the second components of the remaining n-1 terms.

If so, introduce $x_j \notin \mu$,

$$\mu = \mu \cup \{x_j, \omega\}$$

$$t_l = \varkappa x_j \varkappa x_i$$

and goto 7.

Otherwise

$$t_l = \varkappa x_i \varkappa y_i$$

and goto 6.

5. Introduce $x_j \notin \mu$.

$$\mu = \mu \cup \{x_j, \omega\}$$

$$t_l = \varkappa x_j \varkappa x_i$$

where $x_i \in \{x_1, \ldots, x_n\}$ is such that $\varkappa x_i \varkappa y_i$ has the shortest second component. Goto 7.

6. Remove t_l from the static sequence,

$$n = n-1$$

7. If $t_l = \varkappa x_m \varkappa y_m$, then replace all y_m in the second components of the n-term static sequence by x_m , assuming the usual laws of algebras when y_m is an expression.

8. If n=1 , then l = l+1

t_l= the last term of the static sequence

and stop

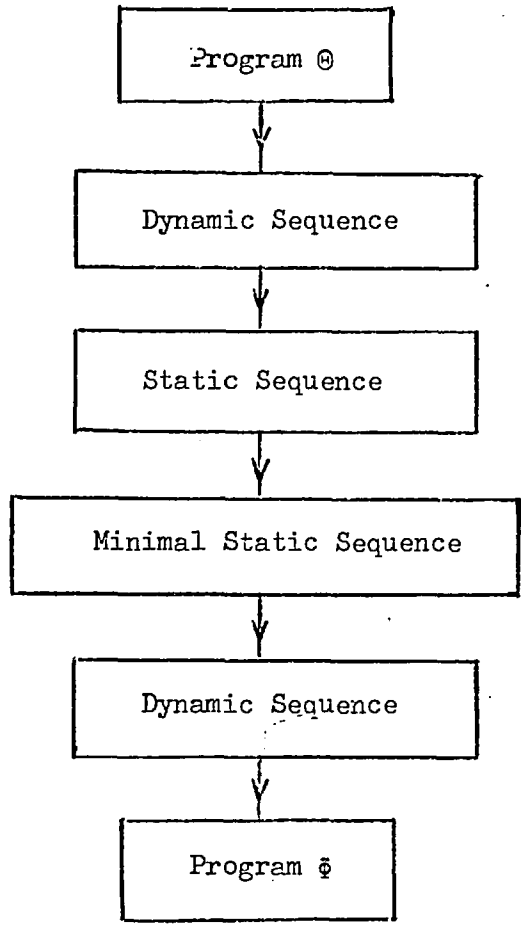
else goto 2.

§4.I The Simplification of Programs

In the last section, an algorithm for transforming a minimal static sequence into a dynamic sequence is developed.

This implies that it is possible to construct a valid program which would produce the specified final values for a given set of variables.

Moreover, given a simple program consisting of assignment statements, if there exists an equivalent program which is simpler then it can be obtained. The process of obtaining such a program is represented diagrammatically below:



Program Φ thus derived is a simplification of program Θ in the sense that it contains fewer statements and the assignments are less complicated.

Example

Let the initial variable space μ be

$$\mu = \{a,\alpha,b,\beta,c,\gamma,d,\delta\} .$$

and Θ consist of the following statements:

Θ : d := a+b+c

c := b+c

a := c-b

b := d-c

c := c-a

d := d-a

The dynamic sequence corresponding to Θ is

$$T = S_\mu[{}_\kappa d_\kappa(a+b+c);\ {}_\kappa c_\kappa(b+c);\ {}_\kappa a_\kappa(c-b);\ {}_\kappa b_\kappa(d-c);\ {}_\kappa c_\kappa(c-a);\ {}_\kappa d_\kappa(d-a)]$$

which is transformed into a static sequence and then reduced to its minimal form as follows:

$$T = S_\mu[{}_\kappa d_\kappa(a+b+c),{}_\kappa c_\kappa(b+c),{}_\kappa a_\kappa(b+c-b),{}_\kappa b_\kappa(a+b+c-b-c),{}_\kappa c_\kappa(b+c-b-c+b),$$
$${}_\kappa d_\kappa(a+b+c-b-c+b)]$$

$$= S_\mu[{}_\kappa d_\kappa(a+b+c),{}_\kappa c_\kappa(b+c),{}_\kappa a_\kappa c,{}_\kappa b_\kappa a,{}_\kappa c_\kappa b,{}_\kappa d_\kappa(a\div b)]$$

$$= S_\mu[{}_\kappa d_\kappa(a\div b),{}_\kappa c_\kappa b,{}_\kappa a_\kappa c,{}_\kappa b_\kappa a]$$

The transformation of T into a dynamic sequence gives

$$T = S_{\mu}[{}_{\kappa}d_{\kappa}(a+b);\ {}_{\kappa}c_{\kappa}b, {}_{\kappa}a_{\kappa}c, {}_{\kappa}b_{\kappa}a] \ .$$

Now let A be the set $\{a,b,c,d\}$, then

$$T = \underset{A}{T_1} = S_{\mu_1}[{}_{\kappa}d_{\kappa}(a+b);\ {}_{\kappa}f_{\kappa}c;\ {}_{\kappa}c_{\kappa}b, {}_{\kappa}a_{\kappa}f, {}_{\kappa}b_{\kappa}a]$$

where $\mu_1 = \mu \cup \{f,\omega\} = \{a,\alpha,b,\beta,c,\gamma,d,\delta,f,\omega\}$.

So $T_1 = S_{\mu_1}[{}_{\kappa}d_{\kappa}(a+b);\ {}_{\kappa}f_{\kappa}c;\ {}_{\kappa}c_{\kappa}b;\ {}_{\kappa}a_{\kappa}f, {}_{\kappa}b_{\kappa}a]$

$= S_{\mu_1}[{}_{\kappa}d_{\kappa}(a+b);\ {}_{\kappa}f_{\kappa}c;\ {}_{\kappa}c_{\kappa}b;\ {}_{\kappa}b_{\kappa}a;\ {}_{\kappa}a_{\kappa}f]$.

The sequence T_1 represents program Φ which is

Φ : d := a+b

f := c

c := b

b := a

a := f

Hence $\Phi \underset{\{a,b,c,d\}}{\simeq} \Theta$

CHAPTER 5

MODEL OF
A CONDITIONAL STATEMENT

§5.A Model of a Conditional Statement

Any conditional statement of the form

$$\underline{\text{if}}\ p\ \underline{\text{then}}\ \theta\ \underline{\text{else}}\ \varphi \tag{5.1}$$

where p is a propositional expression taking the value T or F

and θ , φ are unconditional statements

may be modelled by the S-expression

$$S\ (STp\theta)\ T\varphi$$

since $p = T \Rightarrow S\ (STp\theta)\ T\varphi = S\ (STT\theta)\ T\varphi$

$$= S\theta T\varphi$$

$$= \theta \qquad \because\ \theta \neq T$$

$$p = F \Rightarrow S\ (STp\theta)\ T\varphi = S\ (STF\theta)\ T\varphi$$

$$= STT\varphi$$

$$= \varphi$$

Note that the above expression corresponds to the conditional statement (5.1) under the assumption that θ is an unconditional statement and is, therefore, not a boolean value.

If θ is allowed to have a boolean value, then the S-expression

$$S\ (STp\theta)\ T\ (SpF\varphi) \tag{5.2}$$

will give the desired effect for

$$p = T \Rightarrow S\ (STp\theta)\ T\ (SpF\varphi) = S\ (STT\theta)\ T\ (STF\varphi)$$

$$= S\theta TT$$

$$= \theta$$

whilst

$$p = F \Rightarrow S\ (STp\theta)\ T\ (SpF\varphi) = S\ (STF\theta)\ T\ (SFF\varphi)$$
$$= STT\varphi$$
$$= \varphi$$

By allowing φ to be a dummy statement, denoted by Λ, an *if* statement of the form

if p *then* θ

may be represented by the expression

$$S\ (STp\theta)\ T\Lambda\ .$$

For convenience, we shall write the S-expression representing a conditional statement in a sequence form eg. we write

$S\ (STp\theta)\ T\varphi$ as $ST\ \{\ p\theta : T\varphi\ \}$

and $S\ (STp\theta)\ T\Lambda$ as $ST\ \{\ p\theta : T\Lambda\ \}$

Such a sequence is called a *conditional S-sequence* and has the general form

$$ST\ \{\ p_1\theta_1 : p_2\theta_2 : \cdots : p_n\theta_n\ \}$$

The above sequence, is a shorthand notation for the S-expression

$$S\ (\ \cdots\ S\ (STp_1\theta_1)\ p_2\theta_2\ \cdots\ )\ p_n\theta_n\ .$$

Obviously such a sequence will deliver the value of the θ corresponding to the first p that has the value T .

It is thus an analogue of the ALGOL-60 expression

$$\underline{\text{if}}\ p_1\ \underline{\text{then}}\ \theta_1\ \underline{\text{else if}}\ p_2\ \underline{\text{then}}\ \theta_2 \cdots \underline{\text{else if}}\ p_n\ \underline{\text{then}}\ \theta_n$$

and of McCarthy's conditional form [29]

$$(p \rightarrow \theta_1,\ p_2 \rightarrow \theta_2,\ \cdots,\ p_n \rightarrow \theta_n)$$

§5.B Formal Properties of Conditional S-sequences

Consider the nested S-expression

$$S\ (\ \cdots\ S.(STp_1\theta_1)\ p_2\theta_2\ \cdots\ )\ p_n\theta_n\ .$$

Here, with the exception of $STp_1\theta_1$, the first operand for each S-expression is the result of evaluating the S-expression at its next inner level of nesting.

The expression $STp_1\theta_1$ yields the value θ_1 or T according as $p_1 = T$ or F.

Thus, the first operand of each S-expression may be either the boolean value T or an unconditional statement represented by θ_i.

Under such circumstances, we are clearly dealing with a special class of S-expressions ie. with expressions $Sabc$ in which

$$b \in \{T,F\}$$

$$c \in E \quad \text{such that} \quad E \cap \{T,F\} = \emptyset$$

and $$a \in E \cup \{T\}\ .$$

To emphasise this, different sets of letters will be used for the operands of S ie.

$a, b, c, \ldots$	$\in E \cup \{T\}$	for the first operands
$p, q, r, \ldots$	$\in \{T,F\}$	for the second operands
$\theta, \varphi, \psi, \ldots$	$\in E$	for the third operands.

Hence, a general conditional statement corresponds to a nested S-expression of the form

$$S\ \{\ \ldots\ S\ (Sap_1\theta_1)\ p_2\theta_2\ \ \ldots\ \ )\ p_n\theta_n$$

written as a conditional S-sequence

$$Sa\ \{p_1\theta_1\ :\ p_2\theta_2\ :\ \ldots\ :\ p_n\theta_n\}\ .$$

Such sequences have properties which lead to the following theorems:

Theorem 5.3

$$Sa\ \{\ T\theta\ :\ p\varphi\ \}\ =\ Sa\ \{T\theta\}$$

Proof

$$a \in E \Rightarrow \text{LHS} = S\ (SaT\theta)\ p\varphi = Sap\varphi = a$$
$$\text{RHS} = SaT\theta = a$$
$$a = T \Rightarrow \text{LHS} = S\ (STT\theta)\ p\varphi = S\theta p\varphi = \theta$$
$$\text{RHS} = STT\theta = \theta$$

Theorem 5.4

$$Sa\ \{F\theta\}\ =\ a$$

Proof

Obvious since $a \in E \cup \{T\}$.

Theorem 5.5

$$Sa\ \{p\theta : T\theta\} = Sa\ \{T\theta\}$$

Proof

$a \in E \Rightarrow$ LHS $= S\ (Sap\theta)\ T\theta = SaT\theta$

$= RHS$

$a = T \Rightarrow$ LHS $= S\ (STp\theta)\ T\theta = \theta$ by R3 [Chapter II]

RHS $= STT\theta$

$= \theta$

Theorem 5.6

$$Sa\ \{\sim p\theta : T\varphi\} = Sa\ \{p\varphi : T\theta\}$$

Proof

$p = T \Rightarrow$ LHS $= Sa\ \{F\theta : T\varphi\} = Sa\ \{T\varphi\}$ by theorem 5.4

RHS $= Sa\ \{T\varphi : T\theta\} = Sa\ \{T\varphi\}$ by theorem 5.3

$p = F \Rightarrow$ LHS $= Sa\ \{T\theta : T\varphi\} = Sa\ \{T\theta\}$ by theorem 5.3

RHS $= Sa\ \{F\varphi : T\theta\} = Sa\ \{T\theta\}$ by theorem 5.4

Theorem 5.7

$$Sa\ \{p\theta : p\varphi\} = Sa\ \{p\theta\}$$

Proof

$p = T \Rightarrow$ LHS $= Sa\ \{T\theta : p\varphi\} = Sa\ \{T\theta\}$ by theorem 5.3

RHS $= Sa\ \{T\theta\}$

$p = F \Rightarrow$ LHS $= Sa\ \{F\theta : F\varphi\} = a$ by theorem 5.4

RHS $= Sa\ \{F\theta\} = a$ by theorem 5.4

Theorem 5.8

$$Sa\ \{p\theta : q\theta\} = Sa\ \{(p \vee q)\theta\}$$

Proof

$p = T \Rightarrow$ LHS $= Sa\ \{T\theta : q\theta\} = Sa\ \{T\theta\}$ by theorem 5.3

RHS $= Sa\ \{(T \vee q)\ \theta\} = Sa\ \{T\theta\}$

$p = F \Rightarrow$ LHS $= Sa\ \{F\theta : q\theta\} = Sa\ \{q\theta\}$ by theorem 5.4

RHS $= Sa\ \{(F \vee q)\ \theta\} = Sa\ \{q\theta\}$

Corollary

$$Sa\ \{p\theta : q\theta\} = Sa\ \{q\theta : p\theta\}$$

Proof

Follows directly from the above theorem since $(p \vee q) = (q \vee p)$.

Theorem 5.9

$$Sa\ \{p\theta : \sim p\varphi\} = Sa\ \{p\theta : T\varphi\}$$

Proof

RHS $= Sa\ \{p\theta : (p \vee \sim p)\ \varphi\}$

$= Sa\ \{p\theta : p\varphi : \sim p\varphi\}$ by theorem 5.8

$= Sa\ \{p\theta : \sim p\varphi\}$ by theorem 5.7

$=$ LHS

Theorem 5.10

$$Sa\ \{(p \wedge q)\theta : p\rho : r\theta : T\omega\} = Sa\ \{(p \wedge q) \vee (\sim p \wedge r)\theta : T\omega\}$$

Proof

$$\begin{aligned}
RHS &= Sa\ \{(p\wedge q)\ \theta : (\sim p\wedge r)\theta : T\varphi\} && \text{by theorem 5.8}\\
&= Sa\ \{(p\wedge q)\ \theta : \sim(\sim p\wedge r)\varphi : T\theta\} && \text{by theorem 5.6}\\
&= Sa\ \{(p\wedge q)\ \theta : (p\vee\sim r)\varphi : T\theta\}\\
&= Sa\ \{(p\wedge q)\ \theta : p\varphi : \sim r\varphi : T\theta\} && \text{by theorem 5.8}\\
&= Sa\ \{(p\wedge q)\ \theta : p\varphi : r\theta : T\varphi\} && \text{by theorem 5.6}\\
&= LHS
\end{aligned}$$

Theorem 5.11

$$\begin{aligned}
Sa\ \{(p\wedge q)\ \theta : p\theta : q\varphi\} &= Sa\ \{(p\wedge q)\ \theta : q\varphi : p\theta\}\\
&= Sa\ \{p\theta : q\varphi\}
\end{aligned}$$

Proof

$$p = q \Rightarrow Sa\ \{(p\wedge q)\ \theta : p\theta : q\varphi\} = Sa\ \{(p\wedge p)\ \theta : p\theta : p\varphi\}$$

$$= Sa\ \{p\theta\}$$

by theorem 5.7

$$Sa\ \{(p\wedge q)\ \theta : q\varphi : p\theta\} = Sa\ \{(p\wedge p)\ \theta : p\varphi : p\theta\}$$

$$= Sa\ \{p\theta\}$$

by theorem 5.7

$$\text{and}\quad Sa\ \{p\theta : q\varphi\} = Sa\ \{p\theta : p\varphi\}$$

$$= Sa\ \{p\theta\}$$

by theorem 5.7

$p \neq q$ ie. $p = \sim q$ and $q = \sim p$

$$\Rightarrow Sa\ \{(p\wedge q)\ \theta : p\theta : q\varphi\} = Sa\ \{(p\wedge\sim p)\ \theta : p\theta : \sim p\varphi\}$$

$$= Sa\ \{F\theta : p\theta : \sim p\varphi\}$$

$$= Sa\ \{p\theta : T\varphi\}$$

by theorems 5.4 and 5.9

$$Sa\ \{(p\wedge q)\ \theta : q\varphi : p\theta\} = Sa\ \{(p\wedge\sim p)\ \theta : q\varphi : \sim q\theta\}$$

$$= Sa\ \{F\theta : q\varphi : \sim q\theta\}$$

$$= Sa\ \{q\varphi : T\theta\}$$

by theorems 5.4 and 5.9

$$= Sa\ \{\sim p\varphi : T\theta\}$$

$$= Sa\ \{p\theta : T\varphi\}$$

by theorem 5.6

and $Sa\ \{p\theta : q\varphi\} = Sa\ \{p\theta : \sim p\varphi\}$

$$= Sa\ \{p\theta : T\varphi\}$$

by theorem 5.9

Hence

$$Sa\ \{(p\wedge q)\ \theta : p\theta : q\varphi\} = Sa\ \{(p\wedge q)\ \theta : q\varphi : p\theta\} = Sa\ \{p\theta : q\varphi\}\ .$$

Theorem 5.12

$$Sa\ \{(p\wedge q)\ \theta : p\varphi : q\psi\} = Sa\ \{(p\wedge q)\ \theta : q\psi : p\varphi\}$$

Proof

$LHS = Sa\ \{(p\wedge q)\ \theta : (p\wedge q)\ \varphi : p\varphi : q\psi\}$ by theorem 5.7

$= Sa\ \{(p\wedge q)\ \theta : (p\wedge q)\ \varphi : q\psi : p\varphi\}$ by theorem 5.11

$= Sa\ \{(p\wedge q)\ \theta : q\psi : p\varphi\}$ by theorem 5.7

$= RHS$

§5.C Canonical Form of Conditional S-sequences

So far, we have expressed the ALGOL-60 expression

$$\underline{\text{if}}\ p_1\ \underline{\text{then}}\ \theta_1\ \underline{\text{else if}}\ p_2\ \underline{\text{then}}\ \theta_2 \cdots \underline{\text{else if}}\ p_n\ \underline{\text{then}}\ \theta_n$$

as a conditional S-sequence of the form

$$ST\ \{p_1\theta_1 : p_2\theta_2 : \cdots : p_n\theta_n\} \tag{5.13}$$

and derived some results soncerning the equivalence of sequences of this form.

Henceforth, we shall regard the above form of S-sequences as a canonical form and attempt to express other conditional S-sequence in this manner.

In general, we need to consider the following cases of conditional statements:-

a) $\underline{\text{if}}\ p_1\ \underline{\text{then}}\ \underline{\text{begin}}\ \underline{\text{if}}\ p_2\ \underline{\text{then}}\ \theta_1\ \underline{\text{else}}\ \theta_2\ \underline{\text{end}}$

$\underline{\text{else}}\ \underline{\text{begin}}\ \underline{\text{if}}\ p_3\ \underline{\text{then}}\ \theta_3\ \underline{\text{else}}\ \theta_4\ \underline{\text{end}}$

with its corresponding McCarthy conditional form

$$(p_1 \rightarrow (p_2 \rightarrow \theta_1 , \theta_2) , (p_3 \rightarrow \theta_3 , \theta_4))$$

and

b) a series of conditional statements

$\underline{\text{if}}\ p_1\ \underline{\text{then}}\ \theta_1\ \underline{\text{else}}\ \theta_2$

$\underline{\text{if}}\ p_2\ \underline{\text{then}}\ \theta_3\ \underline{\text{else}}\ \theta_4$

which may be represented by

$$(p_1 \rightarrow \theta_1 , \theta_2) \ ; \ (p_2 \rightarrow \theta_3 , \theta_4)$$

The semi-colon used here will be taken simply to mean "followed by".

Written as conditional S-sequences, the statements in a) and b) are respectively

$$ST\ \{p_1\ ST\ \{p_2\theta_1 : T\theta_2\}\ :\ T\ ST\{p_3\theta_3 : T\theta_4\}\ \}$$

and $ST\ \{p_1\theta_1 : T\theta_2\}\ ;\ ST\ \{p_2\theta_3 : T\theta_4\}$.

Such sequences may be transformed into the canonical form by the following theorems:-

Theorem 5.14

$$Sa\ \{p\ ST\ \{p_1\theta_1 : p_2\theta_2 : \cdots : p_n\theta_n\}\} = Sa\ \{(p\wedge p_1)\theta_1 : (p\wedge p_2)\theta_2 : \cdots : (p\wedge p_n)\theta_n\}$$

Proof

(i) $a \in E \Rightarrow LHS = a = RHS$

(ii) $a = T = p$

$$\Rightarrow LHS = ST\ \{T\ ST\ \{p_1\theta_1 : p_2\theta_2 : \cdots : p_n\theta_n\}\}$$
$$= ST\ \{p_1\theta_1 : p_2\theta_2 : \cdots : p_n\theta_n\}$$
$$RHS = ST\ \{(T\wedge p_1)\theta_1 : (T\wedge p_2)\theta_2 : \cdots : (T\wedge p_n)\theta_n\}$$
$$= ST\ \{p_1\theta_1 : p_2\theta_2 : \cdots : p_n\theta_n\}$$

(iii) $a = T,\ p = F$

$$\Rightarrow LHS = ST\ \{F\ ST\{p_1\theta_1 : p_2\theta_2 : \cdots : p_n\theta_n\}\}$$
$$= T$$
$$RHS = ST\ \{(F\wedge p_1)\theta_1 : (F\wedge p_2)\theta_2 : \cdots : (F\wedge p_n)\theta_n\}$$
$$= ST\ \{F\theta_1 : F\theta_2 : \cdots : F\theta_n\}$$
$$= T$$

Theorem 5.15

$$ST\ \{p_1\theta_1 : p_2\theta_2 : \cdots : p_n\theta_n\}\ ;\ ST\ \{q_1\varphi_1 : q_2\varphi_2 : \cdots : q_m\varphi_m\}$$

$$= ST\ \{(p_1\wedge q_1)\ [\theta_1\ ;\ \varphi_1] : (p_1\wedge q_2)\ [\theta_1 : \varphi_2] : \cdots : (p_1\wedge q_m)\ [\theta_1\ ;\ \varphi_m]:$$
$$(p_2\wedge q_1)\ [\theta_2\ ;\ \varphi_1] : \quad \cdots \quad : (p_2\wedge q_m)\ [\theta_2\ ;\ \varphi_m]:$$
$$\vdots$$
$$(p_n\wedge q_1)\ [\theta_n\ ;\ \varphi_1] : \quad \cdots \quad : (p_n\wedge q_m)\ [\theta_n : \varphi_m]\ \}.$$

Proof

Let p_i be the first p among $\{p_1, \ldots, p_n\}$ which has the value T and q_j be the first q among $\{q_1, \ldots, q_m\}$ which has the value T .

Hence LHS = ST $\{F\theta_1 : \ldots : T\theta_i : \ldots : p_n\theta_n\}$; ST $\{F\varphi_1 : \ldots : T\varphi_j : \ldots : q_m\varphi_m\}$

$= \theta_i ; \varphi_j$

Now $p_r = F \; \forall \; r < i$ and $q_s = F \; \forall \; s < j$

$\Rightarrow (p_r \wedge q_s) = F \quad \forall \; r < i, \; a < j$,

so the first $(p \wedge q)$ which has the value T is $(p_i \wedge q_j)$.

Hence RHS = ST $\{F[\theta_1 ; \varphi_1] : \ldots : F[\theta_1 ; \varphi_m]:$

$\vdots$

$F[\theta_i ; \varphi_1] : \ldots : T[\theta_i ; \varphi_j] : \ldots \}$

$= \theta_i ; \varphi_j$

$\therefore$ LHS = RHS

Thus, any conditional expression may be transformed into a canonical S-sequence by

1. Applying theorem 5.14 repeatedly until all the nested conditions are represented by a canonical S-sequence.

2. The expression now contains a series of canonical forms which may be transformed into a single canonical sequence by repeated applications of theorem 5.15.

eg. the conditional forms

$$(p \rightarrow (q \rightarrow \theta_1, \theta_2) \; , \; (r \rightarrow (t \rightarrow \theta_3, \theta_4) \; , \; \theta_5)) \; ; \; (p_1 \rightarrow \theta_6, \theta_7)$$

written as conditional S-sequences

$ST\{p\ ST\{q\theta_1 : T\theta_2\} : T\ ST\{r\ ST\{t\theta_3 : T\theta_4\} : T\theta_5\}\ ;\ ST\{p_1\theta_6 : T\theta_7\}$

may be transformed into a canonical sequence as follows:-

$ST\{p\ ST\{q\theta_1 : T\theta_2\} : T\ ST\{r\ ST\{t\theta_3 : T\theta_4\} : T\theta_5\}\ ;\ ST\{p_1\theta_6 : T\theta_7\}$

$\equiv ST\{(p\wedge q)\theta_1 : (p\wedge T)\theta_2 : T\ ST\{(r\wedge t)\theta_3 : (r\wedge t)\theta_4 : T\theta_5\}\ ;\ ST\{p_1\theta_6 : T\theta_7\}$

$= ST\{(p\wedge q)\theta_1 : p\theta_2 : (T\wedge r\wedge t)\theta_3 : (T\wedge r\wedge T)\theta_4 : (T\wedge T)\theta_5\}\ ;\ ST\{p_1\theta_6 : T\theta_7\}$

$= ST\{(p\wedge q)\theta_1 : p\theta_2 : (r\wedge t)\theta_3 : r\theta_4 : T\theta_5\}\ ;\ ST\{p_1\theta_6 : T\theta_7\}$

$= ST\{(p\wedge q\wedge p_1)[\theta_1 ; \theta_6] : (p\wedge q\wedge T)[\theta_1 ; \theta_7] : (p\wedge p_1)[\theta_2 ; \theta_6] :$
$(p\wedge T)[\theta_2 ; \theta_7] : (r\wedge t\wedge p_1)[\theta_3 ; \theta_6] : (r\wedge t\wedge T)[\theta_3 ; \theta_7] :$
$(r\wedge p_1)[\theta_4 ; \theta_6] : (r\wedge T)[\theta_4 ; \theta_7] : (T\wedge p_1)[\theta_5 : \theta_6] : (T\wedge T)[\theta_5 : \theta_7]\}$

$= ST\{(p\wedge q\wedge p_1)[\theta_1 ; \theta_6] : (p\wedge q)[\theta_1 ; \theta_7] : (p\wedge p_1)[\theta_2 ; \theta_6] : p[\theta_2 ; \theta_7] :$
$(r\wedge t\wedge p_1)[\theta_3 ; \theta_6] : (r\wedge t)[\theta_3 ; \theta_7] : (r\wedge p_1)[\theta_4 ; \theta_6] : r[\theta_4 ; \theta_7] :$
$p_1[\theta_5 ; \theta_6] : T[\theta_5 ; \theta_7]\}$

Note

To avoid confusing the issue at hand, we did not mention the possible effect of the unconditional statements $\theta_1, \ldots, \theta_n$ on the propositional expressions $q_1, \ldots, q_m$ in theorem 5.15.

In effect, the theorem seems to be applicable only when the q_j's are not functions of the θ_i's.

In fact, it can also be used even if q_j's are dependent on θ_i's provided it is applied after the transformation from the dynamic into the static state.

§5.D Conditional S-sequences with Undefined Value

In the previous sections, we have considered conditional statements

$$\underline{\text{if}}\ \ p\ \ \underline{\text{then}}\ \ \theta\ \ \underline{\text{else}}\ \ \varphi$$

in which the propositional expression p takes only the boolean values T and F.

It is, however, possible that the test condition p may be undefined ie. p may take the value ω and hence the statement yields an undefined value. In which case, the S-expression representing the conditional statement is

$$S\ (STp\theta)\ T\ (SpF\varphi)\ .$$

This is the expression given in (5.2) and has been shown to yield θ and φ according to whether p is T or F.

If $p = \omega$ then

$$\begin{aligned} S\ (STp\theta)\ T\ (SpF\varphi) &= S\ (ST\omega\theta)\ T\ (S\omega F\varphi) \\ &= STT\omega = \omega \end{aligned}$$

Hence the expression models the above conditional statement when $p \in \{T,F,\omega\}$.

To make provision for the possibility that some of the p's and θ's in a conditional form may be undefined, McCarthy [29] applies the following rule for the evaluation of

$$(p_1 \rightarrow \theta_1,\ p_2 \rightarrow \theta_2,\ \cdots,\ p_n \rightarrow \theta_n)\ \ :-$$

"If an undefined p occurs before a true p or if all the p's are false or if the θ corresponding to the first true p is undefined, then the form is undefined. Otherwise, the value of the form is the value of the θ corresponding to the first true p."

In the two-valued case we have considered earlier, the conditional form

$$(p_1 \rightarrow \theta_1,\ p_2 \rightarrow \theta_2,\ \cdots,\ p_n \rightarrow \theta_n) \tag{5.16}$$

corresponds to a simple S-expression

$$S\ (\ \cdots\ S\ (STp_1\theta_1)\ p_2\theta_2\ \cdots\ )\ p_n\theta_n$$

which, in a sequence notation, is

$$ST\ \{p_1\theta_1 : p_2\theta_2 : \cdots : p_n\theta_n\}\ .$$

To derive the S-expression representing (5.16) in the three valued case, we first consider an S-expression which mirrors the effect of

$$(q \rightarrow \varphi)\ .$$

That is

$$S\ (STq\varphi)\ T\ (SqF\omega)$$

which gives the value φ if $q = T$ and ω otherwise. (B.1)†

---

† See Appendix B for verification

Having shown that

$$S\ (STp\theta)\ T\ (SpF\varphi)$$

yields the same result as

$$(p \rightarrow \theta,\ \varphi)\ ,$$

it is now easy to deduce that

$$(p \rightarrow \theta,\ q \rightarrow \varphi)$$

should be represented by

$$(S\ (STp\theta)\ T\ (SpF\ \{S\ (STq\varphi)\ T\ (SqF\omega)\ \}\ )$$

as is substantiated in the Appendix B. (B2)†

Thus, for $p \in \{T,F,\omega\}$,

$$(p_1 \rightarrow \theta_1,\ p_2 \rightarrow \theta_2,\ \cdots\ ,\ p_n \rightarrow \theta_n)$$

corresponds to the S-expression

$$S\ (STp_1\theta_1)\ T\ (Sp_1F\ (S\ (STp_2\theta_2)\ T\ (Sp_2F\ \cdots\ S\ (STp_n\theta_n)\ T\ (Sp_nF\omega)\ \cdots\ ))).$$

This expression in three-valued logic is much too cumbersome to work with. We shall therefore employ the notation used in the two-valued case ie.

$$ST\ \{p_1\theta_1\ :\ p_2\theta_2\ :\ \cdots\ :\ p_n\theta_n\}$$

with the added condition that if either an undefined p occurs before a true p or all the p's are false, then the sequence is undefined. (5.17)

---

† See Appendix B for verification.

The next step is to investigate the validity of the theorems which hold in the two-valued case. In order to do so, we first define an extension of the propositional connectives in three-valued logic.

In McCarthy's conditional forms, these are

$$p \wedge q = (p \rightarrow q\ ,\ T \rightarrow F)$$
$$p \vee q = (p \rightarrow T\ ,\ T \rightarrow q)$$
$$\sim p = (p \rightarrow F\ ,\ T \rightarrow T)$$
$$p \supset q = (p \rightarrow q\ ,\ T \rightarrow T)$$

which yield the following truth tables

	$p \wedge q$	q = T	q = F	q = ω
	T	T	F	ω
p	F	F	F	F
	ω	ω	ω	ω

	$p \vee q$	q = T	q = F	q = ω
	T	T	T	T
p	F	T	F	ω
	ω	ω	ω	ω

	$\sim p$	
	T	F
p	F	T
	ω	ω

	$p \supset q$	q = T	q = F	q = ω
	T	T	F	ω
p	F	T	T	T
	ω	ω	ω	ω

In terms of S , the connectives are

$$p \wedge q = S\,(STpq)\,Tp$$

$$p \vee q = SpFq$$

$$\sim p = S\,(STpF)\,T\,(SpFT)$$

$$p \supset q = S\,(STpq)\,T\,(SpFT) \qquad (B3)^{\dagger}$$

This system of defining the logical connectives is unsymmetrical, but it is most appropriate to the theory of computation eg.

$$F \wedge \omega = F \quad \text{but} \quad \omega \wedge F = \omega$$

indicating that if $p = F$ then q need not be calculated to evaluate $p \wedge q$, but if p is undefined ie. the calculation of p does not terminate, then we never get round to computing q .

Under this system of three-valued logic, it is found that all the theorems proved for the two-valued case hold with the exception of theorem 5.5, the last part of theorem 5.11 and the corollary to theorem 5.8. (See Appendix C for verifcation of results.)

In conclusion, we shall present the theorems under the notions of strong and weak equivalence which are defined as follows:-

Two conditional S-sequences containing the propositional variables $p_1, p_2, \cdots, p_n$ are said to be <u>strongly equivalent</u> if they have the same value for all combinations of $p_1, \cdots, p_n \in \{T, F, \omega\}$.

† See Appendix B for verification.

They are <u>weakly equivalent</u> if they yield the same result for all values of $p_1, \ldots, p_n \in \{T, F\}$ only

Let $=_s$ denote strong equivalence and $=_w$ denote weak equivalence, then the theorems are

1. $Sa\ \{T\theta : p\varphi\} =_s Sa\ \{T\theta\}$

2. $Sa\ \{F\theta\} ==_s a$

3. $Sa\ \{p\theta : T\theta\} =_w Sa\ \{T\theta\}$

4. $Sa\ \{\sim p\theta : T\varphi\} =_s Sa\ \{p\varphi : T\theta\}$

5. $Sa\ \{p\theta : p\varphi\} =_s Sa\ \{p\theta\}$

6. $Sa\ \{p\theta : q\theta\} =_s Sa\ \{(p \vee q)\theta\}$

   Corollary : $Sa\ \{p\theta : q\theta\} =_w Sa\ \{q\theta : p\theta\}$

7. $Sa\ \{p\theta : \sim p\varphi\} =_s Sa\ \{p\theta : T\varphi\}$

8. $Sa\ \{(p \wedge q)\theta : p\varphi : r\theta : T\varphi\} =_s Sa\ \{(p \wedge q) \vee (\sim p \wedge r)\ \theta : T\varphi\}$

9. $Sa\ \{(p \wedge q)\theta : p\theta : q\varphi\} =_s Sa\ \{(p \wedge q)\ \theta : q\varphi : p\theta\}$

   $=_w Sa\ \{p\theta : q\varphi\}$

10. $Sa\ \{(p \wedge q)\theta' : p\varphi : q\psi\} =_s Sa\ \{(p \wedge q)\ \theta : q\psi : p\varphi\}$

11. $Sa\ \{p\ ST\ \{p_1\theta_1 : p_2\theta_2 : \ldots : p_n\theta_n\}\} =_s Sa\ \{(p \wedge p_1)\theta_1 : \ldots : (p \wedge p_n)\theta_n\}$

12. $ST\ \{p_1\theta_1 : p_2\theta_2 : \ldots : p_n\theta_n\}\ ;\ ST\ \{q_1\varphi_1 : q_2\varphi_2 : \ldots : q_m\varphi_m\}$

    $=_s \{(p_1 \wedge q_1)\ [\theta_1\ ;\ \varphi_1] : (p_1 \wedge q_2)\ [\theta_1\ ;\ \varphi_2] : \ldots : (p_1 \wedge q_m)\ [\theta_1\ ;\ \varphi_m] :$

    $(p_2 \wedge q_1)\ [\theta_2\ ;\ \varphi_1] : (p_2 \wedge q_2)\ [\theta_2\ ;\ \varphi_2] : \ldots : (p_2 \wedge q_m)\ [\theta_2\ ;\ \varphi_m] :$

    $\vdots$

    $(p_n \wedge q_1)\ [\theta_n\ ;\ \varphi_1] : (p_n \wedge q_2)\ [\theta_n\ ;\ \varphi_2] \cdots : (p_n \wedge q_m)\ [\theta_n\ ;\ \varphi_m]\}$.

Note

Since in this extension to three-valued logic $p \wedge q \neq q \wedge p$ and $p \vee p \neq q \vee p$ in general, it is important to observe the order of the variables in the propositional expression in the given results.

§5.E Conclusion

We have developed a canonical form for modelling conditional statements in terms of S-expressions. Theorems 5.14 and 5.15 enable us to transform any type of conditional statement into the canonical form and any possible reduction of such a form follows from the results of theorems 5.3 - 5.12.

Thus, if each element of the set E is considered as a dynamic sequence representing a set of assignment statements and the propositional expressions p_i are expressed as functions of some input data, the canonical S-sequence will delineate the result of a program under all possible conditions depending on the input data.

CHAPTER 6

ON CONTROL STRUCTURES

§6.A Preamble

In Chapter IV we investigated the problems of representing a program consisting purely of assignment statements, in terms of the substitution operator S . A model was given in the form of dynamic S-sequences. These sequences may be transformed and reduced to their canonical forms known as minimal static sequences, thus presenting a method for considering the equivalence of two such simple programs.

In this Chapter, we extend our consideration to include control structures.

In general, a dynamic S-sequence is of the form:

$$S\mu[\varkappa a_1 \varkappa b_1;\ \varkappa a_2 \varkappa b_2;\ \cdots\ ;\ \varkappa a_n \varkappa b_n]$$

consisting of the three components

1. the substitution operator S
2. the variable space of the program μ

and

3. the body of the sequence $[\varkappa a_1 \varkappa b_1;\ \varkappa a_2 \varkappa b_2;\ \cdots\ ;\ \varkappa a_n \varkappa b_n]$.

As the body of the sequence plays the major part in our consideration of S-sequences, it would be convenient to be able to refer to it in a more compact form. Henceforth, we shall denote the body of a dynamic sequence corresponding to a section of program X by βX eg. if X consists of a sequence of assignments

a := b

b := c

c := a

then the dynamic S-sequence representing X is

$$S_{\mu}[{}_{\mu}a_{\mu}b;\ {}_{\mu}b_{\mu}c;\ {}_{\mu}c_{\mu}a]$$

and thus

$$\beta X = [{}_{\mu}a_{\mu}b;\ {}_{\mu}b_{\mu}c;\ {}_{\mu}c_{\mu}a]\ .$$

Note that it is possible for a section of program to contain dummy statements or undefined statements. In such cases, if A is a dummy statement and B an undefined statement, we denote

βA by Λ , a dummy body

and βB by ω , an undefined body.

Furthermore, we shall use the end marker $\dashv$ to indicate the end of a program, eg. a complete program is of the form

A

B

C

$\dashv$

The end marker will be represented in the dynamic sequence in the same form.

Concatenation of Programs

Consider a section of program C which is a concatenation of programs A and B , ie.

$$C = \begin{cases} A & = A \ ; \ B \\ B & \end{cases}$$

In terms of dynamic S-sequences, the above sections of program are represented by

$$S\mu\beta C = \begin{cases} S\mu\beta A \\ S\mu\beta B \end{cases}$$

$$= S(S\mu\beta A)\beta B$$

$$= S\mu[\beta A \ ; \ \beta B] \ .$$

Hence the body of a dynamic sequence representing a concatenation of two sections of a program is the concatenation of the bodies of the corresponding dynamic sequences, ie.

$$\beta[A \ ; \ B] = [\ \beta A \ ; \ \beta B \ ]$$

The two special types of body, Λ and ω, have the following trivial properties:-

(i) $\beta X \ ; \Lambda = \Lambda \ ; \beta X = \beta X$

(ii) $\beta X \ ; \omega = \omega \ ; \beta X = \omega$.

It is also clear that for any sections of program A and B,

$$\beta A \ ; \dashv \ ; \beta B = \beta A \ ; \dashv$$

Transformation and Reduction of a Dynamic S-Sequence in Terms of its Subsequences

In the previous section, we indicated that the body of a dynamic sequence may be composed of two or more other bodies. In other words, a dynamic sequence may contain a number of subsequences.

To derive the canonical form of such a sequence, we may apply the transformation and reduction rules as described in Cahpter IV to the entire sequence, or perform the transition in terms of its subsequences.

That is, if a dynamic sequence T consists of a number of subsequences, eg.

$$T = S\mu[\ \beta A\ ;\ \beta B\ ;\ \beta C\ ]$$
$$= S\mu[\kappa a_1 \kappa x_1;\ \cdots\ ;\kappa a_i \kappa x_i];[\kappa b_1 \kappa y_1;\ \cdots\ ;\kappa b_j \kappa y_j];$$
$$[\kappa c_1 \kappa z_1;\ \cdots\ ;\kappa c_k \kappa z_k]]$$

then the canonical form of T may be obtained by

(a) transforming and reducing the sequence

$$T = S\mu[\kappa a_1 \kappa x_1;\cdots;\kappa a_i \kappa x_i;\kappa b_1 \kappa y_1;\cdots;\kappa b_j \kappa y_j;\kappa c_1 \kappa z_1;\cdots;\kappa c_k \kappa z_k]$$

or

(b) transforming and reducing each subsequence βA, βB and βC to its canonical form separately and then considering each one relative to its predecessors.

The alternative process (b) can be stated more precisely as follows:

If the body βX of the dynamic sequence representing a program X consists of subsequences βX_1, βX_2, $\cdots$, βX_n , then βX may be reduced to its minimal static form by

(i) reducing each βX_i independently to its minimal static form by the procedure described in §4.D and §4.E of Chapter IV.

(ii) starting with $i=1$, transform the dynamic sequence βX_{i+1} to its static form with respect to the sequence βX_i ,

perform any possible reduction within the two subsequences and regard the result as βX_{i+1} . Increase i and repeat the process until all subsequences have been considered. The canonical form of βX is then given by βX_n .

Example

$$T = S\mu[[\eta d\eta a;\eta a\eta b];[\eta b\eta d;\eta d\eta c;\eta e\eta d;\eta d\eta a];[\eta c\eta b;\eta a\eta d;\eta e\eta a]]$$

(a) Transforming T as one sequence:

$$T = S\mu[\eta d\eta a;\eta a\eta b;\eta b\eta d;\eta d\eta c;\eta e\eta d;\eta d\eta a;\eta c\eta b;\eta a\eta d;\eta e\eta a]$$
$$= S\mu[\eta d\eta a,\eta a\eta b,\eta b\eta a,\eta d\eta c,\eta e\eta c,\eta d\eta b,\eta c\eta a,\eta a\eta b,\eta e\eta b]$$
$$= S\mu[\eta d\eta b,\eta a\eta b,\eta b\eta a,\eta e\eta b,\eta c\eta a]$$

(b) Transformation of T in terms of its subsequences:

$$\beta X_1 = [\eta d\eta a;\eta a\eta b]$$

$$\beta X_2 = [\eta b\eta d;\eta d\eta c;\eta e\eta d;\eta d\eta a]$$

$$\beta X_3 = [\eta c\eta b;\eta a\eta d;\eta e\eta a]$$

Reduce each βX_i to its minimal form giving

$$\beta X_1 = [\eta d\eta a,\eta a\eta b]$$

$$\beta X_2 = [\eta b\eta d,\eta d\eta c,\eta e\eta c,\eta d\eta a]$$
$$= [\eta b\eta d,\eta d\eta a,\eta e\eta c]$$

$$\beta X_3 = [\eta c\eta b,\eta a\eta d,\eta e\eta d]$$

Transform βX_2 with respect to βX_1 , reduce the two subsequences to give

$$\beta X_2 = \beta X_1 \; ; \; \beta X_2$$

$$= [\mu d_\mu a, \mu a_\mu b]; [\mu b_\mu d, \mu d_\mu a, \mu e_\mu c]$$

$$= [\mu d_\mu a, \mu a_\mu b], [\mu b_\mu a, \mu d_\mu b, \mu e_\mu c]$$

$$= [\mu d_\mu b, \mu a_\mu b, \mu b_\mu a, \mu e_\mu c]$$

Transform βX_3 with respect to βX_2 and reduce

$$\beta X_3 = \beta X_2 \; ; \; \beta X_3$$

$$= [\mu d_\mu b, \mu a_\mu b, \mu b_\mu a, \mu e_\mu c]; [\mu c_\mu b, \mu a_\mu d, \mu e_\mu d]$$

$$= [\mu d_\mu b, \mu a_\mu b, \mu b_\mu a, \mu e_\mu c], [\mu c_\mu a, \mu a_\mu b, \mu e_\mu b]$$

$$= [\mu d_\mu b, \mu a_\mu b, \mu b_\mu a, \mu e_\mu b, \mu c_\mu a]$$

The minimal form of T is

$$T = S_\mu [\mu d_\mu b, \mu a_\mu b, \mu b_\mu a, \mu e_\mu b, \mu c_\mu a] \; .$$

The transformation of dynamic sequences to the canonical form in terms of their subsequences is more suitable for programs which are more complicated than the one given in the example. Its real value is in the transformation of more general programs, eg. those containing control statements, blocks, etc.. Such programs are considered in the investigation which follows.

In our discussion of high-level languages, we shall base our models on a high-level language with the essential features of Algol-60 with occasional reference to other programming languages whenever appropriate.

Moreover, we shall assume that all the programs under consideration are syntactically correct.

§6.B The 'Value' of a Label

Before constructing the S-models for control statements, we first consider the notion of the 'value' of a label.

The evaluation operator $\varkappa$ has been defined in Chapter IV as a function with domain $N \cup V$ and range $R \subset V$, where N denotes a set of names and V denotes a set of values such that

$$\varkappa x = \alpha$$

where $\alpha \in V$ is the value corresponding to $x \in N$, and

$$\varkappa \alpha = \alpha \qquad \forall\, \alpha \in V\,.$$

We now extend the domain of $\varkappa$ to include a set of labels such that the operation of $\varkappa$ upon a label yields what may be simply understood as the 'value' of the label.

Definition

Let L be a set of labels, then

$$\varkappa l \qquad \forall\, l \in L$$

is defined to be all the statements of the program following the label l up to and including the first unconditional goto statement, if one exists, otherwise to the end of the program.

Examples

In the following examples the upper-case letters denote sections of program which do not contain control statements.

1.

```
      A
l1:   B                          ⎤
      if p then goto l2          ⎥
      C                          ⎥  μl1
      goto l1                    ⎦
l2:   D                          ⎤
      if q then E else goto l1   ⎥
      F                          ⎥  μl2
      -|                         ⎦
```

2.

```
      A
l1:   B                                          ⎤
      if p then goto l2                          ⎥
           else if q then C                      ⎥
                     else goto l3                ⎥
                                    ⎤            ⎥  μl1
l2:   D                             ⎥  μl2       ⎥
      goto l4                       ⎦            ⎥
      E                                          ⎥
      .                                          ⎥
      .                                          ⎥
      .                                          ⎦
```

3.

```
      A
      goto l
      B
l:    C          ⎤
      -|         ⎦  μl
```

4.

```
        A
    1:  if p then ( B
                      goto 1 )
                  else C
        D
        -|
```

$\varkappa 1$

In modelling a program containing labels, we shall make use of this notation of the value of a label to present the dynamic sequence in the simplest possible form. That is, a statement X will be represented in the S-form on its own only if it is not included in the range of any label, otherwise it will appear in a dynamic sequence as part of the value of a label.

In the given examples, the programs will be visualized as follows:

1.

A

$\varkappa 11$

$\varkappa 12$

2.

A

$\varkappa 11$

E

.

.

.

3.

A

goto 1

B

$\varkappa 1$

4.

A

$\varkappa 1$

Now, for $l \in L$, $\varkappa l$ gives a section of program, say X ie. $\varkappa l = X$ hence, the body of the dynamic sequence corresponding to the value of l is given by

$$\beta X = \beta \varkappa l$$

For example, the dynamic sequence representing the program in example 2 is

$$S_\mu[\ \beta A\ ;\ \beta \varkappa l 1\ ;\ \beta E\ ;\ \ldots\ ]$$

§6.C GOTO Statement

Consider a program

```
      A
      goto l
      B                                   (6.1)
l:    C
      D
      -|
```

The statement goto l causes the piece of program following l to be executed after A to give

```
      A
      C
      D
      -|
```

ie. the result is simply the concatenation of A and $\varkappa l$.

Thus goto 1 may be modelled simply by

$\varkappa 1$

eg. (6.1) corresponds to

$S_\mu \beta A$

$\varkappa 1$

$S_\mu \beta B$

$\varkappa 1$

where

$\varkappa 1 = S_\mu \beta C$

$S_\mu \beta D$

$\dashv$

That is, the dynamic sequence for (6.1) is

$S_\mu [\ \beta A\ ;\ \beta \varkappa 1\ ;\ \beta B\ ;\ \beta \varkappa 1\ ]$

where

$\varkappa 1 = S_\mu [\ \beta C\ ;\ \beta D\ ;\ \dashv\ ]$.

Thus, substituting for $\beta \varkappa 1$ gives

$S_\mu [\ \beta A\ ;\ \beta \varkappa 1\ ;\ \beta B\ ;\ \beta \varkappa 1\ ]$

$= S_\mu [\ \beta A\ ;\ \beta C\ ;;\beta D\ ;\ \dashv\ ;\ \beta B\ ;\ \beta C\ ;\ \beta D\ ;\ \dashv\ ]$

$= S_\mu [\ \beta A\ ;\ \beta C\ ;\ \beta D\ ;\ \dashv\ ]$.

The above example also illustrates the need for the end marker.

§6.D Conditional Statements

In Chapter V, we consider conditional S-sequences of the form

$$ST\{p_1\theta_1 : p_2\theta_2 : \cdots : p_n\theta_n\} \qquad p_i \in \{T,F,\omega\}$$

which yield the value θ corresponding to the first p which has the value true. If either an undefined p occurs before a true p or all p's are false, then the sequence yields an undefined value.

If q is a propositional expression, then the evaluation operator $\varkappa$ acting upon q will give a value belonging to the set $\{T,F,\omega\}$ [Chapter IV] .

ie. $\varkappa q \in \{T,F,\omega\}$

Thus, a conditional statement of the form

$$\underline{\text{if}}\ q\ \underline{\text{then}}\ \theta\ \underline{\text{else}}\ \varphi$$

where q is a propositional expression

θ and φ are simple variables or values

may be modelled by the conditional S-sequence

$$ST\{\ \varkappa q\theta\ :\ T\varphi\} \tag{6.2}$$

Similarly, if θ and φ represent sections of program X and Y say, then the model of a conditional statement of the form

$$\underline{\text{if}}\ q\ \underline{\text{then}}\ X\ \underline{\text{else}}\ Y$$

is $$ST\{\ \varkappa q\ S_{\mu\beta}X\ :\ T\ S_{\mu\beta}Y\ \} \tag{6.3}$$

Now a program such as

```
Z
if q then X else Y                    (6.4)
```

is represented line-by-line by

```
SμβZ
ST{ κq SμβX : T SμβY }                (6.5)
```

The above conditional S-sequence represents

```
SμβX
or  SμβY
or  ω
```

according as $\kappa q = T, F$ or ω .

That is, (6.5) has three alternatives, ie.

```
SμβZ          SμβZ          SμβZ
      or            or
SμβX          SμβY          ω
```

which are

$$S_\mu[\ \beta Z\ ;\ \beta X\ ] \quad \text{or} \quad S_\mu[\ \beta Z\ ;\ \beta Y\ ] \quad \text{or} \quad S_\mu[\ \beta Z\ ;\ \omega\ ]$$

in terms of dynamic S-sequences.

Thus, the representation of (6.5) as a dynamic sequence is

$$S_\mu[\ \beta Z\ ;\ ST\{\ \kappa q\ [\beta X]\ :\ T[\beta Y]\ \}\ ]$$

To simplify such a representation, we denote a conditional sequence

$$ST\{p_1\theta_1 : p_2\theta_2 : \cdots : p_n\theta_n\}$$

by $\{p_1\theta_1 : p_2\theta_2 : \cdots : p_n\theta_n\}$

hence the program (6.4) is represented by the dynamic S-sequence

$$S\mu[\ \beta Z\ ;\ \{\ \nu q\ [\beta X]\ :\ T\ [\beta Y]\ \}\ ]$$

eg. x := a

y := b

if x>y then z := x else z := y

is modelled by

$$S\mu[\ \nu x \nu a ; \nu y \nu b ; \{\nu(x>y)\ [\nu z \nu x]\ :\ T\ [\nu z \nu y]\}\ ]$$

which gives $S\mu[\ \nu x \nu a ; \nu y \nu b ; \nu z \nu x\ ]$ if $\nu(x>y) = T$

and $S\mu[\ \nu x \nu a ; \nu y \nu b ; \nu z \nu y\ ]$ if $\nu(x>y) = F$.

Following the models given in (6.2) and (6.3), it is clear that simple conditional statements such as

if q then θ

and if q then X

where θ is a simple variable or a value

X is a section of program

are represented by the conditional S-sequences

$$\{\ \nu q\ \theta\ :\ T\Lambda\ \}$$

and $\{\ \nu q\ S\mu\beta X\ :\ T\Lambda\ \}$.

Generalized Assignments

For a simple assignment

$$x := y$$

the S-model is $S_{\mu\varkappa x\varkappa y}$

The same structure may be used in the case of generalized assignments.

Consider the statements

$$\begin{aligned} &(\underline{\text{if}}\ p\ \underline{\text{then}}\ x\ \underline{\text{else}}\ y) := a \\ \text{and}\quad &x := (\underline{\text{if}}\ p\ \underline{\text{then}}\ a\ \underline{\text{else}}\ b) \end{aligned} \tag{6.6}$$

Combining the assignment model with the model of a conditional statement (6.2), we obtain the following representations for the above statements:

$$S_\mu[\varkappa\{\varkappa p\ x{:}Ty\}\ \varkappa a]$$

and $$S_\mu[\varkappa x\ \varkappa\{\varkappa p\ a{:}Tb\}]$$

The conditional $\{\varkappa p\ x{:}Ty\}$ gives x if $\varkappa p = T$ and y otherwise, thus

$$S_\mu[\varkappa\{\varkappa p\ x{:}Ty\}\ \varkappa a] = \begin{cases} S_{\mu\varkappa x\varkappa a} & \text{if } \varkappa p = T \\ S_{\mu\varkappa y\varkappa a} & \text{if } \varkappa p = F \end{cases}$$

From the discussion of the last section, the situation on the RHS may also be represented by:

$$S_\mu[\{\varkappa p[\varkappa x\varkappa a]{:}T[\varkappa y\varkappa a]\}]\ ,$$

we thus deduce the following transformation:

$$S_\mu[\varkappa\{\varkappa p\ x{:}Ty\}\ \varkappa a] = S_\mu[\{\varkappa p[\varkappa x\varkappa a]{:}T[\varkappa y\varkappa a]\}]\ .$$

Similarly, it can be shown that

$$S\mu[\nu x\ \nu\{\nu p\ a{:}Tb\}] \;=\; S\mu[\{\nu p[\nu x_{\nu} a]{:}T[\nu x_{\nu} b]\}] \ .$$

Hence a program containing a conditional form, either as part of a generalized assignment or as a transfer of control, may be represented by a dynamic S-sequence of the form

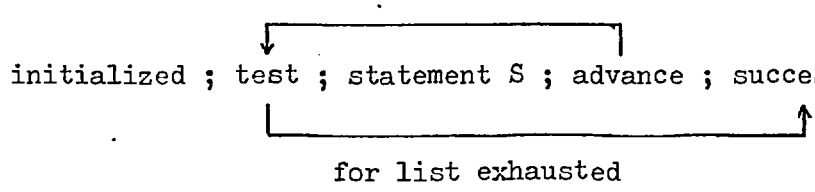
$$S\mu[\ \ldots;\ \{\ \nu p\ \theta\ :\ T\varphi\ \}\ ;\ \ldots\ ]$$

where θ and φ are of the forms βX, βY for some sections of program X and Y .

Loops

The main function of a loop in a programming language is to enable a section of a program to be executed repeatedly. The various constructions for writing a loop consist basically of the same operations although the order of operation may vary from language to language. Here we shall consider Algol-60 for-statement as an example of a loop.

According to the Algol-60 Report [30]
"Algol-60 for-clause causes statement S which it precedes to be repeatedly executed zero or more times. In addition it performs a sequence of assignments to its control variables. The process may be visualized as follows:



Thus, a for-statement of the form

$$\underline{\text{for}}\ \text{i:=1}\ \underline{\text{step}}\ \text{m}\ \underline{\text{until}}\ \text{n}\ \underline{\text{do}}\ \text{S} \tag{6.7}$$

may be described in terms of the following statements:

```
      i := 1
L:    if i≤n then ( S
                    i := i+m
                    goto L )
```
(6.8)

Note

The open and closed brackets are used here to indicate the range of control of *if* ... *then* ... instead of *begin* and *end* which will be considered in due course.

It can be seen that the construction of a loop is based upon a conditional statement and a *goto* statement which have already been discussed. Thus in the light of (6.8), we model the for-statement (6.7) by

$$S_\mu[\ \varkappa i_\varkappa 1\ ;\ \beta_\varkappa L\ ]$$

where $\varkappa L = S_\mu[\{\varkappa(i\leq n)[\beta S;\varkappa i_\varkappa(i+m);\beta_\varkappa L] : T\Lambda\}]$.

The Algol-60 while-statement may be considerd in a similar manner, ie.

$$\underline{\text{while}}\ \text{p}\ \underline{\text{do}}\ \text{S}$$

is equivalent to

```
L:    if p then ( S
                  goto L )
```

which is represented by

$$S\mu[\ \beta\varkappa L\ ]$$

where $\varkappa L = S\mu[\{\varkappa p[\beta S;\beta\varkappa L]:T\Lambda\}]$.

§6.E Transformation of Programs with Control Statements

Consider the program

```
x := a
y := b
if x>y then z := x else z := y
z := z+z
-|                                        (6.9)
```

which is of the form

```
A
if p then B else C
D
-|
```

The dynamic sequence corresponding to the above form is

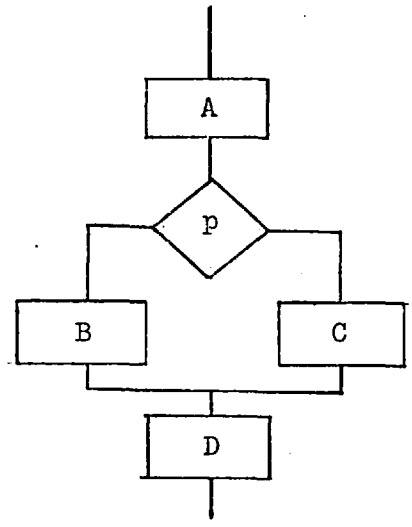
$$S\mu[\ \beta A\ ;\ \{\ \varkappa p[\beta B]\ :\ T[\beta C]\ \}\ \beta D\ ;\ \neg|\ ] \qquad (6.10)$$

To transform this sequence into a static sequence we shall use the subsequence method described in §6.A. The first step is the transformation of βA, βB, βC, βD independently into their minimal forms. The next step is slightly more complicated. Not only have βB, βC and βD to be transformed according to their preceding subsequences, but $\varkappa p$ must also be adjusted with respect to the preceding subsequence βA [Chapter IV].

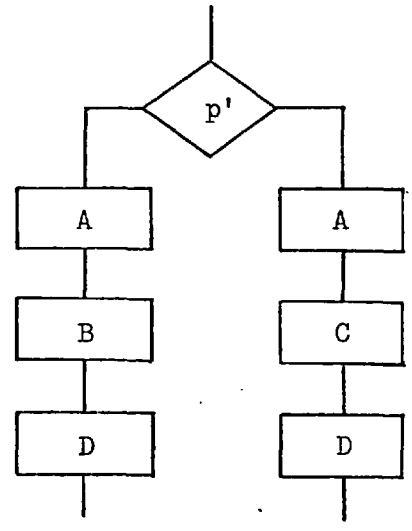
Let $\varkappa p'$ represent the transformation of $\varkappa p$ relative to βA, then the sequence (6.10) becomes

$$\begin{aligned} & S\mu[\{ \varkappa p'[\beta A;\beta B] : T[\beta A;\beta C] \} ; \beta D ; \dashv] \\ = \; & S\mu[\{ \varkappa p'[\beta A;\beta B;\beta D;\dashv] : T[\beta A;\beta C;\beta D;\dashv]\})] \end{aligned} \tag{6.11}$$

That is, a program represented by the flow diagram



is transformed into one of the form



From (6.11), the sequences [βA; βB; βD] and [βA; βC; βD] are then reduced to the minimal forms. The resulting static sequence delineates the final values of the variables involved according to the truth of p'.

Example 6.12

The dynamic sequence corresponding to the program (6.9) is

$$S_\mu[\kappa x_\kappa a;\kappa y_\kappa b;\{\kappa(x>y)[\kappa z_\kappa x]:T[\kappa z_\kappa y]\};\kappa z_\kappa(z+z);\dashv]$$

which is transformed and reduced as follows:

$$\begin{aligned}
&S_\mu[\kappa x_\kappa a;\kappa y_\kappa b;\{\kappa(x>y)[\kappa z_\kappa x]:T[\kappa z_\kappa y]\};\kappa z_\kappa(z+z);\dashv]\\
=\ &S_\mu[\{\kappa(a>b)[\kappa x_\kappa a;\kappa y_\kappa b;\kappa z_\kappa x]:T[\kappa x_\kappa a;\kappa y_\kappa b;\kappa z_\kappa y]\};\kappa z_\kappa(z+z);\dashv]\\
=\ &S_\mu[\{\kappa(a>b)[\kappa x_\kappa a,\kappa y_\kappa b,\kappa z_\kappa a]:T[\kappa x_\kappa a,\kappa y_\kappa b,\kappa z_\kappa b]\};\kappa z_\kappa(z+z);\dashv]\\
=\ &S_\mu[\{\kappa(a>b)[\kappa x_\kappa a,\kappa y_\kappa b,\kappa z_\kappa a;\kappa z_\kappa(z+z);\dashv]:T[\kappa x_\kappa a,\kappa y_\kappa b,\kappa z_\kappa b;\kappa z_\kappa(z+z);\dashv]\}]\\
=\ &S_\mu[\{\kappa(a>b)[\kappa x_\kappa a,\kappa y_\kappa b,\kappa z_\kappa a,\kappa z_\kappa(a+a),\dashv]:T[\kappa x_\kappa a,\kappa y_\kappa b,\kappa z_\kappa b,\kappa z_\kappa(b+b),\dashv]\}]\\
=\ &S_\mu[\{\kappa(a>b)[\kappa x_\kappa a,\kappa y_\kappa b,\kappa z_\kappa(a+a),\dashv]:T[\kappa x_\kappa a,\kappa y_\kappa b,\kappa z_\kappa(b+b),\dashv]\}]
\end{aligned}$$

The minimal sequence obtained shows that x and y have the values a and b respectively but the final value of z is either (a+a) or (b+b) depending on whether a>b or not.

The program given in the example contains only one conditional statement which leads to the minimal sequence consisting of two possible sets of final values. Programs containing n conditionals may be reduced in the same manner with the conditionals transformed to the canonical form as discussed in theorems 5.14 and 5.15 of Chapter V. Some reduction may be possible as shown in theorems 5.3 to 5.12 of the same chapter, but for a large n the number of possibilities can still be quite large.

In the case of programs containing backward goto's, it is not as simple to produce the final static sequence as in the case of the last example.

The S-sequence of a non-terminating program will display the same behaviour as is shown in Example 6.13 to follow. Some sequences involving loops may be reduced to their minimal forms with the help of induction as in Example 6.15.

Example 6.13

```
      x := a
  L:  x := x+1
      goto L
      ⊣
```

The dynamic sequence for the above program is

$T = S_\mu[{}_\mu x_\mu a; \beta_\mu L; \dashv]$ where $\beta_\mu L = [{}_\mu x_\mu (x+1); \beta_\mu L]$

$T = S_\mu[{}_\mu x_\mu a; {}_\mu x_\mu (x+1); \beta_\mu L; \dashv]$

$= S_\mu[{}_\mu x_\mu a, {}_\mu x_\mu (a+1); \beta_\mu L; \dashv]$

$= S_\mu[{}_\mu x_\mu a, {}_\mu x_\mu (a+1); {}_\mu x_\mu (x+1); \beta_\mu L; \dashv]$

$= S_\mu[{}_\mu x_\mu a, {}_\mu x_\mu (a+1), {}_\mu x_\mu (a+1+1); \beta_\mu L; \dashv]$

$= S_\mu[{}_\mu x_\mu a, {}_\mu x_\mu (a+1+1); {}_\mu x_\mu (x+1); \beta_\mu L; \dashv]$

$= S_\mu[{}_\mu x_\mu a, {}_\mu x_\mu (a+1+1+1); {}_\mu x_\mu (x+1); \beta_\mu L; \dashv]$

...

Clearly, the sequence T does not terminate.

Example 6.14

```
      x := a
      y := b
  L:  if x>0 then ( y := y+1
                    x := x-1
                    goto L )
```

The dynamic sequence corresponding to the above piece of program is

$$T = S_\mu[\kappa x_\kappa a;\kappa y_\kappa b;\beta_\kappa L] \quad \text{where}$$

$$\beta_\kappa L = \{\kappa(x>0)[\kappa y_\kappa(y+1);\kappa x_\kappa(x-1);\beta_\kappa L]:T\Lambda\}$$

$$\text{Hence} \quad T = S_\mu[\kappa x_\kappa a;\kappa y_\kappa b;\{\kappa(x>0)[\kappa y_\kappa(y+1);\kappa x_\kappa(x-1);\beta_\kappa L]:T\Lambda\}]$$

$$= S_\mu[\kappa x_\kappa a,\kappa y_\kappa b,\{\kappa(a>0)[\kappa y_\kappa(b+1),\kappa x_\kappa(a-1);\beta_\kappa L]:T\Lambda\}]$$

$$= S_\mu[\{\kappa(a>0)[\kappa x_\kappa(a-1),\kappa y_\kappa(b+1);\beta_\kappa L]:T[\kappa x_\kappa a,\kappa y_\kappa b]\}]$$

If $a>0$, then $\kappa(a>0)=T$ and $\kappa(a-i>0)$ will have the value T for $i=1,2,\ldots,a-1$. Hence the first subsequence in the above sequence gives

$$[\kappa x_\kappa(a-1);\kappa y_\kappa(b+1);\{\kappa(x>0)[\kappa y_\kappa(y+1);\kappa x_\kappa(x-1);\beta_\kappa L]:T\Lambda\}]$$

$$= [\kappa x_\kappa(a-1),\kappa y_\kappa(b+1),\{\kappa(a-1>0)[\kappa y_\kappa(b+2),\kappa x_\kappa(a-2);\beta_\kappa L]:T\Lambda\}]$$

$$= [\kappa x_\kappa(a-1),\kappa y_\kappa(b+1),\kappa y_\kappa(b+2),\kappa x_\kappa(a-2);\beta_\kappa L]$$

$$= [\kappa x_\kappa(a-2),\kappa y_\kappa(b+2);\{\kappa(x>0)[\kappa y_\kappa(y+1);\kappa x_\kappa(x-1);\beta_\kappa L]:T\Lambda\}]$$

$$\cdots$$

$$= [\kappa x_\kappa(a-a),\kappa y_\kappa(b+a);\{\kappa(x>0)[\kappa y_\kappa(y+1);\kappa x_\kappa(x-1);\beta_\kappa L]:T\Lambda\}]$$

by induction

$$= [\kappa x_\kappa 0,\kappa y_\kappa(b+a),\{\kappa(0>0)[\kappa y_\kappa(b+a+1),\kappa x_\kappa(-1);\beta_\kappa L]:T\Lambda\}]$$

$$= [\kappa x_\kappa 0,\kappa y_\kappa(b+a),\Lambda]$$

$$= [\kappa x_\kappa 0,\kappa y_\kappa(b+a)]$$

Hence the minimal static form of T is

$$T = S_\mu[\{\kappa(a>0)[\kappa x_\kappa 0,\kappa y_\kappa(b+a)]:T[\kappa x_\kappa a,\kappa y_\kappa b]\}]$$

indicating that if $a>0$, then the final values of x and y are 0 and $b+a$ respectively, otherwise $x=a$, $y=b$.

Example 6.15

```
      x := a
      y := b
      z := c
  L:  if x<z then ( t := x; x := y; y := t )
      if y<z then ( t := y; y := z; z := t; goto L )
```

For the above program, the dynamic sequence is

$$T = S_{\mu}[\mu x \mu a; \mu y \mu b; \mu z \mu c; \beta \mu L] \quad \text{where}$$

$$\beta \mu L = \{\mu(x<y)[\mu t \mu x; \mu x \mu y; \mu y \mu t]:T\Lambda\} \; ; \; \{\mu(y<z)[\mu t \mu y; \mu y \mu z; \mu z \mu t; \beta \mu L]:T\Lambda\}$$

Perform the transformation on $\beta \mu L$:-

$$\beta \mu L = \{\mu(x<y)[\mu t \mu x, \mu x \mu y, \mu y \mu x]:T\Lambda\} \; ; \; \{\mu(y<z)[\mu t \mu y, \mu y \mu z, \mu z \mu y; \beta \mu L]:T\Lambda\}$$

$$= \{\mu(x<y) \wedge \mu(x<z)[\mu t \mu x, \mu x \mu y, \mu y \mu x; \mu t \mu y, \mu y \mu z, \mu z \mu y; \beta \mu L] :$$

$$\mu(x<y) \wedge T[\mu t \mu x, \mu x \mu y, \mu y \mu x; \Lambda] :$$

$$T \wedge \mu(y<z)[\Lambda; \mu t \mu y, \mu y \mu z, \mu z \mu y; \beta \mu L] :$$

$$T \wedge T [\, \Lambda; \Lambda \,] \; \} \qquad \text{by theorem 5.15}$$

$$= \{\mu(x<y) \wedge \mu(x<z)[\mu t \mu x, \mu x \mu y, \mu y \mu z, \mu z \mu x; \beta \mu L] :$$

$$\mu(x<y)[\mu t \mu x, \mu x \mu y, \mu y \mu x] : \mu(y<z)[\mu t \mu y, \mu y \mu z, \mu z \mu y; \beta \mu L] : T\Lambda\}$$

The first subsequence $[\mu t \mu x, \mu x \mu y, \mu y \mu z, \mu z \mu x; \beta \mu L]$ leads to

$$\mu t \mu x, \mu x \mu y, \mu y \mu z, \mu z \mu x; \{\mu(x<y) \wedge \mu(x<z)[\mu t \mu x, \mu x \mu y, \mu y \mu z, \mu z \mu x; \beta \mu L] :$$

$$\mu(x<y)[\mu t \mu x, \mu x \mu y, \mu y \mu x] :$$

$$\mu(y<z)[\mu t \mu y, \mu y \mu z, \mu z \mu y; \beta \mu L] : T\Lambda \quad \}$$

which is reduced to

$$\{\nu(y<z)\wedge\nu(y<x)[\nu t\nu y,\nu x\nu z,\nu y\nu x,\nu z\nu y;\beta\nu L] :$$
$$\nu(y<z)[\nu t\nu y,\nu x\nu z,\nu z\nu x] : \nu(z<x)[\nu t\nu z,\nu x\nu y,\nu y\nu x;\beta\nu L] :$$
$$T[\nu t\nu x,\nu x\nu y,\nu y\nu z,\nu z\nu x] \}$$

Similarly, the third subsequence $[\nu t\nu y,\nu y\nu z,\nu z\nu y;\beta\nu L]$ gives:-

$$\{\nu(x<z)\wedge\nu(x<y)[\nu t\nu x,\nu x\nu z,\nu z\nu x;\beta\nu L] :$$
$$\nu(x<z)[\nu t\nu x,\nu x\nu z,\nu y\nu x,\nu z\nu y] : \nu(z<y)[\nu t\nu z;\beta\nu L] :$$
$$T[\nu t\nu y,\nu y\nu z,\nu z\nu y] \}$$

Hence, after one stage $\beta\nu L$ is

$$\beta\nu L = \{\nu(x<y)\wedge\nu(x<z)\wedge\nu(y<z)\wedge\nu(y<x)[\nu t\nu y,\nu x\nu z,\nu y\nu x,\nu z\nu y;\beta\nu L] :$$
$$\nu(x<y)\wedge\nu(x<z)\wedge\nu(y<z)[\nu t\nu y,\nu x\nu z,\nu z\nu x] :$$
$$\nu(x<y)\wedge\nu(x<z)\wedge\nu(z<x)[\nu t\nu z,\nu x\nu y,\nu y\nu x;\beta\nu L] :$$
$$\nu(x<y)\wedge\nu(x<z)[\nu t\nu x,\nu x\nu y,\nu y\nu z,\nu z\nu x] :$$
$$\nu(x<y)[\nu t\nu x,\nu x\nu y,\nu y\nu x] :$$
$$\nu(y<z)\wedge\nu(x<z)\wedge\nu(x<y)[\nu t\nu x,\nu x\nu z,\nu z\nu x;\beta\nu L] :$$
$$\nu(y<z)\wedge\nu(x<z)[\nu t\nu x,\nu x\nu z,\nu y\nu x,\nu z\nu y] :$$
$$\nu(y<z)\wedge\nu(z<y)[\nu t\nu z;\beta\nu L] : \nu(y<z)[\nu t\nu y,\nu y\nu z,\nu z\nu y] : T\Lambda \}$$

At this stage, we observe that some of the subsequences contain the term $\beta\nu L$ and thus require further substitution and it seems as if the process might go on indefinitely. However, a closer inspection reveals that some of the conditions always have the value F , eg. the expression $\nu(x<y)\wedge\nu(x<z)\wedge\nu(z<x)$. Thus, the subsequences

following such conditions may be discarded according to theorem 5.4. Hence, we obtain a reduced sequence

$$\begin{aligned}\beta_\mu L = \{&\mu(x<y)\wedge_\mu(x<z)\wedge_\mu(y<z)[\mu t_\mu y,\mu x_\mu z,\mu z_\mu x] :\\ &\mu(x<y)\wedge_\mu(x<z)[\mu t_\mu x,\mu x_\mu y,\mu y_\mu z,\mu z_\mu x] :\\ &\mu(x<y)[\mu t_\mu x,\mu x_\mu y,\mu y_\mu x] :\\ &\mu(y<z)\wedge_\mu(x<z)\wedge_\mu(x<y)[\mu t_\mu x,\mu x_\mu z;\mu z_\mu x;\beta_\mu L] :\\ &\mu(y<z)\wedge_\mu(x<z)[\mu t_\mu x,\mu x_\mu z,\mu y_\mu x,\mu z_\mu y] :\\ &\mu(y<z)[\mu t_\mu y,\mu y_\mu z,\mu z_\mu y] : T\Lambda \quad \}\end{aligned}$$

The only subsequence which requires further attention is $[\mu t_\mu x,\mu x_\mu z,\mu z_\mu x;\beta_\mu L]$. On further substitution of the subsequence and reduction of the terms with the value F as before, the term under consideration is found to be simply

$$\mu(y<z)\wedge_\mu(x<z)\wedge_\mu(x<y)[\mu t_\mu x,\mu x_\mu z,\mu z_\mu x]$$

Hence, we obtain the minimal form of $\beta_\mu L$ to be

$$\begin{aligned}\beta_\mu L = \{&\mu(x<y)\wedge_\mu(x<z)\wedge_\mu(y<z)[\mu t_\mu y,\mu x_\mu z,\mu z_\mu x] :\\ &\mu(x<y)\wedge_\mu(x<z)[\mu t_\mu x,\mu x_\mu y,\mu y_\mu z,\mu z_\mu x] :\\ &\mu(x<y)[\mu t_\mu x,\mu x_\mu y,\mu y_\mu x] :\\ &\mu(y<z)\wedge_\mu(x<z)\wedge_\mu(x<y)[\mu t_\mu x,\mu x_\mu z,\mu z_\mu x] :\\ &\mu(y<z)\wedge_\mu(x<z)[\mu t_\mu x,\mu x_\mu z,\mu y_\mu x,\mu z_\mu y] :\\ &\mu(y<z)[\mu t_\mu y,\mu y_\mu z,\mu z_\mu y] : T\Lambda \quad \}\end{aligned}$$

Therefore, the minimal static sequence corresponding to the above program is

$$T = S_\mu[\mu x_\mu a,\mu y_\mu b,\mu z_\mu c;\beta_\mu L]$$

ie. $T = S_\mu[\{_\mu(a<b)\wedge_\mu(a<c)\wedge_\mu(b<c)[_\mu t_\mu b,{}_\mu x_\mu c,{}_\mu y_\mu b,{}_\mu z_\mu a]$:

$\quad {}_\mu(a<b)\wedge_\mu(a<c)[_\mu t_\mu a,{}_\mu x_\mu b,{}_\mu y_\mu c,{}_\mu z_\mu a]$:

$\quad {}_\mu(a<b)[_\mu t_\mu a,{}_\mu x_\mu b,{}_\mu y_\mu a,{}_\mu z_\mu c]$:

$\quad {}_\mu(b<c)\wedge_\mu(a<c)\wedge_\mu(a<b)[_\mu t_\mu a,{}_\mu x_\mu c,{}_\mu y_\mu b,{}_\mu z_\mu a]$:

$\quad {}_\mu(b<c)\wedge_\mu(a<c)[_\mu t_\mu a,{}_\mu x_\mu c,{}_\mu y_\mu a,{}_\mu z_\mu b]$:

$\quad {}_\mu(b<c)[_\mu t_\mu b,{}_\mu x_\mu a,{}_\mu y_\mu c,{}_\mu z_\mu b]$:

$\quad T[_\mu x_\mu a,{}_\mu y_\mu b,{}_\mu z_\mu c]\ \}$

That is, the program sorts three numbers in descending order. The minimal sequence indicates the final values of x, y, z with respect to all the possible values of the data a, b, c.

From what we have seen so far, it is apparent that although it is theoretically possible to transform a dynamic sequence representing a program with transfers of control into its minimal form, the task is by no means trivial.

§6.F Blocks

In our discussion so far, in order not to confuse the issue at hand, we have avoided the concept of block structures completely. Our programs consist of sequences of statements only. Therefore, we require statement brackets merely to indicate the range of *if* ... *then* ... *else* ... constructs. These statement brackets are represented by (and) .

We now turn our attention to the concepts involved in setting up the S-model of a block.

Augmentation and Deletion

In Chapter IV, X is defined to be a set whose elements are an alternate sequence of corresponding names and values, ie.

$$X = \{n_1, v_1, n_2, v_2, \cdots\}$$

where $n_i \in N$, $v_i \in V$ and $\varkappa n_i = v_i$.

Let μ be a subset of X .

Definition 6.16

An augmentation of a, possibly empty, set of names $M = \{m_1, m_2, \ldots, m_n\}$ to $\mu \subset X$ denoted by

$$\mu \oplus < m_1, m_2, \cdots, m_n >$$

is defined to be a concatenation of μ with the set $\{m_1, \omega, m_2, \omega, \ldots, m_n, \omega\}$, $\omega \in V$, which will be written as

$$\{ \mu \,/\, m_1, \omega, m_2, \omega, \cdots, m_n, \omega \}$$

eg. if $\mu = \{a, 1, b, 2\}$, then

$$\mu \oplus <c, d> = \{a, 1, b, 2 \,/\, c, \omega, d, \omega\}$$

Definition 6.17

A deletion of a set $M = \{m_1, m_2, \ldots, m_n\}$ from μ denoted by

$$\mu \ominus < m_1, m_2, \cdots, m_n >$$

is defined as

$$\mu \setminus \{m_1, v_1, m_2, v_2, \cdots, m_n, v_n\}$$

where $\setminus$ denotes relative complement and $\mu m_i = v_i \quad \forall\, i=1,2,\ldots,n$.

$\mu\ominus< m_1,m_2, \ldots ,m_n >$ is not defined if $\{m_1,m_2,\ldots,m_n\} \not\subset \mu$.

eg. if $\mu = \{a,1,b,2,c,3,d,4\}$, then

$\mu\ominus\langle a,c\rangle = \{b,2,d,4\}$

Definition 6.18

$\mu\ominus$ represents the deletion in μ of all names (and their corresponding values) following the last marker / .

eg. for $\mu = \{a,1,b,2 \,/\, c,\omega \,/\, a,4,d,5,c,\omega\}$

$\mu\ominus = \{a,1,b,2 \,/\, c,\omega\}$.

Definition 6.19

An n-level deletion in μ , $\mu\ominus^n$, denotes the deletion in μ of all names (and their corresponding values) following the last n markers.

eg. for $\mu = \{a,1,b,2 \,/\, c,\omega \,/\, a,4,d,5,c,\omega \,/\, b,\omega,d,6\}$

$\mu\ominus^3 = \{a,1,b,2\}$.

In the sequel, a block will be regarded as consisting of the keyword begin, a list of variables declared preceded by the word dec, a sequence of statements and the keyword end , ie.

begin dec a,b,c,...

X

end

We model the above block by the following sequence of S-expressions

$S_{\mu\mu\mu} \oplus < a,b,c,\ldots >$

$S_{\mu\beta X}$

$S_{\mu\mu\mu} \ominus$ (6.20)

A compound statement of the form

<u>begin</u>

X

<u>end</u>

is considered as a block with an empty set of variables being declared, and is thus modelled by

$S_{\mu\mu\mu} \oplus$

$S_{\mu\beta X}$

$S_{\mu\mu\mu} \ominus$ (6.21)

The dynamic sequences for (6.20) and (6.21) are

$S_{\mu}[\mu\mu \oplus <a,b,c,\ldots>;\ \beta X\ ;\ \mu\mu \ominus]$

and $S_{\mu}[\mu\mu \oplus\ ;\ \beta X\ ;\ \mu\mu \ominus]$.

For convenience we denote the terms

$\mu\mu \oplus <\ldots>$, $\mu\mu \oplus$ and $\mu\mu \ominus$

simply by

$\oplus <\ldots>$, $\oplus$, and $\ominus$ respectively.

Thus (6.20) and (6.21) will be represented as

$S_{\mu}[\ \oplus <a,b,c,\ldots>\ ;\ \beta X\ ;\ \ominus]$

and $S_{\mu}[\ \oplus\ ;\ \beta X\ ;\ \ominus]$.

Example

```
begin dec x,y,z
      x := a
      y := b
      if x<y then
                begin
                      z := x
                      x := y
                      y := z
                end-
end
```

The corresponding dynamic sequence is

$$S\mu[\oplus\langle x,y,z\rangle;\kappa x\kappa a;\kappa y\kappa b;\{\kappa(x<y)[\oplus;\kappa z\kappa x;\kappa x\kappa y;\kappa y\kappa z;\ominus]:T\Lambda\};\ominus]$$

Scope of Variables

The effect of augmentation as defined in definition 6.16 is to add a new set of variables to the existing variable space. This closely models the allocation of storage space as a result of a declaration. Similarly, deletion removes all variables in a block from the variable space and thus causes the variables to loose their values as on exit from a block through end. The scope of variables may be characterized by a more precise specification of names in the assignment model as follows:

At any stage in a program, the names x and y in an expression $S\mu\kappa x\kappa y$, representing an assignment statement, always refer to the x and y first encountered in the right-to-left search of the

current state of μ eg. if $\mu = \{a,1,b,2 \,/\, c,\omega,a,\omega\}$, the effect of $S_{\mu\kappa a\kappa b}$ is the assignment of the value 2 to the name a appearing after the marker / ie. $S_{\mu\kappa a\kappa b}$ yields $\{a,1,b,2 \,/\, c,\omega,a,2\}$.

To illustrate the working of the S-model of a block we shall look at the line-by-line model of a program and observe the change in the variable space at each stage.

```
begin dec x,y,z
      x := 6
      y := x-2
      begin dec x
            x := y-2
            begin dec y
                  y := x-2
                  z := y
            end
            x := z+4
            z := y
      end
      x := z+4
end                                                    (6.22)
```

Assuming that the above example constitutes the whole program, then the variable space μ at the start is empty.

S-sequence model	Variable space μ
	$\emptyset$
$S_{\mu\mu\mu} \oplus \langle x,y,z \rangle$	$\{x,\omega,y,\omega,z,\omega\}$
$S_{\mu\mu x_{\mu}} 6$	$\{x,6,y,\omega,z,\omega\}$
$S_{\mu\mu y_{\mu}} (x-2)$	$\{x,6,y,4,z,\omega\}$
$S_{\mu\mu\mu} \oplus \langle x \rangle$	$\{x,6,y,4,z,\omega \;/\; x,\omega\}$
$S_{\mu\mu x_{\mu}} (y-2)$	$\{x,6,y,4,z,\omega \;/\; x,2\}$
$S_{\mu\mu\mu} \oplus \langle y \rangle$	$\{x,6,y,4,z,\omega \;/\; x,2 \;/\; y,\omega\}$
$S_{\mu\mu y_{\mu}} (x-2)$	$\{x,6,y,4,z,\omega \;/\; x,2 \;/\; y,0\}$
$S_{\mu\mu z_{\mu} y}$	$\{x,6,y,4,z,0 \;/\; x,2 \;/\; y,0\}$
$S_{\mu\mu\mu} \ominus$	$\{x,6,y,4,z,0 \;/\; x,2\}$
$S_{\mu\mu x_{\mu}} (z+4)$	$\{x,6,y,4,z,0 \;/\; x,4\}$
$S_{\mu\mu z_{\mu} y}$	$\{x,6,y,4,z,4 \;/\; x,4\}$
$S_{\mu\mu\mu} \ominus$	$\{x,6,y,4,z,4\}$
$S_{\mu\mu x_{\mu}} (z+4)$	$\{x,8,y,4,z,4\}$
$S_{\mu\mu\mu} \ominus$	$\emptyset$

Irregular Exit from a Block

As stated earlier, our model of a high-level language is based upon features of Algol-60. The restriction imposed upon inter-block transfers of control in Algol-60 are such that it is possible to leave a block before the end but not to jump into the middle of a block from outside that block, eg.

```
begin
    A
    begin
        B
        goto 1
```

```
        C
        end
        D
    1:  E
        F
end
```

We shall therefore briefly include irregular exit from a block in our discussion of the S-model. This is not entirely essential since a method has been outlined [6] by which multiple entry/exit blocks may be transformed into single entry/exit blocks. However, it would be more convenient to have an S-model of an irregular exit especially when it involves only a slight modification of the model of the goto statement.

Normally, a statement goto 1 is modelled by $\mu 1$. In the case of an irregular exit, such a statement causes the control to leave a block thus causing the release of storage for all the variables declared in that block. That is, the result is the same as passing through end before the statement labelled 1 is executed.

Therefore, a goto statement which causes an irregular exit from a block will be marked by * to differentiate it from a normal goto. That is, a goto which effects an exit through n levels of nesting will be written as

goto $*^n$ 1

and modelled by $\ominus^n ; \mu 1$

where $\ominus^n$ denotes the n-level deletion as defined in 6.19.

This implies that a check must be made to detect the irregular exits. In simple cases the successor to the statement goto l is found by first searching the block in which the statement occurs, and then the outer block until a statement with label l is found, keeping count of the depth to give n . Techniques have been devised for more complicated cases such as for blocks imbedding in recursive procedures, these are given elsewhere eg. [17].

Note goto $*^1$l will be written as goto* l

```
eg.     begin
            A
            begin
        l1: B
            if p then goto*12
            C
            goto l1
            end
        l2: E
        end
```

Here, goto l1 is modelled normally as $\varkappa$l1 but goto*12 causes an irregular exit from the inside block and thus the conditional statement corresponds to the sequence

$$\{\ \varkappa p\ [\ \ominus\ ;\ \beta\varkappa 12]\ :\ T\Lambda\ \}$$

Transformation of Dynamic Sequence with Blocks

The transformation of dynamic sequences containing blocks is basically the same as that discussed earlier. The presence of augmentation and deletion merely helps to exhibit the scope of the variables and does not affect the actual transformation and reduction in any way.

Broadly speaking, we view such a sequence as a series of subsequences separated by either $\oplus\langle\cdot\cdot\rangle$ or $\ominus$. Each subsequence is first transformed and reduced independently, then the effect of the preceding subsequences is taken into consideration as explained in §6.A. The only difference is that at the second stage, transformation of dynamic terms to static depends largely on the scope of variables which is indicated by the list of variables following augmentation, eg. From the program given in (6.22), the corresponding dynamic sequence is

$$
\begin{aligned}
S_{\mu}[\, \oplus\langle x,y,z\rangle &: \nu x_{\nu} 6;\ \nu y_{\nu}(x-2);\\
\oplus\langle x\rangle &: \nu x_{\nu}(y-2);\\
\oplus\langle y\rangle &;\ \nu y_{\nu}(x-2);\ \nu z_{\nu} y;\ \ominus;\\
&\quad \nu x_{\nu}(z+4);\ \nu z_{\nu} y;\ \ominus;\\
&\quad \nu x_{\nu}(z+4);\ \ominus \qquad]
\end{aligned}
$$

This sequence consists of 5 subsequences. By convention, we shall regard $\oplus$ as the beginning of a subsequence and $\ominus$ as marking the end of a subsequence. Hence, a subsequence may have any of the following forms:-

$\oplus\langle\,\rangle$;

$\oplus\langle\,\rangle$; terms ;

$\oplus\langle\,\rangle$; terms ; $\ominus$;

terms ;

terms ; $\ominus$;

$\ominus$;

eg. a program of the form

```
begin
        .
        .
        .
      begin
        .
        .
        .
      end
        .
        .
      begin
         begin
           .
           .
           .
         end
      end
        .
        .
        .
end
```

corresponds to a dynamic sequence of the form

$S_\mu[$ ⊕ < >; terms ;
⊕ < >; terms ; ⊖ ;
terms ;
⊕ < >;
⊕ < >; terms ; ⊖ ;
⊖ ;
terms ; ⊖]

Note that a subsequence which contains both ⊕ and ⊖ represents a complete block.

Each subsequence is now transformed and reduced whenever possible to give

$$
\begin{aligned}
S\mu[\ &\oplus\langle x,y,z\rangle\ ,\ \nu x\mu 6,\nu y\mu 4;\\
&\oplus\langle x\rangle\ ,\ \nu x\mu(y-2);\\
&\oplus\langle y\rangle\ ,\ \nu y\mu(x-2),\nu z\mu(x-2),\ominus;\\
&\qquad\nu x\mu(z+4),\nu z\mu y,\ominus;\\
&\qquad\nu x\mu(z+4),\ominus\qquad\ ]
\end{aligned}
\tag{6.23}
$$

The next stage is to transform each subsequence with respect to its predecessors. The process is carried out as follows:

Starting with the second subsequence, i=2

a. For each νy which is subject to alteration[†] in subsequence i , if there exists a term $\nu y\mu z$ for some z in the preceding subsequences then replace νy in subsequence i by νz. If there exists several such terms, then the replacement is made according to the term in the "closest"[††] predecessor.

b. Reduce subsequence i to the minimal possible form.

c. If subsequence i contains neither $\oplus$ nor $\ominus$, then incorporate subsequence i into its immediate predecessor.

d. If subsequence i ends with $\ominus$, then find the corresponding augmentation $\oplus\langle\ \rangle$ which will be at the beginning of either subsequence i itself or its current immediate predecessor. Suppose $\oplus\langle A\rangle$, where A is a set of names, is found at the beginning of subsequence j , say, then

---

† i.e. being the second component of a term or part of the propositional expression in a conditional sequence.

†† i.e. the term such that no other term satisfying these conditions intervenes

d1. Consider the terms between $\oplus\langle A\rangle$ and $\ominus$ as a single subsequence and reduce it to the minimal form.

d2. If $\exists$ a term $\nu x_\nu y$ between $\oplus\langle A\rangle$ and $\ominus$ such that $x \notin A$, then transfer the term to the end of subsequence j-1 and reduce subsequence j-1 whenever possible. If such a term is part of a conditional sequence, then together with the corresponding propositional expression it is incorporated into subsequence j-1 [(6.24)].

d3. Remove what remains between $\oplus\langle A\rangle$ and $\ominus$ inclusive from the sequence.

e. Consider the next subsequence ie. increase i by 1 and repeat a to d until the sequence is exhausted.

From (6.23),

$$S_\mu[\ \oplus\langle x,y,z\rangle,\nu x_\nu 6,\nu y_\nu 4;$$
$$\oplus\langle x\rangle,\nu x_\nu (y-2);$$
$$\oplus\langle y\rangle,\nu y_\nu (x-2),\nu z_\nu (x-2),\ominus;$$
$$\nu x_\nu (z+4),\nu z_\nu y,\ominus;$$
$$\nu x_\nu (z+4),\ominus\ ]$$

i=2: $$S_\mu[\ \oplus\langle x,y,z\rangle,\nu x_\nu 6,\nu y_\nu 4;$$
$$\oplus\langle x\rangle,\nu x_\nu 2;$$

i=3: $$S_\mu[\ \oplus\langle x,y,z\rangle,\nu x_\nu 6,\nu y_\nu 4;$$
$$\oplus\langle x\rangle,\nu x_\nu 2;$$
$$\oplus\langle y\rangle,\nu y_\nu 0,\nu z_\nu 0,\ominus;$$
$$\Rightarrow\ S_\mu[\ \oplus\langle x,y,z\rangle,\nu x_\nu 6,\nu y_\nu 4;$$
$$\oplus\langle x\rangle,\nu x_\nu 2,\nu z_\nu 0;$$
$$\underbrace{\oplus\langle y\rangle,\nu y_\nu 0,\ominus;}_{\text{remove}}$$

i=4: $S\mu[\oplus \langle x,y,z\rangle, \nu x_\nu 6, \nu y_\nu 4;$

$\oplus \langle x\rangle, \nu x_\nu 2, \nu z_\nu 0;$

$\nu x_\nu 4, \nu z_\nu 4, \ominus ;$

$\Rightarrow \quad S\mu[\oplus \langle x,y,z\rangle, \nu x_\nu 6, \nu y_\nu 4;$

$\oplus \langle x\rangle, \nu x_\nu 4, \nu z_\nu 4, \ominus ;$

$\Rightarrow \quad S\mu[\oplus \langle x,y,z\rangle, \nu x_\nu 6, \nu y_\nu 4, \nu z_\nu 4;$

$\underbrace{\oplus \langle x\rangle, \nu x_\nu 4, \ominus ;}_{\text{remove}}$

i=5: $S\mu[\oplus \langle x,y,z\rangle, \nu x_\nu 6, \nu y_\nu 4, \nu z_\nu 4;$

$\nu x_\nu 8, \ominus \quad]$

$\Rightarrow \quad S\mu[\underbrace{\oplus \langle x,y,z\rangle, \nu x_\nu 8, \nu y_\nu 4, \nu z_\nu 4, \ominus}_{\text{remove}}]$

Note

1. The scope of variables is taken into account in (a) by making a replacement of terms according to the term in the closest preceding subsequence whenever there is a choice. This ensures that the scope of a variable does not include an inner block in which the same name is declared again.

2. From d2 , as an example for the case of conditional sequences, we might have the following:-

$S\mu[\oplus \langle x,\ldots\rangle, \ldots ;$

$\oplus \langle a\rangle, \{\nu(\alpha>0)[\nu a_\nu \alpha, \nu x_\nu \beta] : T[\nu x_\nu(-\alpha)]\};$

$\ldots \quad]$

which, after the transfer, becomes

$$
\begin{array}{l}
S\mu[\oplus\langle x,\ldots\rangle,\ldots;\ \{\varkappa(\alpha>0)[\varkappa x\varkappa\beta] : T[\varkappa x\varkappa(-\alpha)]\};\\
\quad \oplus\langle a\rangle,\ \{\varkappa(\alpha>0)[\varkappa a\varkappa\alpha] : T[\ \Lambda\ ]\};\\
\qquad\qquad \ldots \qquad\qquad\qquad]
\end{array}
\tag{6.24}
$$

3. In d3 , we remove each minimal subsequence which constitutes a complete block from the sequence. This is beacuase once we exit from a block, all the variables declared in it are lost and hence have no further effect on the rest of the sequence.

§6.G Initialization

So far we have considered only one form of initialization ie. as used in an assignment statement of the form a := b . This corresponds to what is known as initialization by value in CPL [2]. Another type of initialization which we need to consider is initialization by substitution characterized by the '≡' sign. According to [2]

"Initialization by substitution denotes that a variable is equivalent to the right-hand side expression, which must be evaluated afresh each time the variable is used."

To produce the required effect in terms of S-expressions, initialization by substitution of the form

$$a \equiv b$$

is represented by the expression

$$S\mu ab\ .$$

In contrast with the model $S_{\mu\kappa a_\kappa b}$ for initialization by value in which the value of a and b are involved in the operation, $S_{\mu ab}$ causes the name a to be replaced by the name b . Thus, the effect of an expression $S_{\mu ab}$ on its successors is different to that of $S_{\mu\kappa a_\kappa b}$ which we consider in Chapter IV. We shall, therefore, discuss the transformation associated with name replacement (which, from now on, shall be known as 'name transformation' as opposed to 'value transformation' described in Chapter IV) and develop a process by which we may obtain the minimal form of a dynamic sequence involving both types of replacement.

Name Transformation

Consider a dynamic sequence

$$S_\mu[\ \ldots;\ ab\ ;\ \ldots;\ \kappa x_\kappa a\ ;\ \ldots\ ;\ \kappa a_\kappa y\ ;\ \ldots\ ] \qquad (6.25)$$

where the intermediate terms between ab and $\kappa a_\kappa y$ do not contain the name a .

Initialization by substitution denotes that a variable is equivalent to the right-hand side implying that both the names and the values are involved. Hence in a dynamic sequence containing a name replacement, say ab , all occurences of a must be replaced by b ie. the transformation affects both the first and the second component of the terms.

That is, in a dynamic sequence of n terms, if xy appears as the i^{th} term, then all occurences of x in the terms $(i+1)^{th}$ to n^{th} are replaced by y . (6.26)

eg. the sequence (7.1.1) after the name transformation gives

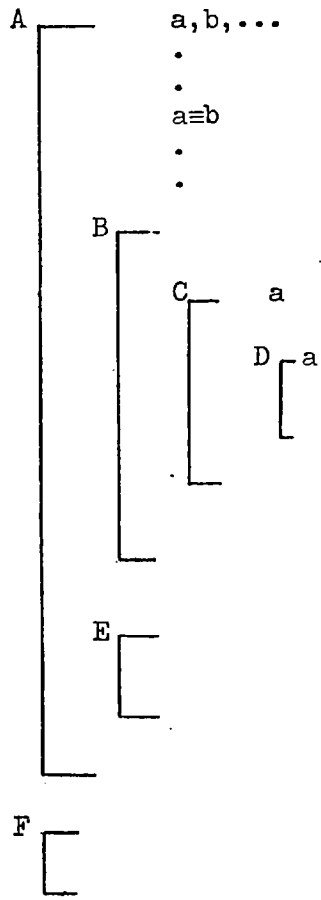
$$S_\mu[\ldots\ ;\ ab\ ;\ldots;\ \mu x_\mu b\ ;\ldots;\ \mu b_\mu y\ ;\ldots\ ]$$

In effect, name transformation of a sequence such as (6.25) is as though a subsequence headed by the term ab is detached from the whole sequence and the transformation (6.26) is applied to it. In the simple example given, the subsequence concerned consists of all the terms following ab up to the end of the sequence. In a sequence with block structure, the range of transformation may not extend so far. The subject will now be discussed in detail.

Range of Name Transformation

The range of name transformation in a sequence with block structure is indicated by the scope of variables in the program.

Consider a program with the following block structure:-



(6.27)

where A,B,C,D,E and F denote the blocks and the variable a is declared at the beginning of blocks A, C and D. The scope of the variable a in block A is the whole of block A except block C. Note that block D is part of block C.

Thus the name transformation following a≡b has the effect on all the terms up to the end of block A except those in block C. The range indicated as such may be identified as follows:

In terms of an S-sequence, a program with block structure as shown in (6.27) has the following form:-

$$
\begin{array}{lll}
 & \overset{A}{} & \\
S_\mu[& \oplus\langle a,b,\ldots\rangle\ ;\ \ldots\ ;\ ab\ ;\ \ldots\ ; & \\
 & \overset{B}{} & \\
 & \oplus\langle\ \ldots\ \rangle\ ;\ \ldots\ ; & \\
 & \overset{C}{} & \\
 & \oplus\langle a,\ldots\ \rangle\ ;\ \ldots\ ; & \\
 & \overset{D}{} & \\
 & \oplus\langle a,\ldots\ \rangle\ ;\ \ldots\ ; & ;\ominus\ ; \\
 & \qquad\qquad\ldots\ ; & ;\ominus\ ; \\
 & \qquad\qquad\ldots\ ; & ;\ominus\ ; \\
 & \quad\ldots & \\
 & \overset{E}{} & \\
 & \oplus\langle\ \ldots\ \rangle\ ;\ \ldots & ;\ominus\ ; \\
 & \qquad\qquad\ldots & ;\ominus\ ; \\
 & \quad\ldots & \\
 & \overset{F}{} & \\
 & \oplus\langle\ \ldots\ \rangle\ ;\ \ldots & ;\ominus\ ]
\end{array}
$$

with the beginning and the end of each block being indicated by $\oplus$ and $\ominus$ respectively.

The term ab appears in the sequence indicating that name replacement must be carried out. The range of replacement is found by setting up

two counters r and s which are initialized to 1 and 0 respectively.

Consider the sequence term-by-term starting from the term ab .

1. If the next term is an augmentation $\oplus$<...> then

   1.1 If s>0, then set s=s+1 .

   1.2 If <...> contains a , then set s=s+1 else set r=r+1 .

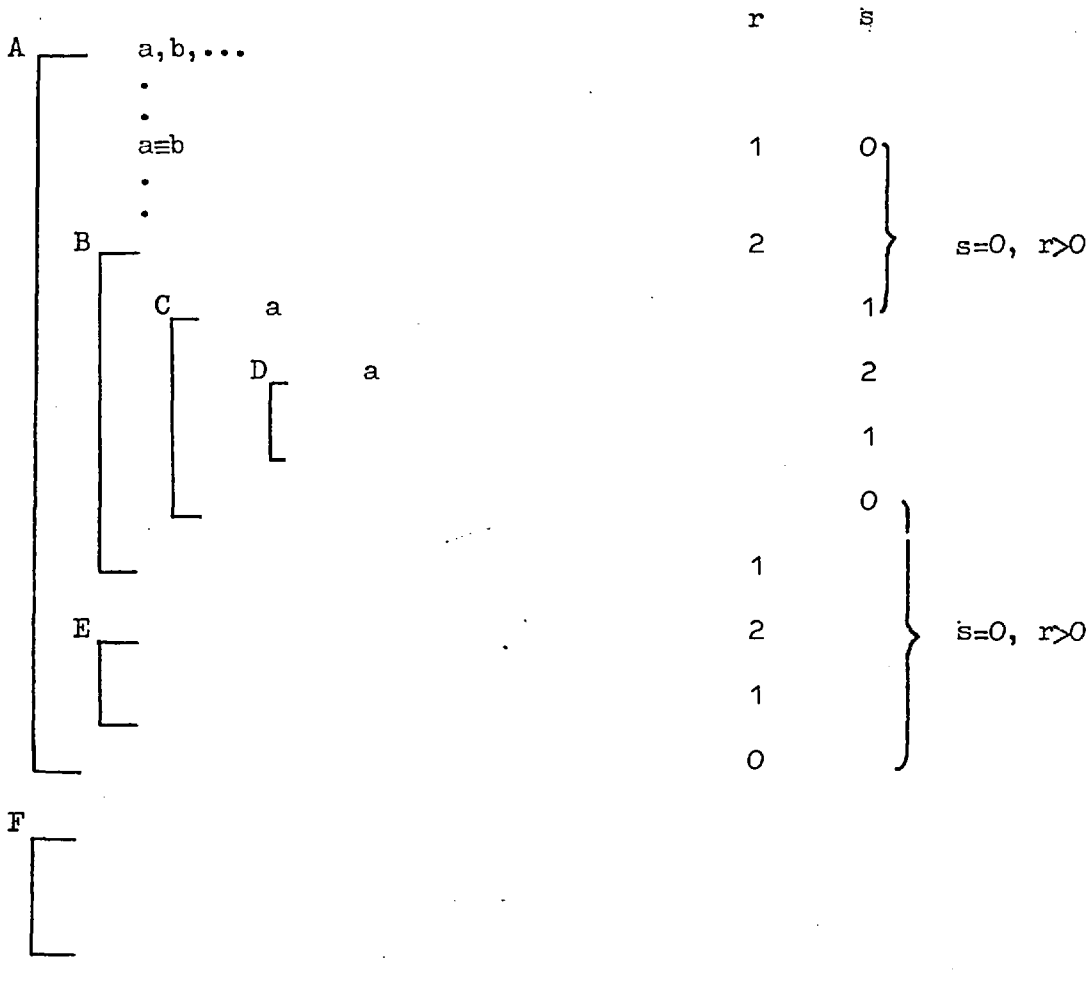
2. If the next term is a deletion $\ominus$, then

   2.1 If s>0, set s=s-1 else r=r-1 .

3. If the next term contains a and s=0, r>0 then replace a by b else no change is made.

4. r≤0 indicates the end of the range.

For example, from (6.27), the changes in r and s are:-



thus the subsequence to be considered for name replacement contains the terms appearing in the range $s=0$, $r\neq 0$ ie. from the term ab up to the end of block A excluding block C.

Note

If the sequence contains n-level deletion $\ominus^n$, then (2) is modified to read:-

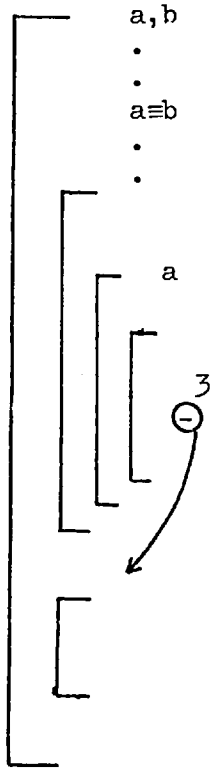
2* If the next term is a deletion $\ominus^n$, then

2.1* If $s>0$, then

a) set $s=s-n$. If $s<0$ then set $r=r+s$ and $s=0$

else

b) $r=r-n$.



Reduction of Static Sequence with Name Replacement

A static sequence containing a term of the form ab indicating name replacement may be reduced to its minimal form according to the following rules:-

6.28 A sequence of the form

$$S_\mu[\ \ldots\ ,\ \nu x_\nu \alpha\ ,\ \ldots\ ,\ xy\ ,\ \ldots\ ]$$

in which the terms between $\nu x_\nu \alpha$ and xy do not include $\oplus$ and $\ominus$ is reduced to

$$S_\mu[\ \ldots\ ,\ xy\ ,\ \ldots]$$

6.29 A sequence of the form

$$S_\mu[\ \ldots\ ,\ xy\ ,\ \ldots\ ,\ \nu y_\nu \beta\ ,\ \ldots\ ]$$

where $\oplus$ and $\ominus$ do not appear between xy and $\nu y_\nu \beta$ is replaced by

$$S_\mu[\ \ldots\ ,\ \nu x_\nu \beta\ ,\ \ldots\ ,\ \nu y_\nu \beta\ ,\ \ldots]$$

These rules are justified by the fact that once a variable x has been identified with a variable y , these two variables share the same value.

Name and Value Transformation

The transformation to static of a dynamic sequence containing both name and value replacement closely resembles the process described for

the transformation of dynamic sequences with blocks with a few modifications.

1. Consider the sequence as consisting of subsequences separated by $\oplus$ and $\ominus$ as described in the case of blocks except, in addition, each term of the form ab also marks the beginning of a subsequence eg. a sequence

$$S_\mu[\oplus\langle a,b\rangle;{}_\mu a_\mu\alpha;{}_\mu b_\mu\beta;\ \oplus\langle a,c\rangle;{}_\mu c_\mu a;ab;{}_\mu a_\mu(a+c);\ \ominus;\ \ominus]$$

consists of the following subsequences

$$S_\mu[\oplus\langle a,b\rangle\ ;{}_\mu a_\mu\alpha;{}_\mu b_\mu\beta;$$
$$\oplus\langle a,c\rangle\ ;{}_\mu c_\mu a;$$
$$ab;{}_\mu a_\mu(a+c);\ \ominus;$$
$$\ominus\ ]$$

2. Apply the value transformation to each subsequence independently and reduce using theorems 4.18 and 4.19.

3. For each subsequence starting with a term of the form ab , find its range of transformation and apply the name transformation [6.26] within that range. Apply theorems 4.18 and 4.19 to reduce the subsequences affected by the transformation whenever possible.

4. Starting with i=2 , apply the value transformation to subsequence i relative to its predecessors [(a), page 173].

5. Reduce subsequence i using theorems 4.18 and 4.19.

6. If subsequence i contains neither $\oplus$ nor $\ominus$, then incorporate it into its immediate predecessor.

7. If subsequence i ends with ⊖, then find the corresponding augmentation ⊕<...> which will be at the beginning of i itself or its current immediate predecessor. Suppose ⊕<A>, where A is a set of names, is found at the beginning of subsequence j, say, then

7a. Consider the terms between ⊕<A> and ⊖ as a single subsequence. Apply value transformation and reduce it to the minimal form using theorems 4.18 and 4.19, rules 6.28 and 6.29 respectively.

7b. Remove all the terms containing variables which are global to the block to the next outer block [(d2), page 174].

7c. Remove what remains in the block between ⊕<A> and ⊖ inclusive from the sequence.

8. Consider the next subsequence ie. increase i by 1 and repeat steps 4 to 7 until the sequence is exhausted.

Example

$$S\mu[\ \oplus\langle a,b\rangle;\aleph a\aleph\alpha;\aleph b\aleph\beta;$$
$$\oplus\langle x,y,z\rangle;$$
$$xa;$$
$$yb;$$
$$\oplus\ ;\aleph z\aleph x;\aleph x\aleph y;\aleph y\aleph z;\ \ominus\ ;$$
$$\ominus\ ;$$
$$\ominus\ ]$$

Step 2 =>

$$S\mu[\oplus\langle a,b\rangle,\eta a\eta\alpha,\eta b\eta\beta;$$
$$\oplus\langle x,y,z\rangle;$$
$$xa;$$
$$yb;$$
$$\oplus,\ \eta z\eta x,\eta x\eta y,\eta y\eta x,\ominus;$$
$$\ominus;$$
$$\ominus;\ ]$$

Step 3 =>

$$S\mu[\oplus\langle a,b\rangle,\eta a\eta\alpha,\eta b\eta\beta;$$
$$\oplus\langle x,y,z\rangle;$$
$$xa;$$
$$yb;$$
$$\oplus,\eta z\eta a,\eta a\eta b,\eta b\eta a,\ominus;$$
$$\ominus;$$
$$\ominus\ ]$$

Step 4 -> 7: i=2

$$\Rightarrow\quad S\mu[\oplus\langle a,b\rangle\ ,\ \eta a\eta\alpha,\eta b\eta\beta,$$
$$\oplus\langle x,y,z\rangle;$$

i=3: $$\Rightarrow\quad S\mu[\oplus\langle a,b\rangle\ ,\ \eta a\eta\alpha,\eta b\eta\beta;$$
$$\oplus\langle x,y,z\rangle\ ,\ xa;$$

i=4: $$\Rightarrow\quad S\mu[\oplus\langle a,b\rangle\ ,\ \eta a\eta\alpha,\eta b\eta\beta;$$
$$\oplus\langle x,y,z\rangle\ ,\ xa,yb;$$

i=5: $\Rightarrow$ $S\mu[\oplus\langle a,b\rangle , \varkappa a\varkappa\alpha,\varkappa b\varkappa\beta;$

$\oplus\langle x,y,z\rangle , xa,yb,$

$\oplus ,\varkappa z\varkappa\alpha,\varkappa a\varkappa\beta,\varkappa b\varkappa\alpha, \ominus ;$

$\ominus ;$

$\ominus\ ]$

$\Rightarrow$ $S\mu[\oplus\langle a,b\rangle , \varkappa a\varkappa\alpha,\varkappa b\varkappa\beta;$

$\oplus\langle x,y,z\rangle , xa,yb,\varkappa z\varkappa\alpha,\varkappa a\varkappa\beta,\varkappa b\varkappa\alpha;$

i=6 $\Rightarrow$ $S\mu[\oplus\langle a,b\rangle , \varkappa a\varkappa\alpha,\varkappa b\varkappa\beta;$

$\oplus\langle x,y,z\rangle , xa,yb,\varkappa z\varkappa\alpha,\varkappa a\varkappa\beta,\varkappa b\varkappa\alpha, \ominus ;$

$\Rightarrow$ $S\mu[\oplus\langle a,b\rangle , \varkappa a\varkappa\alpha,\varkappa b\varkappa\beta;$

$\oplus\langle x,y,z\rangle , \varkappa x\varkappa\beta,\varkappa y\varkappa\alpha,\varkappa z\varkappa\alpha,\varkappa a\varkappa\beta,\varkappa b\varkappa\alpha, \ominus ;$

$\Rightarrow$ $S\mu[\oplus\langle a,b\rangle , \varkappa a\varkappa\alpha,\varkappa b\varkappa\beta,\varkappa a\varkappa\beta,\varkappa b\varkappa\alpha;$

$\oplus\langle x,y,z\rangle , \varkappa x\varkappa\beta,\varkappa y\varkappa\alpha,\varkappa z\varkappa\alpha, \ominus ;$

$\Rightarrow$ $S\mu[\oplus\langle a,b\rangle , \varkappa a\varkappa\alpha,\varkappa b\varkappa\beta,\varkappa a\varkappa\beta,\varkappa b\varkappa\alpha;$

i=7 $\Rightarrow$ $S\mu[\oplus\langle a,b\rangle , \varkappa a\varkappa\alpha,\varkappa b\varkappa\beta,\varkappa a\varkappa\beta,\varkappa b\varkappa\alpha, \ominus]$

$\Rightarrow$ $S\mu[\oplus\langle a,b\rangle , \varkappa a\varkappa\beta,\varkappa b\varkappa\alpha, \ominus]$

§6.H Procedures

In establishing a model for an Algol-60 procedure, we are concerned with two aspects ie. procedure declaration and procedure statements. Procedure declaration specifies, only partially, a sequence of statements to be executed at the time when the procedure is called. This specification is incomplete because

(1) the procedure declaration contains formal parameters not the actual ones

(2) it may contain some identifiers which are global to the procedures. Such identifiers have no meaning unless the corresponding procedure statement appears in the range in which these identifiers are defined.

On the other hand, a procedure statement has the effect of assigning values to the formal parameters quoted in the value part of the procedure declaration, replacing all other parameters by the corresponding actual parameters, inserting the modified procedure body in place of the procedure statement and executing.

In modelling procedures, we shall consider procedure declarations and procedure statements in parallel. Moreover we maintain that a procedure definition only has meaning through its 'potential for incarnation'. Thus our model will be not an exact translation of the procedure definition, but rather that definition suitably modified by the need for the passing of actual parameters at the time of call ie. it is a specification of what has to be inserted in a procedure statement. This does not, of course, preclude the fact that the body of a procedure can be reduced to minimal form prior to any of its incarnations.

Consider an Algol-60 procedure of the form

$$\begin{aligned}
&\underline{\text{procedure}}\ \text{name}(a_1,\ldots,a_m,b_1,\ldots,b_n)\\
&\underline{\text{value}}\ a_1,\ldots,a_m\\
&\underline{\text{specification}}\ a_1,\ldots,a_m,b_1,\ldots,b_n\\
&\qquad \text{body}
\end{aligned} \tag{6.30}$$

Algol-60 Report [30] states that a procedure body always acts like a block whether it has the form of one or not ie. (6.30) behaves as:

```
procedure name(a1,...,am,b1,...,bn)
value a1,...,am
specification a1,...,am,b1,...,bn
begin
    body
end                                          (6.31)
```

In our model, we regard a procedure declaration as a block labelled by the procedure identifier with all the formal parameters being declared at the beginning of the block ie. (6.31) is regarded as

```
name: begin dec a1,...,am,b1,...,bn
      value a1,...,am
      begin
          body
      end
      end
```

When a procedure is called, all formal parameters called by value are assigned the values of the corresponding actual parameters, these assignments being considered as being performed explicitly before entering the procedure body. Parameters called by name are replaced throughout the procedure body by the corresponding actual parameters.

Now, value assignment corresponds to initialization by value and name replacement is the sameas initialization by substitution [§6.G], thus at the time of call the above block has the modified form:

```
name:  begin dec a1,...,am,b1,...,bn
           a1:=( )
            :
           am:=( )
           b1 ≡( )
            :
           bn ≡( )
           begin
               body
           end
       end                                          (6.32)
```

where the right-hand side are filled with the corresponding actual parameters.

eg.

```
procedure update(x,y)
value y
specification x,y
x := x+y
```

is modified to give the following block:-

```
update: begin dec x,y
            y :=( )
            x ≡( )
            begin
                x := x+y
            end
        end
```

The modification for a type procedure is similar to that already described for the non-type procedure as far as the block structure, the value assignment and the name replacement are concerned. However, as the body of a type procedure must contain at least one explicit assignment statement with the procedure identifier in a left part, we must add the procedure identifier to the list of variables being declared at the beginning of the procedure block in our model.

eg.
```
dec procedure add(x,y)
    value x,y
    specification x,y
    add := x+y
```

corresponds to the following block:-

```
add:  begin dec x,y,add
          x := ( )
          y := ( )
          begin
              add := x+y
          end
      end                                    (6.33)
```

Now, suppose a call of the type procedure occurs in the following program:

```
begin dec a,b,c
      a := 1
      b := 2
      c := add(a,b)
end                                          (6.34)
```

This has the effect of inserting block (6.33) in place of statement 'c := add(a,b)'. Bearing in mind that the value of 'add(a,b)' is the last value assigned to the procedure identifier 'add', the result of (6.34) is the following sequence:-

```
begin dec a,b,c
      a := 1
      b := 2
      begin dec x,y,add
            x := a
            y := b
            begin
                  add := x+y
            end
            c := add
      end
end
```

That is, the block inserted in place of 'c := add(a,b)' is

```
add: begin dec x,y,add
            x := a
            y := b
            begin
                  add := x+y
            end
            c := add
      end
```

Thus, the type procedure of the form

```
dec procedure name(a1,...,am,b1,...,bn)
    value a1,...,am
    specification a1,...,am,b1,...,bn
        body                                        (6.35)
```

has the modified form

```
name: begin dec a1,...,am,b1,...,bn,name
            a1 := ( )
               :
            am := ( )
            b1 ≡ ( )
               :
            bn ≡ ( )
            begin
                body
            end
            ( ) := name
      end                                           (6.36)
```

where the right-hand sides are filled with the corresponding actual parameters and the left-hand side of '( ) := name' is completed with the identifier on the left-hand side of the procedure statement.

Procedures without parameters are considered in the same manner with the parameter list being empty.

Having thus specified the modification of a procedure declaration, we model procedure statements as follows:

A call to a non-type procedure of the form

$$\text{name}(x_1, \ldots, x_i)$$

is modelled by the expression

$$\varkappa\ \text{name}(x_1, \ldots, x_i)$$

where $\varkappa$name, being the value of a label, is given by the following dynamic sequence representing (6.32):

$$\begin{aligned}\varkappa\text{name} = S\mu[\ &\oplus\langle a_1,\ldots,a_m,b_1,\ldots,b_n\rangle;\varkappa a_1\varkappa();\ldots;\varkappa a_m\varkappa();\\ &\qquad b_1();\ldots;b_n();\\ &\oplus;\ \beta X\ ;\ominus;\\ &\ominus\ ]\end{aligned}$$

where X denotes the body of the dynamic S-sequence representing the procedure body.

Thus, the call ' $\varkappa$name$(x_1, \ldots, x_i)$ ' refers to the sequence

$$\begin{aligned}S\mu[\ &\oplus\langle a_1,\ldots,a_m,b_1,\ldots,b_n\rangle\ ;\varkappa a_1\varkappa x_1;\ldots;\varkappa a_m\varkappa x_m;\\ &\qquad b_1x_{m+1};\ldots;b_nx_i;\\ &\oplus;\ \beta X\ ;\ominus;\\ &\ominus\ ]\end{aligned} \tag{6.37}$$

where i=m+n and the correspondence between the actual and formal parameters is obtained by taking the entries of the two lists in the same order. (The formal parameter list appears as the augmentation list in our modified block).

Similarly, a call to a type procedure of the form

$$y := \text{name}(x_1, \cdots, x_i)$$

is modelled by the expression

$$S_{\mu\varkappa}y_{\varkappa}\text{name}(x_1, \cdots, x_i)$$

which is, from (6.36), the dynamic sequence

$$S\,[\,\oplus\langle a_1,\cdots a_m,b_1,\cdots b_n,\text{name}\rangle;\varkappa a_1\varkappa x_1;\cdots;\varkappa a_m\varkappa x_m;\; b_1x_{m+1};\cdots;b_nx_i;\; \oplus;\,\beta X\,;\ominus;\; \varkappa y_{\varkappa}\text{name};\ominus\,] \qquad (6.38)$$

A more general case in which the right-hand side of a call to a type procedure is an expression follows the same pattern ie. a call of the form

$$y := a\ \underline{op}\ \text{name}(x_1, \cdots, x_i)\ ,$$

where a is a name or a value and op denotes an arithmetic or a boolean operator, is modelled [§4.G] by

$$S_{\mu\varkappa}y_{\varkappa}(a\ \underline{op}\ \text{name}(x_1,\cdots,x_i))$$

which corresponds to the dynamic sequence

$$S_{\mu}[\,\oplus\langle a_1,\cdots,a_m,b_1,\cdots,b_n,\text{name}\rangle;\varkappa a_1\varkappa x_1;\cdots;\varkappa a_m\varkappa x_m;\; b_1x_{m+1};\cdots;b_nx_i;\; \oplus;\,\beta X\,;\ominus;\; \varkappa y_{\varkappa}(a\ \underline{op}\ \text{name})\,;\ominus\,] \qquad (6.39)$$

Example

Suppose a procedure 'update' is defined as

```
procedure update(x,y)
value y
specification x,y
x := x+y
```

and a call to update occurs in the following program

```
begin dec a,x
      a := α
      x := ∂
      update(a,x)
end
```

Then, the S-sequence for the procedure declaration is

$$\eta\text{update} = S\mu[\ \oplus\langle x,y\rangle\ ;\ \eta y_{\eta}();\ x()\ ;\ \oplus\ ;\ \eta x_{\eta}(x+y)\ ;\ \ominus\ ;\ \ominus\ ]$$

and the dynamic sequence representing the program is

$$S\mu[\ \oplus\langle a,x\rangle\ ;\ \eta a_{\eta}\alpha;\eta x_{\eta}\partial;\eta\text{update}(a,x);\ \ominus\ ]$$

which, on substitution of ηupdate(a,x) , gives

$$S\mu[\ \oplus\langle a,x\rangle\ ;\ \eta a_{\eta}\alpha;\eta x_{\eta}\partial;\ \oplus\langle x,y\rangle\ ;\ \eta y_{\eta}x;\ xa\ ;\ \oplus\ ;\ \eta x_{\eta}(x+y)\ ;\ \ominus\ ;\ \ominus\ ;\ \ominus\ ]$$

This sequence can then be transformed and reduced by the procedure specified in §6.G as shown in Appendix D. Note that the name x in $\nu y_{\nu} x$ refers to the variable x in the outer block while the name x in xa and $\nu x_{\nu}(x+y)$ refers to x declared in the inner block. As seen in the Appendix, this causes no problem in the transformation process. This is because only the value of x is passed from the program to the procedure body. However, confusion does arise if the parameter is called by name.

To avoid the conflict between the actual parameter and other variables already present in the procedure body, we make the actual parameters corresponding to the parameters called by name unique by marking the names with star (*). Thus, $x_{m+1}, \ldots, x_i$ in the sequences (6.37) and (6.38) become $x^*_{m+1}, \ldots, x^*_i$. The effect of this on the transformation process is that $x^*_{m+1}, \ldots, x^*_i$ become global to the procedure and hence remain in the outer block when the procedure block is removed [(7b & 7c), page 24] together with its local variables which may have the names $x_{m+1}, \ldots, x_i$.

As a consequence of marking the actual parameters with * to differentiate them from the local variables of the same names in the procedure, they are treated as unique names in the value transformation in step 4 [§6.G]. We allow for this by modifying the process slightly ie. (7b) now reads:-

7b. If there exists a term $\nu x_{\nu} y$ between $\oplus\langle A\rangle$ and $\ominus$ such that $x \notin A$, then transfer the term to the end of subsequence j-1. If the term carries starred names, then remove a star from each name and apply the value transformation on this term relative

to the terms already present in subsequence j-1 . Reduce subsequence j-1 whenever possible. If the term to be transferred is part of a conditional sequence then it, together with its corresponding propositional expression, is incorporated into subsequence j-1 .

An example is given in Appendix E to illustrate this process.

A similar problem of conflict occurs in recursive type procedure between the names of the variables on the left-hand side of the procedure statement at various levels of call. This is avoided in the same way ie. by marking the name of the variable on the left-hand side with star (*) to make it unique. That is, the name y in '$\mu y \mu$name' of the sequence (6.38) becomes y^*. The transformation as modified in this section still applies in this case. An example of a program containing a recursive procedure is given in Appendix F.

CHAPTER 7

CONCLUSION

Investigators in the field of program modelling and program equivalence have adopted various approaches to the problem, some of these approaches and their relationship with our work is given below:-

1. Church's λ - calculus approach

Landin [21,22,23,24] uses the λ - calculus for modelling expressions (as opposed to statements) occuring in programming languages, taking into account the imperative features of languages with particular reference to ALGOL-60.

Burge [3] considers similar problems of evaluation and interpretation of expressions.

Strachey's work [39] on semantics is based on mapping the constructs of CPL into pure λ-notation. His work is developed in parallel with the design of CPL [2,40] and thus centralised on the concept of the "left-hand value" and "right-hand value "of an expression. Some of Strachey's work have been extended by Burstall [4,5] who also makes a detailed study of assignment based on Landin's work [25].

As the formal equivalence of the substitution properties of λ and S has been established by Nixon and Wesselkamper [31,32], we observe that any result which is obtained by the λ-calculus approach can also be derived within the framework of S.

Another approach to the theory of computation in this direction is developed by Scott [35,36,37] based upon lattice theoric models of the λ-calculus. This approach is built on the premiss that the data spaces manipulated by a computation are complete lattices whose partial ordering

represents a notion of approximation. The presentation is oriented towards potential applications to the formal definition of programming languages.

Having observed that Lattice theory can be included within the structure of S, we believe that the theorems of S can be extended to include results obtained through the lattice theoric consideration.

2. Program Schemata Approach

All the works in this direction stem from Yanov's idea of "program schemata" which are abstract algorithms or programs. Yanov's work on equivalence of programs [42,43,44] is based on the observation that the application of an algorithm to one of its arguments determines uniquely a sequence of elementary action resulting in different sequences for different arguments in general. However, it is always possible to find a finite set of predicates, representing properties of the arguments, such that the sequence of elementary actions to be performed for a given argument may be considered as a function of the values of these predicates for that argument.

This approach is similar to our treatment of the conditional statement. Another similarity between our approach and Yanov's is the concept of a canonical form, but the canonical forms introduced in these two works are very different.

Further, Yanov shows that the equivalence problem for program schemata is decidable and bases the decision on the canonical forms. However, his notion of equivalence is a very strong one i.e., two schemata are defined to be equivalent if and only if they result in exactly the same sequence of operations for all inputs. Our decision on program equivalence is also

based upon the canonical forms, but we adopt a weaker notion of equivalence namely that two programs are equivalent if they produce the same output for any given input regardless of their sequences of operations.

Yanov's approach which may be classed as the study of the properties of flow diagrams is adopted by a number of other people, mainly those in the Russian group doing research on semantics. Some of their works are presented in [8,9,10,11,12,18,19,34].

3. The Axiomatic Approach

The merit of an axiomatic treatment of concepts in programming is discussed by Hoare [13]. The main representative of this approach is Igarashi in his papers [14,15,16].

Igarashi constructs an axiom system for assignment and shows that it is complete. De Bakker [1] gives a further analysis of the axiomatics of simple assignment statements, the type we consider in the first part of Chapter IV. We note that the theorems of S given in Chapter IV cover de Bakker's axioms and rules of inference for simple assignment statements.

Axiom systems for conditional expressions have been given by McCarthy [29] and Igarashi [14,15,16]. It turns out that McCarthy's axioms can be derived from Igarashi's system and vice versa. From our consideration of conditional statements in Chapter V, it is apparent that McCarthy's axioms are also derivable from the conditional theorems of S and vice versa.

Igarashi also develops axioms for goto statements which include the label concept. He shows that equivalence of programs can be derived from his axioms and inference rules. We approach the equivalence problem by treating the goto statement and the label concept in a simpler manner in Chapter VI.

Conclusion

This thesis presents an algebraic approach to the problem of program equivalence. The use of algebraic expressions in modelling programs has the following advantages:-

1. Algebraic expressions give a precise interpretation of the result of a program statement.
2. An algebraic representation of a program can be communicated easily.
3. Algebraic expressions can be simplified, some operations can be combined and unnecessary ones eliminated.

The canonical form derived from the algebraic model of a program is essentially a problem statement of the program. It defines the program output directly in terms of the program input. In this sense, producing the canonical form generates a statement of the program output independent of the process or algorithm used to create that output. In fact, it is not only algorithm independent, but also machine independent. However, unlike most problem statements, our canonical form is highly symbolic and mathematical. Even so, we believe that it makes the program more transparent and the comparison of canonical forms does not require a great effort.

This work, together with that of Wesselkamper [41] and Maile [26], demonstrates the versatility of the operator S which, in itself, is a simple operator. Wesselkamper shows that the computer can be described in terms of S and the programs in terms of S and μ. Our interest is in the use of S and μ to represent a program. Although our model of a program differs from that of Wesselkamper, we come to the same conclusion that S and μ are sufficient as basic operators to represent any program. Furthermore, we show that the model so defined is feasible as a basis for considering program equivalence.

Closely related to our field of studies is the investigation into the setting up of a formal (pragmatic) system for describing the semantics of computing systems based on the operators S and μ carried out by Cooke [7]. While we are only interested in the concept of name and value in a program, Cooke is also concerned with representation. To cope with this notion, he has found it necessary to introduce another operator, e, to represent activation. Using the three operators S, μ and e, he gives a procedure for specifying the semantics of high-level programming languages in terms of these fundamental operations.

As a further development, it may perhaps be desirable to investigate the use of S-operations to express optimisation formulae and hence to develop a uniform theory of optimisation.

So far as the algebra of S is concerned, it would be interesting to investigate the feasibility of using the computer to perform the transformations and reduction of S-expressions as specified by the theorems developed in this work. This would give us a tool for the automatic investigation of program equivalence and related problems.

Thus we feel that we have demonstrated that in the operators S and μ we have a valid means of expressing the semantics of a program in such a way that the problems of program equivalence, transportability and compiler validation can now be tackled more meaningfully.

REFERENCES

1. de Bakker, J.W.
"Axiom System for Simple Assignment Statements", Symposium on Semantics of Algorithmic Languages, Lecture Notes in Mathematics No. 188, Springer-Verlag, Berlin (1971).

2. Barron, D.W., Buxton, J.N., Hartley, D.F., Nixon, E. and Strachey, C.
"The Main Features of CPL", Computer Journal 6 (2) (1963).

3. Burge, W.H.
"The Evaluation, Classification, and Interpretation of Expressions", Proceedings of the ACM 19th National Conference, ACM, New York (1964).

4. Burstall, R.M.
"Some Aspects of CPL Semantics", Experimental Programming Reports No. 3, University of Edinburgh (April 1965).

5. Burstall, R.M.
"Semantics of Assignment", Machine Intelligence Vol. 2, Oliver and Boyd, Edinburgh (1967).

6. Cooke, D.J.
Carabiner Paper 8, University of London, Institute of Computer Science, internal document ICSI 456 (1972).

7. Cooke, D.J.
"The Pragmatic Formalization of Computing Systems Relative to a Given High-Level Language", PhD Thesis, University of London (1975).

8. Ershov, A.P.
"Theory of Program Schemata", Information Processing 71, North Holland Publishing Company (1972).

9. Ershov, A.P.
"On Yanov's Operator Schemes", Problemy Kibernetiki 20, Nauka, Moscow (1967).

10. Ershov, A.P. and Lyapunov, A.A.
"On the Formalisation of the Notion of Program", Kibernetika (Kiev) 5(10) (1967).

11. Gorel, E.L.
"On Yanov Schemata with the Finite Equivalence Relation", Kibernetika (Kiev) (1972).

12. Gorel, E.L.
"Logical Independence of the Axiomatics of Yanov Program Schemata", Proc. 1st All-Union Conference on Program, Part A, Kiev (1968).

13. Hoare, C.A.R.
"The Axiomatic Method", The National Computing Centre, Manchester, England, (1968).

14. Igarashi, S.
"On the Logical Schemes of Algorithms", Information Processing in Japan 3 (1963).

15. Igarashi, S.
"An Axiomatic Approach to the Equivalence Problems of Algorithms with Applications", PhD Thesis, University of Tokyo (1964); reprinted in Report of the Computer Centre University of Tokyo 1 (1968).

16. Igarashi, S.
"On the Equivalence of Programs Represented by ALGOL - Like Statements", Report of the Computer Centre University of Tokyo 1 (1968).

17. Irons, E.T. and Feurzeig, W.
"Comments on the Implementation of Recursive Procedures and Blocks in ALGOL-60", CACM 4(1) (1961).

18. Itkin, V.E.
"Yanov Program Schemata with Identical Statements",
Proc. 1st All-Union Conference on Program, Part A, Kiev (1968).

19. Itkin, V.E.
"Parallel Program Schemata", Kibernetika 1, Kiev (1971).

20. Kleene, S.C.
"Introduction to Metamathematics", Amsterdam, North Holland Publishing Company (1952).

21. Landin, P.J.
"The Mechanical Evaluation of Expressions", Comp.J.6 (1964).

22. Landin, P.J.
"A Correspondence between ALGOL-60 and Church's Lambda Notation", CACM 8 (1965)

23. Landin, P.J.
"A λ - Calculus Approach", Advances in Programming and Non-Numerical Computation (L. Fox, ed.), Pergamon Press, New York (1966).

24. Landin, P.J.

"A Formal Description of ALGOL-60", Formal Language Description Languages for Computer Programming, Proc. IFIP Working Conference 1964 (T.B. Steel, Jr., ed.), North Holland Publishing Company, Amsterdam (1966).

25. Landin, P.J.

"The Next 700 Programming Languages", CACM 9 (1966).

26. Maile, J.V.

"An Investigation into the Use of a Markov-Like Substitution Operator as a Basis for Logic Circuit Design", M.Sc. Dissertation, Universtiy of London (1972).

27. Mendelson, E.

"Introduction to Mathematical Logic", D. Van Nostrand Co., Inc., Princeton, N.J. (1966).

28. Markov, A.A.

"Theory of Algorithms", U.S.S.R. Academy of Sciences (1954); English translation by the Israeli Program for Scientific Translations (1961).

29. McCarthy, J.

"A Basis for a Mathematical Theory of Computation", Computer Programming and Formal Systems (P. Brafford and D. Hirschberg, ed.), North Holland Publishing Company, Amsterdam (1963).

30. Naur, P. et al

"Revised Report on the Algorithmic Language ALGOL-60", CACM (1963).

31. Nixon E. and Wesselkamper, T.C.

"Crampon Paper I, University of London Institute of Computer Science, internal document ICSI 294 (1970).

32. Nixon, E. and Wesselkamper, T.C.
Crampon Paper II, University of London Institute of Computer Science, internal document ICSI 309 (1971).

33. Rosenbloom, P.C.
"The Elements of Mathematical Logic", Dover Publications, Inc (1950).

34. Rutledge, J.D.
"On Yanov's Program Schemata", Journal ACM 11 (1964).

35. Scott, D.S.
"Lattice-Theoric Models for the λ - Calculus", Formal Language Description Languages, IFIP Working Group 2.2, Bulletin No. 5 (1970).

36. Scott, D.S.
"The Lattice of Flow Diagrams", Formal Language Description Languages, IFIP Working group 2.2, Bulletin No. 5 (1970).

37. Scott, D.S.
"Outline of a Mathematical Theory of Computation", Formal Language Description Languages, IFIP Working Group 2.2, Bulletin No. 5 (1970).

38. Stoll, R.R.
"Set Theory and Logic", W.H. Freeman and Company, San Francisco and London (1963).

39. Strachey, C.
"Towards a Formal Semantics", Formal Language Description Languages for Computer Programming, Proc. IFIP Working Conference 1964 (T.B. Steel, Jr., Ed.), North Holland Publishing Company, Amsterdam (1966).

40. Strachey, C. (ed.)
"CPL Working Papers", University of London Institute of Computer Science (1966).

41. Wesselkamper, T.C.
"A Mathematical Model of the Computing Process in a High Level Language", PhD Thesis, University of London (1972).

42. Yanov, Y.I.
"On the Equivalence and Transformation of Program Schemes", CACM 1(10) (1958).

43. Yanov, Y.I.
"On the Matrix Program Schemes", CACM 1 (12) (1958).

44. Yanov, Y.I.
"On the Logical Schemes of Algorithm", Problems in Cybernetics, Vol. 1, Pergamon Press, New York (1960).

APPENDIX A

$\sim a$	a	T a F	STaF
F	T	T T F	F
T	F	T F F	T

a ∧ b	a	b	a T b	SaTb
T	T	T	T T T	T
F	T	F	T T F	F
F	F	T	F T T	F
F	F	F	F T F	F

a ∨ b	a	b	a F b	SaFb
T	T	T	T F T	T
T	T	F	T F F	T
T	F	T	F F T	T
F	F	F	F F F	F

a ⊃ b	a	b	T a b	STab
T	T	T	T T T	T
F	T	F	T T F	F
T	F	T	T F T	T
T	F	F	T F F	T

a ≡ b	a	b
T	T	T
F	T	F
F	F	T
T	F	F

a	b	STab	T	STba	S(STab)T(STba)
T	T	T	T	T	T
T	F	F	T	T	F
F	T	T	T	F	F
F	F	T	T	T	T

a ≠ b	a	b
F	T	T
T	T	F
T	F	T
F	F	F

a	b	SFab	F	SabF	S(SFab)F(SabF)
T	T	F	F	F	F
T	F	F	F	T	T
F	T	T	F	F	T
F	F	F	F	F	F

a	b	SaTb	a	STba	S(SaTb)a(STba)
T	T	T	T	T	T
T	F	F	T	T	F
F	T	F	F	F	F
F	F	F	F	T	T

a	b	SaTb	SaFb	T	S(SaTb)(SaFb)T
T	T	T	T	T	T
T	F	F	T	T	F
F	T	F	T	T	F
F	F	F	F	T	T

STab	a	b	F	STaF	SbF(STaF)
T	T	T	F	F	T
F	T	F	F	F	F
T	F	T	F	T	T
T	F	F	F	T	T

APPENDIX B

B1. $S\ (STq\varphi)\ T\ (SqF\omega) = \begin{cases} \varphi & \text{if } q = T \\ \omega & \text{otherwise} \end{cases}$

Proof $q = T \Rightarrow LHS = S\ (STT\varphi)\ T\ (STF\omega)$

$= S\varphi TT$

$= \varphi$

$q = F \Rightarrow LHS = S\ (STF\varphi)\ T\ (SFF\omega)$

$= STT\omega$

$= \omega$

$q = \omega \Rightarrow LHS = S\ (ST\omega\varphi)\ T\ (S\omega F\omega)$

$= STT\omega = \omega$

B2. $R = S\ (STp\theta)\ T\ (SpF\{S(STq\varphi)\ T\ (SqF\omega)\})$ corresponds to $(p \to \theta,\ q \to \varphi)$

Proof $p = T \Rightarrow R = S\ (STT\theta)\ T\ (STF\{S\ (STq\varphi)\ T\ (SqF\omega)\ \})$

$= S\theta TT$

$= \theta$

$p = F \Rightarrow R = S\ (STF\theta)\ T\ (SFF\ \{S\ (STq\varphi)\ T\ (SqF\omega)\ \})$

$= STT\ \{S\ (STq\varphi)\ T\ (SqF\omega)\ \}$

$= S\ (STq\varphi)\ T\ (SqF\omega)$

$= \begin{cases} \varphi & \text{if } q = T \\ \omega & \text{otherwise} \end{cases}$ from above

$p = \omega \Rightarrow R = S\ (ST\omega\theta)\ T\ (S\omega F\ \{S\ (STq\varphi)\ T\ (SqF\omega\}\ )$

$= STT\omega$

$= \omega$

B3. From A2, we have

$(p \to \theta,\ q \to \varphi) = S\ (STp\theta)\ T\ (SpF\ \{S\ (STq\varphi)\ T\ (SqF\omega)\ \})$

$p \wedge q = (p \rightarrow q,\ T \rightarrow F)$

$= S\ (STpq)\ T\ (SpF\ \{S\ (STTF)\ T\ (STFw)\ \})$

$= S\ (STpq)\ T\ (SpF\ \{SFTT\})$

$= S\ (STpq)\ T\ (SpFF)$

$= S\ (STpq)\ Tp$

$p \vee q = (p \rightarrow T,\ T \rightarrow q)$

$= S\ (STpT)\ T\ (SpF\ \{S(STTq)\ T\ (STFw)\ \})$

$= STT\ (SpF\ \{SqTT\})$

$= SpFq$

$\sim p = (p \rightarrow F,\ T \rightarrow T)$

$= S\ (STpF)\ T\ (SpF\ \{S(STTT)\ T\ (STFw)\ \})$

$= S\ (STpF)\ T\ (SpF\ \{STTT\})$

$= S\ (ST_IF)\ T\ (SpFT)$

$p \succ q = (p \rightarrow q,\ T \rightarrow T)$

$= S\ (STpq)\ T\ (SpF\ \{S\ (STTT)\ T\ (STFw)\ \})$

$= S\ (STpq)\ T\ (SpFT).$

APPENDIX C

In three-valued logic, the sequence

$$ST\ \{p_1\theta_1 : p_2\theta_2 : \ \cdots\ : p_n\theta_n\}$$

is used to denote an S-expression which gives the value θ corresponding to the first p with the value T except when an undefined p occurs before a true p then the expression is undefined.

Following the two-valued case, we are therefore considering a general sequence

$$Sa\ \{p_1\theta_1 : p_2\theta_2 : \ \cdots\ : p_n\theta_n\}$$

in which

$$a \in E \cup \{T, \omega\}$$

$$p_i \in \{T, F, \omega\}$$

and $\theta_i \in E \cup \{\omega\}$.

In investigating the validity of the theorems, we have to add to the given proofs the possibility of a and p_i being undefined.

The case of $a = \omega$ is trivial as the sequence will yield the value ω by definition.

It then remains to be shown whether the theorems hold for $a \in E \cup \{T\}$ and $p_i = \omega$.

Theorem 5.3 $Sa\{T\theta : p\varphi\} = Sa\{T\theta\}$

Proof Obvious

Theorem 5.4 $Sa\{F\theta\} = a$

Proof Obvious

Theorem 5.5 $Sa\{p\theta : T\theta\} = Sa\{T\theta\}$

$p = \omega \Rightarrow LHS = Sa\{\omega\theta : T\theta\} = \omega$

$RHS = Sa\{T\theta\} \neq \omega$

$LHS \neq RHS$

Theorem 5.6 $Sa\{\sim p\theta : T\varphi\} = Sa\{p\varphi : T\theta\}$

Proof $p = \omega \Rightarrow \sim p = \omega$

$\Rightarrow LHS = Sa\{\omega\theta : T\varphi\} = \omega$

$RHS = Sa\{\omega\varphi : T\theta\} = \omega$

Theorem 5.7 $Sa\{p\theta : p\varphi\} = Sa\{p\theta\}$

Proof $p = \omega \Rightarrow LHS = Sa\{\omega\theta : \omega\varphi\} = \omega$

$RHS = Sa\{\omega\theta\} = \omega$

Theorem 5.8 $Sa\{p\theta : q\theta\} = Sa\{(p \vee q)\ \theta\}$

Proof $p = \omega \Rightarrow p \vee q = \omega \Rightarrow LHS = RHS = \omega$

$p = T,\ q = \omega \Rightarrow LHS = Sa\{T\theta : \omega\theta\}$

$= Sa\{T\theta\}$ by Theorem 5.3.

$RHS = Sa\{(T \vee \omega)\ \theta\}$

$= Sa\{T\theta\}$

$p = F,\ q = \omega \Rightarrow LHS = Sa\ \{F\theta : \omega\theta\}$

$= \omega$

$RHS = Sa\ \{(F \vee \omega)\ \theta\}$

$= Sa\ \{\omega\theta\}$

$= \omega$

Corollary $Sa\ \{p\theta : q\theta\} = Sa\ \{q\theta : p\theta\}$

This result does not hold since $(p \vee q) \neq (q \vee p)$

Theorem 5.9 $Sa\ \{p\theta : \sim p\varphi\} = Sa\ \{p\theta : T\varphi\}$

Proof $p = \omega \Rightarrow LHS = Sa\ \{\omega\theta : \omega\varphi\} = \omega$

$RHS = Sa\ \{\omega\theta : T\varphi\} = \omega$

Theorem 5.10 $Sa\ \{(p \wedge q)\theta : p\varphi : r\theta : T\varphi\} = Sa\ \{(p \wedge q) \vee (\sim p \wedge r)\theta : T\varphi\}$

Proof $p = \omega \Rightarrow (p \wedge q) = \omega$

$\Rightarrow LHS = Sa\ \{\omega\theta : \omega\varphi : r\theta : T\varphi\} = \omega$

and $\omega \vee (\sim p \wedge r) = \omega$

$\Rightarrow RHS = Sa\ \{\omega\theta : T\varphi\} = \omega$

Theorem 5.11 $Sa\ \{(p \wedge q)\ \theta : p\theta : q\varphi\} = Sa\ \{(p \wedge q)\theta : q\varphi : p\theta\}$

$= Sa\ \{p\theta : q\varphi\}$

Proof

(i) $p = \omega \Rightarrow p \wedge q = \omega$

$\Rightarrow Sa\ \{(p \wedge q)\theta : p\theta : q\varphi\} = Sa\ \{\omega\theta : \omega\theta : q\varphi\}$

$= \omega$

$Sa\ \{(p \wedge q)\theta : q\varphi : p\theta\} = Sa\ \{\omega\theta : q\varphi : \omega\theta\}$

$= \omega$

$Sa\ \{p\theta : q\varphi\} = Sa\ \{\omega\theta : q\varphi\}$

$= \omega$

(ii) $p = F,\ q = \omega$

$$\Rightarrow \quad Sa\ \{(p\wedge q)\theta : p\theta : q\varphi\} = Sa\ \{(F\wedge\omega)\theta : F\theta : \omega\varphi\}$$
$$= Sa\ \{F\theta : F\theta : \omega\varphi\}$$
$$= \omega$$
$$Sa\ \{(p\wedge q)\theta : q\varphi : p\theta\} = Sa\ \{(F\wedge\omega)\theta : \omega\varphi : F\theta\}$$
$$= \omega$$
$$Sa\ \{p\theta : q\varphi\} = Sa\ \{F\theta : \omega\varphi\}$$
$$= \omega$$

(iii) $p = T,\ q = \omega$

$$\Rightarrow \quad Sa\ \{(p\wedge q)\theta : p\theta : q\varphi\} = Sa\ \{(T\wedge\omega)\theta : T^{\theta} : \omega\varphi\}$$
$$= Sa\ \{\omega\theta : T\theta : \omega\varphi\}$$
$$= \omega$$
$$Sa\ \{(p\wedge q)\theta : q\varphi : p\theta\} = Sa\ \{\omega\theta : \omega\varphi : T\theta\}$$
$$= \omega$$
$$Sa\ \{p\theta : q\varphi\} = Sa\ \{T\theta : \omega\varphi\}$$
$$= Sa\ \{T\theta\}$$

Hence when p,q are allowed to take the value ω

$$Sa\ \{(p\wedge q)\theta : p\theta : q\varphi\} = Sa\ \{(p\wedge q)\theta : q\varphi : p\theta\}$$
$$\neq Sa\ \{p\theta : q\varphi\}$$

Theorem 5.12 $Sa\ \{(p\wedge q)\theta : p\varphi : q\psi\} = Sa\ \{(p\wedge q)\theta : q\psi : p\varphi\}$

Proof (i) $p = \omega \Rightarrow p\wedge q = \omega$

$\Rightarrow$ LHS = RHS = ω

(ii) $p = T,\ q = \omega \Rightarrow p\wedge q = \omega$

$\Rightarrow$ LHS = RHS = ω

(iii) $p = F,\ q = \omega \Rightarrow\ p \wedge q = F$

$\Rightarrow$ LHS $= Sa\ \{F\theta : F\varphi : \omega\psi\}$

$= \omega$

RHS $= Sa\ \{F\theta : \omega\psi : F\varphi\}$

$= \omega$

Theorem 5.14 $Sa\ \{p\ ST\ \{p_1\theta_1 : p_2\theta_2 : \ldots : p_n\theta_n\}\} = Sa\ \{(p \wedge p_1)\theta_1 : \ldots : (p \wedge p_n)\theta_n\}$

Proof

(i) $p = \omega \Rightarrow p \wedge p_i = \omega \quad \forall\ p_i \in \{T, F, \omega\}$

$\therefore$ LHS $= Sa\ \{\omega\ ST\ \{p_1\theta_1 : \ldots ; p_n\theta_n\}\}$

$= \omega$

RHS $= Sa\ \{\omega\theta_1 : \ldots : \omega\theta_n\}$

$= \omega$

Now if $a \in E$ then the result is trivially true so consider $a = T$.

Suppose p_i is the first p which has the value T and $p_j = \omega$

(ii) $p = T$

$\Rightarrow$ LHS $= ST\ \{TST\ \{p_1\theta_1 : p_2\theta_2 : \ldots : p_n\theta_n\}\}$

$= ST\ \{p_1\theta_1 : p_2\theta_2 : \ldots : p_n\theta_n\}$

$$= \begin{cases} \theta_i & \text{if } i < j \\ \omega & \text{if } i > j \end{cases}$$

RHS $= ST\ \{(T \wedge p_1)\theta_1 : (T \wedge p_2)\theta_2 : \ldots : (T \wedge p_n)\theta_n\}$

$T \wedge \omega = \omega \Rightarrow$ RHS $= \begin{cases} \theta_i & \text{if } i < j \\ \omega & \text{if } i > j \end{cases}$

(iii) $p = F$

$\Rightarrow$ LHS $= ST \{F\ ST \{p_1\theta_1 : \ldots : p_n\theta_n\}\}$

$= \omega$ by (5.17)

$p = F \Rightarrow p \wedge p_i = F \quad \forall\ i \in \{T, F, \omega\}$

$\therefore$ RHS $= ST \{F\theta_1 : F\theta_2 : \ldots : F\theta_n\}$

$= \omega$ by (5.17)

Theorem 5.15 $ST \{p_1\theta_1 : p_2\theta_2 : \ldots : p_n\theta_n\} ; ST \{q_1\varphi_1 : q_2\varphi_2 : \ldots : q_m\varphi_m\}$

$= ST \{(p_1 \wedge q_1)\ [\theta_1 ; \varphi_1] : \ldots : (p_1 \wedge q_m)\ [\theta_1 ; \varphi_m] :$

$\vdots$

$(p_n \wedge q_1)\ [\theta_n : \varphi_1] : \ldots : (p_n \wedge q_m)\ [\theta_n ; \varphi_m]\}$

Proof Suppose p_i is the first p that has the value T

q_j is the first q that has the value T

and $p_r = q_s = \omega$, then

(i) $r < i$

$\Rightarrow$ LHS $= ST \{F\theta_1 : \ldots : \omega\theta_r : \ldots : T\theta_i : \ldots : p_n\theta_n\} ; ST \{q_1\varphi_1 : \ldots : q_m\varphi_m\}$

$$= \begin{cases} \omega ; \varphi_j & \text{if } j < s \\ \omega ; \omega & \text{if } j > s \end{cases}$$

$= \omega$

$F \wedge p = F \ \forall\ p \in \{T, F, \omega\}$ and $(T \wedge p) = T$ only if $p = T$

$\Rightarrow \quad (p_k \wedge q_l) = F \quad \forall\ k < i$ and $\forall\ l$

and $(p_i \wedge q_j)$ is the first expression with the value T.

$p_r = \omega \Rightarrow (p_r \wedge q_j) = \omega$

and $r < i \Rightarrow (p_r \wedge q_j)$ occurs before $(p_i \wedge q_j)$

Hence RHS $= \omega$

(ii) $r > i,\ s > j$

$\Rightarrow$ LHS = $\theta_i\ ;\ \varphi_j$

and $r > i,\ s > j \Rightarrow (p_i \wedge q_j) = T$ occurs before any undefined expression,

hence RHS = $\theta_i\ ;\ \varphi_j$

(iii) $r > i,\ s < j$

$\Rightarrow$ LHS = $\theta_i\ ;\ \omega = \omega$

and $s < j \Rightarrow (p_i \wedge q_s) = \omega$ occurs before $(p_i \wedge q_j) = T$.

Hence RHS = ω.

APPENDIX D

The dynamic sequence

$S\mu[\ \oplus\langle a,x\rangle\ ;\ \text{нач}\alpha;\text{нхч}\partial$

$\oplus\langle x,y\rangle\ ;\ \text{нун}x;$

$\text{ха}\ ;$

$\oplus\ ;\ \text{нхч}(x+y)\ ;\ \ominus\ ;$

$\ominus\ ;$

$\ominus\quad]$

may be transformed by the method described in [§6.G] as follows:

Step 2 $\Rightarrow\ S\mu[\ \oplus\ \langle a,x\rangle\ ,\ \text{нач}\alpha,\text{нхч}\partial;$

$\oplus\ \langle x,y\rangle,\ \text{нун}x;$

$\text{ха};$

$\oplus\ ;\ \text{нхч}(x+y)\ ,\ \ominus\ ;$

$\ominus$

$\ominus\quad]$

Step 3 $\Rightarrow\ S\mu[\ \oplus\langle a,x\rangle\ ,\ \text{нач}\alpha,\text{нач}\partial;$

$\oplus\langle x,y\rangle\ ,\ \text{нун}x;$

$\text{ха}\ ;$

$\oplus\ ;\ \text{нач}(a+y)\ ,\ \ominus\ ;$

$\ominus\ ;$

$\ominus\quad]$

Step 4-7

i=2: $\Rightarrow\ S\mu[\ \oplus\langle a,x\rangle\ ;\ \text{нач}\alpha,\text{нхч}\partial;$

$\oplus\langle x,y\rangle\ ,\ \text{нун}\partial;$

i=3: $\Rightarrow\ S\mu[\ \oplus\langle a,x\rangle\ ,\ \text{нач}\alpha,\text{нхч}\partial;$

$\oplus\langle x,y\rangle\ ,\ \text{нун}\partial,\text{ха};$

i=4: $\Rightarrow\ S\mu[\ \oplus\langle a,x\rangle\ ,\ \text{нач}\alpha,\text{нхч}\partial;$

$\oplus\langle x,y\rangle\ ,\ \text{нун}\partial,\text{ха};$

$\oplus\ ,\ \text{нач}(\alpha+\partial)\ ,\ \ominus;$

$\Rightarrow$ $S\mu[\oplus\langle a,x\rangle$, $\text{наи}\alpha,\text{нхи}\,\partial$;

$\oplus\langle x,y\rangle$, $\text{нуи}\,\partial,\text{ха},\text{наи}(\alpha+\partial)$;

i=5: $\Rightarrow$ $S\mu[\oplus\langle a,x\rangle$, $\text{наи}\alpha,\text{нхи}\,\partial$;

$\oplus\langle x,y\rangle$, $\text{нуи}\,\partial,\text{ха},\text{наи}(\alpha+\partial),\ominus$;

$=$ $S\mu[\oplus\langle a,x\rangle$, $\text{наи}\alpha,\text{нхи}\,\partial$;

$\oplus\langle x,y\rangle$, $\text{нуи}\,\partial,\text{нхи}(\alpha+\partial),\text{наи}(\alpha+\partial),\ominus$;

$=$ $S\mu[\oplus\langle a,x\rangle$, $\text{наи}\alpha,\text{нхи}\,\partial,\text{наи}(\alpha+\partial)$;

i=6: $\Rightarrow$ $S\mu[\oplus\langle a,x\rangle$, $\text{наи}\alpha,\text{нхи}\,\partial,\text{наи}(\alpha+\partial),\ominus\,]$

$=$ $S\mu[\oplus\langle a,x\rangle$, $\text{нхи}\,\partial,\text{наи}(\alpha+\partial),\ominus\,]$

APPENDIX E

Consider the following program:

```
begin dec x
      procedure halve (x)
      specification x
         x:=x/2
      x:=2
      halve (x)
end
```

The procedure declaration has the model

```
ν halve =  Sμ[ ⊕ <x> ;
                        x( ) ;
                ⊕ ; νxν(x/2) ; ⊖ ;
                                ⊖  ]
```

and the program corresponds to the sequence

$$S\mu[\oplus\ \langle x\rangle\ ;\ \nu x\nu 2\ ;\ \nu\text{halve}\ (x)\ ;\ \ominus]$$

which, on substitution of νhalve(x) gives:-

```
Sμ[ ⊕ <x> ;  νxν2 ;
    ⊕ <x> ;
                xx* ;
    ⊕ ; νxν(x/2) ; ⊖ ;
                    ⊖ ;
                    ⊖  ]
```

The transformation [§6.G] gives

```
Sμ[ ⊕ <x> ; νxν2  ;
    ⊕ <x> ;
              xx* ;
    ⊕ , νx*ν(x*/2) , ⊖ ;
                     ⊖ ;
                     ⊖  ]
```

$$= \quad S\mu[\ \oplus\ \langle x\rangle\ ,\ n\,x\,n2\ \ ;$$

$$\oplus\ \langle x\rangle\ ,\ xx^*, n x^* n(x^*/2),\ \ominus\ ;$$

$$\ominus\ \ ]$$

$$= \quad S\mu[\ \oplus\ \langle x\rangle\ ,\ n\,x\,n2\ \ ;$$

$$\oplus\ \langle x\rangle\ ,\ n\,x\,n(x^*/2), n x^* n(x^*/2),\ \ominus\ ;$$

$$\ominus\ ]$$

$$= \quad S\mu[\ \oplus\ \langle x\rangle\ ,\ n\,x\,n2; n x^* n(x^*/2);$$

$$\oplus\ \langle x\rangle\ ,\ n\,x\,n(x^*/2)\ \ ,\ \ominus\ ;$$

$$\ominus\ \ ]$$

$$= \quad S\mu[\ \oplus\ \langle x\rangle\ ,\ n\,x\,n2, n\,x\,n(2/2);$$

$$\ominus\ \ ]$$

$$= \quad S\mu[\ \oplus\ \langle x\rangle\ ,\ n\,x\,n1\ \ ,\ \ominus\ ]$$

APPENDIX F

```
begin dec x
      dec procedure fact(n)
      value n
      specification n
      if n = 1 then fact := 1
               else fact := n × fact(n-1)
     x := fact(3)
end
```

$$\nu\,\text{fact} = S\mu[\ \oplus\ \langle n,\ \text{fact}\rangle\ ;\ \nu n\nu(\ )\ ;\ \oplus\ ;\ \{\nu(n=1)[\nu\text{fact}\nu 1]\ :\ T[\nu\text{fact}\nu(n\times\text{fact}(n-1))]\};\ \ominus\ ;\ \nu(\ )\nu\text{fact}\ ;\ \ominus\ ] \qquad (1)$$

and the program corresponds to the sequence

$$S\mu[\ \oplus\ \langle x\rangle\ ;\ \nu x\nu\text{fact}(3)\ ;\ \ominus\ ] \qquad (2)$$

Now, from (1),

$$\nu x\nu\text{fact}(3) = S\mu[\ \oplus\ \langle n,\text{fact}\rangle\ ;\nu n\nu 3;\ \oplus\ ;\ \{\nu(n=1)[\nu\text{fact}\nu 1]\ :\ T[\nu\text{fact}\nu(n\times\text{fact}(n-1))]\};\ \ominus\ ;\ \nu x^{*}\nu\text{fact}\ ;\ \ominus\ ]$$

$$= S\mu[\ \oplus\ \langle n,\text{fact}\rangle,\nu n\nu 3;\ \oplus\ ,\ \{\nu(3=1)[\nu\text{fact}\nu 1]\ :\ T[\nu\text{fact}\nu(3\times\text{fact}(3-1))]\},\ \ominus\ ;\ \nu x^{*}\nu\text{fact}\ ,\ \ominus\ ]$$

$$= S\mu[\ \oplus\ \langle n,\text{fact}\rangle\ ,\nu n\nu 3;\ \oplus\ ,\ \nu\text{fact}\nu(3\times\text{fact}(2))\ ,\ \ominus\ ;\ \nu x^{*}\nu\text{fact}\ ,\ \ominus\ ] \qquad (3)$$

Again from (1),

$$\nu\text{fact}\nu(3\times\text{fact}(2)) = S\mu[\ \oplus\ \langle n,\text{fact}\rangle,\nu n\nu 2;$$

$\oplus$; {μ(n=1)[μfactμ1] : T[μfactμ(n×fact(n-1))]}; $\ominus$;

μfact*μ(3×fact) ; $\ominus$] from (6.39)

= Sμ[$\oplus$ <n,fact>,μnμ2;

$\oplus$, μfactμ(2×fact(1)) , $\ominus$;

μfact*μ(3×fact) , $\ominus$] (4)

Similarly, μfactμ(2×fact(1))

= Sμ[$\oplus$ <n,fact> , μnμ1 ;

$\oplus$, μfactμ1, $\ominus$;

μfact*μ(2×fact) , $\ominus$]

= Sμ[$\oplus$ <n,fact> , μnμ1,μfactμ1;

μfact*μ(2×1) , $\ominus$]

= Sμ[$\oplus$ <n,fact>, μnμ1,μfactμ1,μfact*μ2, $\ominus$] (5)

Substituting (5) in (4) gives

μfactμ(3×fact(2))

= Sμ[$\oplus$ <n,fact> , μnμ2 ;

$\oplus$;

$\oplus$ <n,fact> , μnμ1,μfactμ1,μfact*μ2, $\ominus$;

$\ominus$;

μfact*μ(3×fact) , $\ominus$]

= Sμ[$\oplus$ <n,fact> , μnμ2 ;

$\oplus$, μfactμ2 , $\ominus$;

μfact*μ(3×fact) , $\ominus$]

= Sμ[$\oplus$ <n,fact> , μnμ2,μfactμ2 ;

μfact*μ(3×fact) , $\ominus$]

= Sμ[$\oplus$ <n,fact> , μnμ2,μfactμ2,μfact*μ(3×2), $\ominus$]

(6)

Substituting (6) in (3) to give

$$\begin{aligned}
\text{fact}\,\text{fact}(3) \;=\; & S\mu[\ \oplus\ \langle n,\text{fact}\rangle\ ,\ n\,n3;\\
& \quad \oplus\ ;\\
& \quad \oplus\ \langle n,\text{fact}\rangle,\ n\,n2,n\,\text{fact}\,n2,n\,\text{fact}*n6,\ \ominus\ ;\\
& \qquad\qquad\qquad\qquad\qquad\qquad \ominus\ ;\\
& \qquad\qquad\qquad\qquad n\,x*n\,\text{fact}\ ,\ \ominus\ \ ]\\
= \; & S\mu[\ \oplus\ \langle n,\text{fact}\rangle,\ n\,n3;\\
& \quad \oplus\ ,\ n\,\text{fact}\,n6\ ,\ \ominus\ ;\\
& \qquad n\,x*n\,\text{fact}\ ,\ \ominus\ ]\\
= \; & S\mu[\ \oplus\ \langle n,\text{fact}\rangle\ ,\ n\,n3,n\,\text{fact}\,n6,n\,x*n6,\ \ominus\ ] \qquad (7)
\end{aligned}$$

And thus, the substitution of (7) in (2) gives

$$\begin{aligned}
& S\mu[\ \oplus\ \langle x\rangle\ ;\\
& \quad \oplus\ \langle n,\text{fact}\rangle\ ,\ n\,n3,n\,\text{fact}\,n6,n\,x*n6,\ \ominus\ ;\\
& \qquad\qquad\qquad\qquad \ominus\ ]\\
= \; & S\mu[\ \oplus\ \langle x\rangle\ ,\ n\,x\,n6\ ,\ \ominus\ ]
\end{aligned}$$