Jones PERS

TECHNICAL REPORT

TR 25.104
3 April 1970

THE BLOCK CONCEPT AND SOME POSSIBLE IMPLEMENTATIONS, WITH PROOFS OF EQUIVALENCE

W. HENHAPL
C. B. JONES

IBM LABORATORY VIENNA

IBM LABORATORY VIENNA, Austria

THE BLOCK CONCEPT AND SOME POSSIBLE IMPLEMENTATIONS, WITH PROOFS OF EQUIVALENCE

W. HENHAPL
C.B. JONES

ABSTRACT:
The block concept is discussed based on a formal definition of those parts of a programming language which are affected by block structure. This includes introduction of local variables, procedure call and goto statements. The formal definition uses the Vienna Method to give a model based on the copy rules given in the ALGOL 60 report. Two basically different implementations for referencing variables are developed via a number of explanatory techniques. All of these models, which are also defined using the Vienna Method, are shown to be equivalent to the defining model.

Locator Terms for IBM Subject Index
Programming Language
Block Structure
Semantics
Equivalence
Proofs
21 PROGRAMMING

TR 25.104

3 April 1970

CONTENTS

0. INTRODUCTION

Block structure[1] was introduced to programming languages by ALGOL 60 (see /3/) and was intended mainly to facilitate the definition of local variables. However, it has also had a considerable effect on some of the existing features of languages: the ability to write nested procedure definitions and the range of possibilities for parameter passing have proved to be useful generalisations, whereas the generalisation of the goto statement so that it can induce abnormal block termination has presented significant problems and is not such an obvious advance. We shall employ the term block concept to cover all of these aspects of block structure.

Section 1 gives a formal definition, on which all subsequent discussion is based, of the relevant parts of a language. This has not been extracted from a particular language but is an attempt to abstract the important parts common to block structure languages, as a result the report should be relevant to this whole class of languages. The division of the language puts any implementation in its correct role as an organizing routine which sets up the environment in which all other code will operate. Thus the further (diverse) details of block structure languages are successfully excluded from our discussion. Section 1 justifies the omission of some further parts of languages whose relevance to the block concept may give rise to surprise at their exclusion. Furthermore, the section includes an outline of some of the problems to be resolved by an implementation.

The efficient implementation of reference to local variables in block structure languages is a nontrivial problem. The elegant system described by Dijkstra and used extensively (/4/,/5/) is developed in sections 2 to 4 of this report. The development of an alternative system, of which the authors became aware via the IBM F-level compiler for PL/I (/9/), occupies sections 5 to 7. Both of these actual implementation techniques are formally described as are a number of conceptually simpler models used in the development of the methods. (Under certain circumstances some of these intermediate models could also be used for implementation.) Section 2 begins with an introductory discussion of implementation techniques in general before progressing to the first implementation in detail.

Throughout the report the formal definitions are written as abstract interpreters using the Vienna Method [2] (/2/). The use of formal definitions permits the construction of formal equivalence proofs for the models. The claim that all of the proposed implementation models are equivalent to the defining model is supported, in the more difficult cases, by just such formal proofs.

---

1) Terms defined in the report are underlined on their defining occurence.

2) The observation that the models for the presented parts of our language can be written without using any direct reference to the control part of the machine (see /6/) has been utilized to write instructions of a form which are trivially convertable to functions.

It is hoped that this report will provide a useful source document for implementation designers who are interested in block structure languages, and the report assumes the reader is familiar with an outline of the problems. The texts introducing both the language and each implementation technique aim to review the problem and its solution. This text does not require a knowledge, and therefore lacks the precision, of formal definition methods. The formal definitions of the techniques are connected to the introductory text by a description which links facets of the model to requirements mentioned in the text, (section 1.2.3 is included to explain how the base model was derived). Appendices I and II present models which may be more directly useable by implementors because the use of recursion has been replaced by equivalent iteration.

The justifications have been simplified by the separation of a number of preliminary properties of the models. These properties should be intuitively acceptable and in most cases the report only indicates how a proof would be constructed, in the more difficult cases formal lemmas are provided. Given the properties, the proofs of the theorems are usually straightforward (even theorem 2-4 has a simple structure, its length results from the induction required). In spite of these efforts, formal proofs of correctness are certainly the most difficult part of the report and certain readers may choose to accept their presence as reassuring without following through their details. However, the authors' experience suggests that anyone proposing his own implementation technique for a concept of this complexity would do well to provide a justification along these lines.

A summary section makes some informal remarks about efficiency and includes a table summarizing the main differences between the models given in the report. Appendix III contains definitions, or references to where definitions may be found, for the notation used in the report. Basically a reader familiar with the introductory parts of /2/ should experience little difficulty in reading the formal parts of the report. Appendix IV is included to suggest a way of reading abbreviations compatible with their formal use.

1. THE LANGUAGE

This section introduces the language on which the report is based. Before coming to its formal definition, an indication is given of what is included in, and what is omitted from, the language. In addition the readers attention is drawn to the sources of some of the problems involved in its implementation.

The definition includes the interpretation of blocks, procedure call and goto statements and in addition any other statements satisfying certain properties (1-1) could be included. All statements use names to refer to values. Names come into existence by being declared in a block and have a scope which includes all of the statements of the declaring block and its nested blocks unless the name is redeclared. If a procedure is invoked, all occurences of its formal parameters within its body are replaced by the corresponding element of the argument list (making necessary changes to avoid clashes as described in /3/). Thus values can be referred to outside the scope of the name with which they were first associated. The association of names with values (known as denoting) is either constant, in which case the value is linked with the name when the block is entered, or variable, in which case the association may be changed by assignment. The only values discussed in detail below are labels and procedures where both are used only as constants. Procedure values consist of a formal parameter list and a body which is a block. Label values consist of an index to the statement list.

It is assumed that the original concrete program has been translated to an object which satisfies the abstract syntax given below. In particular, the translator is assumed to have converted the label prefix notation into a declaration, in the smallest embracing block, of the name with the index of the statement it preceded as its attribute. Furthermore, the translator will have surrounded any procedure body, which was not in the form of an unlabelled block, with a dummy block. The translator will also reject any program which would not yield a "proper program" (see section 1.1).

A number of language features related to the block concept are not included in the language discussed, the following paragraphs list the more important of these and indicate why they are exlcuded.

All names introduced in a block are considered to be local (i.e. their existence is linked to that of the block in which they were declared). The own variables of ALGOL 60 or static variables of PL/I may, if their size is known, be considered to be local variables of a global block.

The method of parameter passing used in the definition is known as "by name". "By value" passing can be expressed in terms of by name and is therefore not discussed. Moreover, "by reference" is essentially adequate to model "by name" and the implementations can make use of this by assuming that only names are passed as arguments.

Values which are aggregates of other values can be handled, providing their size is fixed on introduction, by treating a dope vector for the aggregate as a simple value A dope vector contains control information including a pointer to a separate area where the aggregate value is stored.

A further relevant limitation (the omission of label and procedure variables) is discussed after the following paragraphs which describe some implementation consequences of the block concept.

The syntax of a block structure language is such that blocks [1)] are properly nested pieces of text. The semantics of such a language ensures that blocks are active on a strictly last-in first-out basis (1-5). Thus, given dynamic areas for each block invocation, containing space to store the values of the names introduced, these can be organized into a stack. All of the implementations shown below will rely on such a stack and the pointers linking the elements of the stack will comprise the dynamic chain.

Were the language such that a block could only be active at most once it would be possible to associate fixed adresses with all names, and replace all names by these addresses at compile time. However, recursive invocation of procedures results in the possibility of the existence of an arbitrary number of activations of a block and thus its names. Thus, apart from the difficulty that names are not unique in the original program, it is not even possible to assign a single address to a name qualified by its declaring block which will locate the required value. There are three possible solutions: the original names can be associated with addresses using a directory which is dynamically updated on block entry; the original name can be used to search through the dynamic areas in an order such that the appropriate value is found; the names may be supplemented by an index to a dynamically updated directory which points to the dynamic area containing the required value. The first of these techniques yields clear expositions (see /7/, and in an even more basic form, section 1.2). The second and third, which lead to practical implementations, are discussed further at the beginning of section 2.

Thus far, given the name of the block in which a reference was declared, a directory of the last formed dynamic area for each block would suffice to locate the relevant dynamic area. The possibility of invoking procedures which have been passed as parameters would invalidate such an implementation for the following reason: it is now possible that an invoked procedure may have been declared in, and passed from, a block which has been reactivated before the call. Such a reactivation would have introduced

1) Throughout the sequel, with the exception of syntaxes, the term simple block is used to refer to the syntactic unit identified by is-block and block is used generically to include blocks and procedures.

a new dynamic area for the declaring block, but this is not the one containing the names referencable from the body of the invoked procedure. It must now be possible to locate the dynamic area of the block which introduced the invoked procedure.

Apart from recovery of the values of names, goto statements require that we are able to recreate the state at the time of the declaration of the label value.

We can now mention the last constraint imposed on our language. Label and procedure parameters have presented us with the problem of locating the declaring block, but we have always been secure in the knowledge that it still exists. The addition of label and procedure variables opens the possibility that a reference is made to such a variable at a time after its current value has ceased to exist. If, as in most languages, this situation is defined as an error, checks must be devised to dynamically detect such errors.

1.1 Abstract Syntax

```
A1    is-program = is-block
A2    is-block = (<s-dp:is-dec-list>,<s-sp:is-st-list>)
A3    is-dec = (<s-id:is-id>,<s-atr:is-atr>)
A4    is-atr = is-lab-atr ∨ is-proc-atr ∨ is-other-atr
A5    is-lab-atr = is-int
A6    is-proc-atr = (<s-pp:is-id-list>,<s-body:is-block>)
A7    is-other-atr = ...
A8    is-st = is-block ∨ is-call ∨ is-goto ∨ is-other-st
A9    is-call = (<s-nm:is-id>,<s-ap:is-id-list>)
A10   is-goto = is-id
A11   is-other-st = ...
```

The definition model (and all implementations) will assume that any program conforms to all of the normal context-sensitive syntax constraints. The term proper program is used for an object which satisfies is-program and uses all names in accordance with the declared properties and values.

1.2 Interpreter

This section contains an interpreter written in the ULD style which will be taken as the definition of the language. Since the definition is by a model its essential points must be stated, so that in the subsequent proofs the criteria for a correct implementation are clear. In the absence of any output statements we are forced to relate correctness to the values referred to by the names. Since the label and procedure values are used for housekeeping, it is only strictly necessary to consider the other denotations. However, all of the implementations given in the sequel follow this interpreter closely enough that the information in all values is available. Thus our notion of equivalence will be the relation of the values of names.

The derivation of the model given below from the ALGOL copy rule is discussed in 1.2.3, so this introduction confines itself to an outline of the state components of the model. The DN component contains the association of names with the values they denote (the names being objects satisfying is-un which have been inserted by the interpreter in place of the original names [1]). The ABN component is used to signal that termination via a goto statement has occured within int-st, so that int-next-st can take the required action (for a fuller discussion of ABN see /6/). The BL component is a list, with one element for each active block, whose elements contain TX, PTR and CTR sub-components. The TX component contains the complete text of the program as expanded by the copy rule. The PTR component is a pointer to the currently active block within TX and within that block CTR is the index of the current statement.

The intial state of the interpreter contains a single element BL list. The TX subcomponent contains the given program as the only statement of a dummy block which has an empty declaration part. The interpretation is caused to begin with int-next-st.

1.2.1 State

```
B1    is-state = (<s-bl:is-bl-list>,<s-dn:({<un:is-den> || is-un(un)})>,
                 <s-abn:is-lab-den v is-Ω>,<s-c:is-c>)
B2    is-bl = (<s-tx:is-m-block>,<s-ptr:is-sel>,<s-ctr:is-int>)
B3    is-den = is-proc-den v is-lab-den v is-other-den
B4    is-proc-den = is-sel
B5    is-lab-den = (<s-b:is-sel>,<s-dcl:is-int>)
B6    is-other-den = ...
B7    is-c = see /2/
      is-m-block etc. are as in section 1.1 except that all occurences of is-id have been
      changed to is-id' where:is-id' = is-id v is-un
```

1) The function un is not like that proposed in /7/ which yields unique names throughout a computation. It is sufficient for the purposes of this definition that two equal "unique" names are not active at the same time, which is assured by the definition given below. The reader may find it useful to think of unique names as addresses.

```
    Abbreviations used;
    BL = s-bl(ξ)
    DN = s-dn(ξ)
    ABN = s-abn(ξ)
    B = last(BL)
    TX = s-tx(B)
    PTR = s-ptr(B)
    CTR = s-ctr(B)
    P = elem(CTR)∘s-sp∘PTR
    ST = P(TX)
    cur(bl) = elem(length(bl))∘s-bl
```

The initial state ξ_o for any t_o such that is-program(t_o) is

```
B0  ξ_o = μ_o(<s-bl:[μ_o(<s-tx:μ_o(<s-sp:[t_o]>,<s-dp:[]>)>,
                          <s-ptr:I>,
                          <s-ctr:0>)]>,
           <s-c:int-next-st>)
```

1.2.2 State transition function

```
B8  int-next-st =
       CTR < length(s-sp∘PTR(TX)) ∧ ABN = Ω ──→ int-next-st;
                                                 int-st;
                                                  step-ctr
       s-b(ABN) = PTR ──→ int-next-st;
                           int-st;
                            set-ctr
       T ──→ null

B9  int-st =
       is-m-block(ST) ──→ epilogue;
                            int-next-st;
                             inst-bl
       is-m-call(ST) ──→ epilogue;
                           epilogue;
                            int-next-st;
                             inst-bl;
                              inst-proc(s-ap(ST),s-nm(ST)(DN))
       is-m-goto(ST) ──→ s-abn:ST(DN)
                  ⋮
```

B10 inst-bl =

s-bl:BL$^\frown$[μ_0(<s-tx:μ(TX;<P:mod-b(ST,BL)>)>,
<s-ptr:P>,
<s-ctr:0>)]

s-dn:μ(DN;{<s-id$\circ$elem(i)$\circ$s-dp(mod-b(ST,BL)):s-atr$\circ$elem(i)$\circ$s-dp$\circ$P> |
is-m-proc-atr(s-atr$\circ$elem(i)$\circ$s-dp(mod-b(ST,BL)))}

$\cup${<s-id$\circ$elem(i)$\circ$s-dp(mod-b(ST,BL)):μ_0(<s-b:P>,<s-dcl:i>)> |
is-m-lab-atr(s-atr$\circ$elem(i)$\circ$s-dp(mod-b(ST,BL)))} $\cup$...)

B11 inst-proc(arg-1,s) =

s-bl:BL$^\frown$[μ_0(<s-tx:μ(TX;<P:s-body(mod-p(s(TX),arg-1))>)>,
<s-ptr:PTR>,
<s-ctr:CTR>)]

B12 step-ctr = s-ctr$\circ$cur(BL):CTR + 1

B13 set-ctr = s-ctr$\circ$cur(BL):s-atr$\circ$elem(s-dcl(ABN))$\circ$s-dp$\circ$PTR(TX)
s-abn : Ω

B14 epilogue = cur(BL):Ω

B15 mod-b(blk,bl) =

μ(blk;{<α:un(α(blk),length(bl))> | is-id(α(blk)) $\wedge$ is-decl(α(blk),blk) $\wedge$
$\neg$is-red(α(blk),α,blk)})

B16 mod-p(prc,arg-1) =

μ(prc;{<α:elem(i,arg-1)> |
is-id(α(prc)) $\wedge$ elem(i)$\circ$s-pp(prc) = α(prc) $\wedge$ $\neg$ is-red(α(prc),α,prc)})

B17 un(id,n): one to one mapping of identifier integer pairs to objects satisfying
is-un.

B18 is-red(id,sel,t) = ($\exists\,\alpha,\beta$)(sel = $\alpha\circ\beta$ $\wedge$ $\beta \neq$ I $\wedge$ is-intr(id,β(t)))

B19 is-intr(id,t) = is-m-block(t) $\longrightarrow$ is-decl(id,t)
is-m-proc-atr(t) $\longrightarrow$ is-parmd(id,t)
T $\longrightarrow$ F

B20 is-decl(id,blk) = ($\exists$ i)(s-id$\circ$elem(i)$\circ$s-dp(blk) = id)

B21 is-parmd(id,prc) = ($\exists$ i)(elem(i)$\circ$s-pp(prc) = id)

1.2.3 Design criteria

As was mentioned in the introduction, this section explains how the given model is related to the copy rule. The essence of this rule is to handle procedure calls by replacing them with an appropriately modified copy of the body of the relevant procedure value. In order to avoid clashes of those argument names, substituted for the formal parameters, and the declared names within the body, necessary changes are made to the latter. Since the necessity of a change can only be determined dynamically, a considerable simplification results from making a larger set of changes. With the aid of mod-b, unique names are substituted for all local names on block entry, thus obviating the requirement to check for clashes.

The copy rule can now be followed without difficulty. The simplest way to recover the old text after procedure execution (not discussed in /3/) is to retain copies of all old texts. This accounts for the local text components in the model.

In the description of procedure calling no guidance is given as to where to obtain the original text. No justification is given for creating copies of all procedure values in the denotation directory, nor would it be in the spirit of the copy rule to search for the denotation. In the model, the necessary connection of the name of a procedure with its value is achieved by storing a pointer into TX as the value.

We now list some other facets of the model and describe why they are handled in the way given.

The treatment of labels on a par with declarations is compatible with their role. It is assumed that the translator has extracted, from the common label prefix notation, the name and its location and formed this pair into an element of the declaration set of the smallest enclosing block.

The procedure denotation is a way of accessing the procedure declaration. Whilst this is also true of the label denotation, this latter must provide us with more information. In the case of a goto statement leading to a previous activation of some block it is necessary to be able to locate the relevant state. This difference is reflected in the information stored in the relevant denotations: a procedure denotation is simply a text pointer, whereas a label separates the information permitting location of the block from that which pinpoints the declaration within that block.

The definition (A2) of s-dp as a list might appear to contradict the knowledge that the order of declarations, in a language of this type, is unimportant. The decision to use a list is really based on the rejection of the alternatives. It is possible to define a translator which will put the names of the declarations on the selectors, but this creates an anomolous object which contains information of the program in its structure thus confounding the distinction between the text and the structure thereof. The collection of the declarations into a set introduces the notational problem of there not being a selector to locate information and thus forcing the use of existential quantifiers. The use of a list, providing no essential use is made of the order, gives a set plus selectors, which is what is required.

The organisation of the data about temporarily suspended activations into a list rather than the more orthodox, but obviously equivalent, dump has two advantages. First, a pointer to an element of the list is constant whereas a component of the dump changes its position each time the dump is pushed down or popped up. Second, the implementation in a linear store is far easier to relate to actual implementations.

1.3 Properties of the Language

This section contains some properties (1-2 to 1-9) which can be deduced from the machine given in section 1.2. Since that machine is the definition of the language, these properties are considered to be of the language. This section begins with a predicate and some functions required below and an axiom which constrains any extension to the language. (The sub/superscript notation used is explained in Appendix III).

The term directly contained is used of a selector with respect to a piece of text if no part of the selector selects a part of the text satisfying is-block.

B22 $\text{is-dir-cont}(\alpha,t) = \neg(\exists\beta,\gamma)(\alpha = \gamma\circ\beta \wedge \beta \neq I \wedge \text{is-m-block}(\beta(t)))$

The inverse functions of un(B17) will be

B23 id-part(un)
B24 len-part(un)

Axiom 1-1: This axiom imposes the restriction that the interpretation of any statement of type is-other-st can change no part of the state other than those elements of DN which satisfy is-other-den.

For any state ξ about to execute int-st, let ξ' be the state after completion of that instruction.

$$\begin{aligned}&\text{is-m-other-st}(ST) \wedge \alpha(\xi) \neq \alpha(\xi') \supset\\&\quad(\exists\beta,\text{un})(\alpha = \beta\circ\text{un}\circ\text{s-dn} \wedge \text{is-un}(\text{un}) \wedge \text{is-other-den}(\text{un}\circ\text{s-den}(\xi)))\end{aligned}$$

Property 1-2: This property shows that the only state components which can be changed, except by deletion and reinsertion, are CTR, ABN and those elements of DN which satisfy is-other-den.

$$\alpha(\xi^i) \neq \alpha(\xi^j) \wedge i < j \wedge \neg(\exists k)(i \leq k \leq j \wedge \alpha(\xi^k) = \Omega) \supset$$
$$(\alpha = \text{s-abn} \vee (\exists k)(\alpha = \text{s-ctr}\circ\text{elem}(k)\circ\text{s-bl}) \vee$$
$$(\exists\beta,\text{un})(\alpha = \beta\circ\text{un}\circ\text{s-dn} \wedge \text{is-other-den}(\text{un}(DN^i))))$$

A proof can be constructed by observing that (assuming 1-1) the only instructions which can change the rémainder of the state components are inst-bl and inst-proc. These instructions can only be invoked to create a new element i of BL if the preceding BL is of length i-1. Therefore the changing of the other values of the state components would follow a state in which they were Ω.

Property 1-3: This property shows that a label or procedure denotation, whose unique name is contained in the portion of text currently pointed to, has the same value as when it was installed.

$$\text{is-un}(\alpha\circ PTR_k^i(TX_k^i)) \wedge (\text{is-lab-den} \vee \text{is-proc-den})((\alpha\circ PTR_k^i(TX_k^i))(DN^i)) \supset$$
$$(\alpha\circ PTR_k^i(TX_k^i))(DN^i) = (\alpha\circ PTR_k^j(TX_k^j))(DN^j)$$
$$\text{where } j = \underset{j}{MAX}\ \ j \leq i \wedge \text{length}(BL^j) = \text{len-part}(\alpha\circ PTR_k^i(TX_k^i)) \wedge CTR^j = 0$$

A proof can be constructed by induction on states of the proposition that for all unique names in the current text parts there exists no earlier state with a BL shorter than that which existed when they were introduced. This is ensured by the exact bracketing of epilogue instructions with instructions which extend BL. Application of 1-2, both to show old text components are unchanged and to show such denotation entries are unchanged, concludes the argument.

Property 1-4: This property shows that the portion of the text component which comprises a procedure denotation has not changed since that denotation was installed.

$$\text{is-un}(\alpha\circ PTR^i(TX^i)) \wedge \text{is-proc-den}(\alpha\circ PTR^i(TX^i)(DN^i)) \supset$$
$$(\alpha\circ PTR^i(TX^i)(DN^i))(TX^i) = (\alpha\circ PTR^i(TX^i)(DN^j))(TX^j)$$
$$\text{where } j = \underset{j}{MAX}\ \ j \leq i \wedge \text{length}(BL^j) = \text{len-part}(\alpha\circ PTR^i(TX^i)) \wedge CTR^j = 0$$

A proof would rely on the fact that the denotations have already been shown to be equal (1-3). The fact that procedure attribute parts of the text are unchanged unless denotation is re-installed follows from the observation that B10 and B11 can only extend or change text within the block given by their PTR component.

Property 1-5: [1] This property shows that for any state, the embracing or calling block, of a block pointed to by (the PTR component of) an element of BL, is pointed to by (the PTR component of) an earlier element of BL.

$$1 < i \leq \text{length}(BL) \supset$$
$$(\exists j)(j < i \wedge ((\exists k,bl)(PTR_i(TX_i) = \text{mod-b}(\text{elem}(k)\circ \text{s-sp}\circ PTR_j(TX_j),bl))$$
$$\vee\ (\exists k,\text{un-}l)(PTR_i(TX_i) = \text{mod-p}(\text{s-atr}\circ\text{elem}(k)\circ\text{s-dp}\circ PTR_j(TX_j),\text{un-}l))))$$

A proof of the property by induction on BL can be constructed (only inst-bl and inst-prc can extend BL, the former case is trivial and the latter relies on 1-4).

Property 1-6: This property shows that, under the assumption of a proper program, for any block pointed to by (the PTR component of) an element of BL, all of the names not contained in a nested block have been changed to ones satisfying is-un.

$$1 \leq i \leq \text{length}(BL) \wedge \text{is-dir-cont}(\alpha,PTR_i(TX_i)) \wedge \text{is-id}'(\alpha\circ PTR_i(TX_i)) \supset$$
$$\text{is-un}(\alpha\circ PTR_i(TX_i))$$

A proof by induction, of the property that the only variables within PTR(TX) which are not unique names are redeclared in an intervening block, can be constructed. (Use is made of 1-5 and the definition of a proper program.) Coupled with the definition of is-dir-cont (B22) this property is enough to show that the required property holds.

Property 1-7: [2] This property shows that, with proper programs, any goto statement contained in a block pointed to by (the PTR component of) an element of BL references a label which was introduced in a block which is pointed to by (the PTR component of) an earlier element of BL.

$$1 < i \leq \text{length}(BL) \wedge \text{is-m-goto}\ (\text{elem}(j)\circ\text{s-sp}\circ PTR_i(TX_i)) \supset$$
$$(\exists k)(k \leq i \wedge PTR_k = \text{s-b}\circ\text{elem}(j)\circ\text{s-sp}\circ PTR_i(TX_i)(DN))$$

A proof follows from the following observations: the assumption of proper program assures that the identifier has been changed to a unique name (1-6); the label denotation has not changed since it was intalled (1-3); the installing block is still active (1-5).

1) This property would be lost by the introduction of procedure variables which could be used without restriction (1-3 and 1-4 would not substantiate the pre-conditions).

2) This property is lost with the unrestricted use of label variables (1-3 applies only to constants).

<u>Lemma 1-8</u>: This lemma shows that if, in some block which is not yet active, an identifier has already been changed to a unique name, then the original identifier could not have been a local name of that block.

a) $\text{is-m-block}(st) \wedge \text{is-un}(\alpha \circ \text{s-sp} \circ st) \supset$
$\neg\,\text{is-decl}(\text{id-part}(\alpha \circ \text{s-sp} \circ st), st)$

b) $\text{is-m-call}(st) \wedge$
$\text{is-un}(\alpha \circ \text{s-body}(\text{s-nm}(st)(DN)(TX))) \supset$
$\neg\,\text{is-parmd}(\alpha \circ \text{s-body}(\text{s-nm}(ST)(DN))(TX)), (\text{s-nm}(st(DN))(TX))$

where $st = \text{elem}(i) \circ \text{s-sp} \circ PTR(TX)$

Proof: We prove a) by contradiction:

1. Assume PTR_o, i_o, TX_o and α_o represent a counter example
let $un_o = \alpha_o \circ \text{s-sp} \circ \text{elem}(i_o) \circ \text{s-sp} \circ PTR_o(TX_o)$

2 $\text{is-m-block}(\text{elem}(i_o) \circ \text{s-sp} \circ PTR_o(TX_o))$ — 1

3 $\text{is-un}(un_o)$ — 1

4 $\text{is-decl}(\text{id-part}(un_o), \text{elem}(i_o) \circ PTR_o(TX_o))$ — 1

5 Since the original text has no unique names, un_o must have been generated. — 3,B0

6 Inspection of the machine shows only mod-b and mod-p can change text — B10,B11

7 Assume un_o was inserted by mod-b

8 $(\exists \alpha,\beta)(\text{is-m-block}(\alpha(TX_o)) \wedge \beta \circ \alpha = \text{elem}(i_o) \circ \text{s-sp} \circ PTR_o \wedge \beta \neq I \wedge$
$\text{is-decl}(\text{id-part}(un_o), \alpha(TX_o)) \wedge \neg\,\text{is-red}(\text{id-part}(un_o), \alpha_o \circ \text{s-sp} \circ \beta, \alpha(TX_o)))$ — 7,B15

9 $(\exists \alpha,\beta)(\beta \circ \alpha = \text{elem}(i_o) \circ \text{s-sp} \circ PTR_o \wedge \beta \neq I \wedge$
$\neg(\exists \alpha',\beta')(\alpha' \circ \beta' = \alpha_o \circ \text{s-sp} \circ \beta \wedge \beta' \neq I \wedge$
$\text{is-intr}(\text{id-part}(un_o), \beta' \circ \alpha(TX_o)))$ — 8,B18

In particular $\alpha' = \alpha_o \circ \text{s-sp}$ and $\beta' = \beta$ cannot be such a decomposition:

10 $(\exists \alpha,\beta)(\beta \circ \alpha = \text{elem}(i_o) \circ \text{s-sp} \circ PTR_o \wedge \beta \neq I \wedge$
$\neg\,\text{is-decl}(\text{id-part}(un_o), \beta \circ \alpha(TX_o)))$ — 9,B19

11 $\neg\,\text{is-decl}(\text{id-part}(un_o), \text{elem}(i_o) \circ \text{s-sp} \circ PTR_o)$

12 But 11 and 4 contradict.

13 A similar contradiction arises if mod-p is assumed to have inserted un_o.
Thus no un_o satisfying 1 can occur. — 12,13,5

This concludes the proof of a), a similar proof would justify b).

Lemma 1-9: This lemma shows that if a block becomes active, any of the identifiers which had already been changed to a unique name is unchanged by the activation step.

a) $1 < i < \text{length}(BL) \wedge \text{is-m-block}(ST_i) \supset$
$(\text{is-un}(\alpha(ST_i)) \supset \alpha \circ PTR_{i+1}(TX_{i+1}) = \alpha(ST_i))$

b) $1 < i < \text{length}(BL) \wedge \text{is-m-call}(ST_i) \supset$
$(\text{is-un}(\alpha \circ \text{s-body}(\text{s-nm}(ST_i)(DN))) \supset$
$\alpha \circ PTR_{i+1}(TX_{i+1}) = \alpha \circ \text{s-body}(\text{s-nm}(ST_i)(DN)))$

Proof of a):

1	$PTR_{i+1} = \text{elem}(CTR_i) \circ \text{s-sp} \circ PTR_i$	B10
2	$TX_{i+1} = \mu(TX_i; \langle \text{elem}(CTR_i) \circ \text{s-sp} \circ PTR_i : \text{mod-b}(ST_i, BL_i) \rangle)$	B10
3	$\neg \text{is-id}(\alpha(ST_i))$	
4	$\alpha \circ \text{mod-b}(ST_i, BL_i) = \alpha(ST_i)$	3,B15
5	$\alpha \circ PTR_{i+1}(TX_{i+1}) = \alpha(ST_i)$	2,4

This concludes the proof of a), a similar proof would justify b).

2. IMPLEMENTATION USING A SEARCH OF THE STATIC CHAIN

The model described in section 1.2 does not provide us with a practical implementation, if for no other reason than that the physical modification of the text is very difficult. However, derivable from that model is an implementation which creates a directory in each dynamic area relating all known names to their unique names (this is the environment mechanism used in /7/). Such models would be very efficient in referencing variables but the amount of work involved in updating these directories during block activation and exit would be prohibitive, as would the amount of space required for their storage.

As mentioned in the discussion of the language, the recursive nature of procedure calls prevents, in general, a fixed association of names with addresses (i.e. selectors to DN in the base model) being made at compile time. We are therefore forced to look at alternative ways of storing variables such that their access is simple yet the bookkeeping required to facilitate this access is not excessive.

All of the subsequent models adopt the idea of storing values in storage obtained dynamically with the dynamic area, leaving us with the problem of how to locate the appropriate dynamic area for a name. Two basically different ways of performing this location are considered below: searching, which entails the chaining together of the dynamic areas in some way, then locating the required area by inspecting elements of the chain until one with some specified property is found; via a <u>display</u> (i.e. a vector of pointers to the dynamic areas) which requires that the identifiers of the text have been supplemented with an index to this vector specifying which element will be assumed to point at the relevant dynamic area. Again, because of the dynamic nature of recursive procedures and the limitation that a prepass must assign a constant index, such a display must be updated dynamically. A chaining of the dynamic areas may well be used to simplify such updating. Although the search methods tend to require a minimum of housekeeping, the indirectness of references is rarely acceptable in a practical implementation. Their description below is provided more to supply stepping stones for the reader en route to the rather indirect display methods.

The search implementation described in section 2.1 chains the dynamically obtained storage areas for blocks together on the principle that each such area is chained to that area belonging to the block which lexicographically contained it in the original text. Because of the role it fulfils, the dynamic area corresponding to the statically containing block is called the <u>environmentally preceding activation</u> (E.P.A.). The chain made up by following the E.P.A. pointers is referred to as the <u>static chain</u> (for a fuller description see /5/). It is a property of the language under consideration that the containing block of any active block must itself be active (see 1-5). Therefore, the chains all lead back to the first dynamic area (i.e. that of the program). This is the only one lacking a pointer.

The values of arguments must now be accessible from their formal parameter names. This is achieved by a new type of denotation entry (is-parm-den) which contains the necessary information about the argument (note that this will include the index of the introducing dynamic area).

The updating of the static chain for blocks is trivial since they can only be invoked from their E.P.A. The maintenance of the chain for procedures, whether activated directly or via a parameter to which they were passed as an argument, requires that the knowledge of the block in which they were declared is accessible.

The idea of using a static chain is introduced by Dijkstra in /4/, and a very clear description is contained in /5/. The correctness of the method is the subject of section 2.3.

2.1 The Model

This section contains a model which realizes the search process just outlined. This introduction is confined to identifying the main differences from the model of section 1.2.

The procurement of storage dynamically is modelled by the use of denotation components (DN) for each of the block local elements (BL). In addition a pointer (EPA) is added to each such element which points to the preceding element in the required chain. The retrieval of all information using the original text names requires that denotation entries be made for all parameters.

The non-modification of the text permits, and is clearly shown by, its storage as a global component. The fact that entering a procedure will, therefore, require an entirely different modification of the PTR component leads to the possibility of CTR becoming Ω .

The information that would have previously been obtained by applying a dynamically formed unique name to the global denotation is now found as follows: The block in which a variable was declared is found by searching the static chain for the first element where this name occurs in the denotation component; the denotation of the variable is found by applying its name to the denotation element of the block found as above. The denotations of parameters are made to contain this information about the variable which was passed as an argument, and the search mechanisms are extended to anticipate such entries.

The fact that the original names are used as selectors to the local denotations relies on their local uniqueness (i.e. only one block is handled by each denotation). This may be contrasted with the situation in section 4.1.

2.1.1 State

```
S1    is-state = (<s-bl:is-bl-list>,
                  <s-tx:is-block>,
                  <s-abn:(<s-b:is-int>,<s-dcl:is-int>) v is-Ω>,
                  <s-c:is-c>)

S2    is-bl = (<s-epa:is-int v is-Ω>,<s-ptr:is-sel>,<s-ctr:is-int v is-Ω>,
               <s-dn:({<id:is-den> || is-id(id)})>)

S3    is-den = is-proc-den v is-lab-den v is-parm-den v is-other-den

S4    is-proc-den = is-int

S5    is-lab-den = is-int

S6    is-parm-den = (<s-b:is-int>,<s-dcl:is-int>)

S7    is-other-den = ...
         N.B. is-other-den must never satisfy is-Ω

S8    is-c = see /2/
```

Abbreviatons used:

```
BL = s-bl(ξ)
TX = s-tx(ξ)
ABN = s-abn(ξ)
B = last(BL)
EPA = s-epa(B)
PTR = s-ptr(B)
CTR = s-ctr(B)
DN = s-dn(B)
P = CTR ≠ Ω ——→ elem(CTR)∘s-sp∘PTR
        T ——→ s-body∘PTR
ST = P(TX)
cur(bl) = elem(length(bl))∘s-bl
```

The initial state ξ_o for any t_o such that is-program(t_o) is

S0 $\xi_o = \mu_o$(<s-bl:[μ_o(<s-ptr:I>,
<s-ctr:0>)]>,
<s-tx:μ_o(<s-sp:[t_o]>,<s-dp:[]>)>,
<s-c:$\underline{\text{int-next-st}}$>)

2.1.2 State transition function

```
S9     int-next-st =

          CTR < length(s-sp∘PTR(TX)) ∧ ABN = Ω ——→ int-next-st;
                                                     int-st;
                                                       step-ctr

          s-b(ABN) = length(BL) ——→ int-next-st;
                                      int-st;
                                        set-ctr

          T ——→ null

S10    int-st =

          is-block(ST) ——→ epilogue;
                             int-next-st;
                               inst-bl

          is-call(ST) ——→ epilogue;
                            epilogue;
                              int-next-st;
                                inst-bl;      length(s-ap(ST))
                                  inst-proc(  LIST   d(elem(i,s-ap(ST)),length(BL),BL),
                                              i=1
                                          s-atr∘elem(d)∘s-dp∘(s-ptr(elem(b,BL))),b)

          is-goto(ST) ——→ s-abn:d(ST,length(BL),BL)
                            ⋮

          where b = d-b(s-nm(ST),length(BL),BL)
                d = d-d(s-nm(ST),length(BL),BL)

S11    inst-bl =

          s-bl:BL⁀[μ₀(<s-epa:length(BL)>,
                      <s-ptr:P>,
                      <s-ctr:0>,
                      <s-dn:μ₀({<s-id∘elem(i)∘s-dp(ST):i> |
                               ¬is-other-atr(s-atr∘elem(i)∘s-dp(ST))} ∪ ...)>)]

S12    inst-proc(arg-l,s,epa) =

          s-bl:BL⁀[μ₀(<s-epa:epa>,
                      <s-ptr:s>,
                      <s-dn:μ₀({<elem(j)∘s-pp∘s(TX):elem(j,arg-l)>|
                                                1 ≤ j ≤ length(arg-l)})>)]
```

S13 <u>step-ctr</u> = s-ctr∘cur(BL):CTR + 1

S14 <u>set-ctr</u> = s-ctr∘cur(BL):s-atr∘elem(s-dcl(ABN))∘s-dp∘PTR(TX)
s-abn:Ω

S15 <u>epilogue</u> = cur(BL):Ω

S16 d-b(id,n,bl) = d(id,n,bl) = $\Omega \longrightarrow 0$
T $\longrightarrow$ s-b(d(id,n,bl))

S17 d-d(id,n,bl) = s-dcl(d(id,n,bl))

S18[1] d(id,n,bl) = n $\leq$ 1 $\longrightarrow \Omega$
is-parm-den(id∘s-dn∘elem(n,bl)) $\longrightarrow$ id∘s-dn∘elem(n,bl)
id∘s-dn∘elem(n,bl) $\neq \Omega \longrightarrow \mu_0$(<s-b:n>,<s-dcl:id∘s-dn∘elem(n,bl)>)
T $\longrightarrow$ d(id,s-epa∘elem(n,bl),bl)

<u>2.2 Properties</u>

This section contains some properties (2-1 to 2-3) of the above model, which will be used in the subsequent proofs. First some functions and predicates are defined:

S19 is-intr-s(id,bl) = is∘s-dn(last(bl)) $\neq \Omega$

S20 is-refbl(α,ptr,t) = $(\exists\beta)(\beta$∘ptr = $\alpha \wedge$ count-blk(ptr,t) = count-blk(α,t))

S21 count-blk(sel,t) =

$(\exists\alpha$,i)(sel = α∘elem(i)∘s-sp $\wedge$ is-block(elem(i)∘s-sp(t)) $\longrightarrow$
count-blk(α,elem(i)∘s-sp(t)) + 1
$(\exists\alpha$,i)(sel = α∘s-atr∘elem(i)∘s-dp $\wedge$ is-proc-atr(s-atr∘elem(i)∘s-dp(t)) $\longrightarrow$
count-blk(α,s-atr∘elem(i)∘s-dp(t)) + 1
$(\exists\alpha)$(sel = α∘s-body $\wedge$ is-block(s-body(t)) $\longrightarrow$ count-blk(α,s-body(t)) + 1
T $\longrightarrow$ 1

where the unbound α's and i's on the right of the conditional expression are unique values which satisfy the left hand side.

S22 nth-blk(i,sel,t) = $(\iota\alpha)$((is-block $\vee$ is-proc-atr)(α(t)) $\wedge$
count-blk(α,t) = i $\wedge$ $(\exists\beta)(\beta$∘α = sel))

1) Notice that this function, and all subsequent functions which operate on lists, do not rely on any element of the list whose index is higher than that passed as argument. This property will be used below, in the proofs, without reference.

S23 nth-proc(i,sel,t) =

$$\text{count-proc}(sel,t) = 1 \longrightarrow I$$
$$T \longrightarrow (\iota\alpha)(\text{is-proc-atr}(\alpha(t)) \wedge \text{count-proc}(\alpha,t) = i \wedge (\exists\beta)(\beta\circ\alpha = sel))$$

S24 count-proc(sel,t) =

$$\text{card}(\{\text{is-proc-atr}(\text{nth-blk}(i,sel,t)(t)) \mid 1 \le i \le \text{count-blk}(sel,t)\}) + 1$$

S25 cont-proc(sel,t) =

$$\text{nth-proc}(\text{count-proc}(sel,t),sel,t)$$

S26 find-epa(n,bl) = $n = 0 \longrightarrow \text{length}(bl)$
$$T \longrightarrow \text{find-epa}(n-1,\text{front}(\text{s-epa}(\text{last}(bl)),bl))$$

Property 2-1: This property is identical with 1-2 but applies to the machine of section 2.1. Its proof again relies on 1-1.

Property 2-2: The first part of this property specifies the relation between the pointer components of successive elements of the static chain. The second part extends this to a relation between arbitrary elements of the static chain.

a) $$EPA_k \ge 1 \supset (\exists\alpha)(PTR_k = \alpha\circ PTR_{EPA_k} \wedge ((\exists i)(\alpha = \text{elem}(i)\circ\text{s-sp}) \vee (\alpha = \text{s-body}) \vee (\exists i)(\alpha = \text{s-atr}\circ\text{elem}(i)\circ\text{s-dp})))$$

b) $$j > k \supset (PTR_i = \text{nth-blk}(j,\alpha,TX) \supset PTR_{\text{find-epa}(k,BL_i)} = \text{nth-blk}(j-k,\alpha,TX))$$

A proof of a) follows from S11, S11, S12 respectively (2-1 supports the claim that the EPA and PTR components are unchanged). A proof of b) follows from S26.

Property 2-3: This property states that if two blocks, which are contained in the same procedure and are such that one is not contained in the other, are pointed to by elements of BL, then there must be an intervening element of BL which points to their containing procedure.

$$i < j \le \text{length}(BL) \wedge \text{cont-proc}(PTR_i,TX) = \text{cont-proc}(PTR_j,TX) \wedge \neg(\exists\alpha,\beta)(\alpha\circ PTR_i = \beta\circ PTR_j) \supset (\exists k)(i < k < j \wedge PTR_k = \text{cont-proc}(PTR_i,TX))$$

A proof by contradiction can be constructed based on 2-2b, S11 and S12.

2.3 Justification

This section contains a proof that the interpreter of section 2.1 is "equivalent" to that given as the definition of the language in 1.2. As discussed in the introduction to section 1.2, we shall attain this objective by showing that all information retrievable from the denotation directory of the defining interpreter is also obtainable from the alternative interpreter.

The framework to be used for this proof will be the so-called "twin machine" method of /1/. This method can be explained by an analogy: Suppose some organisation is changing an internal recording system such as its accounts, it will probably elect to go through a period of running the new system in parallel with that already established. The mechanism required for the new system being installed so as not to interfere (logically) with the results of the established. When a certain level of confidence has been created, that the new system yields the same results as the established (and "therefore" correct) one, the mechanism for the old system is discarded and the new system becomes established.

Much the same approach is followed by our "twin machine" proof of the correctness of the system for access to variables. The first step is the installation of the state components which the alternative system will require. This change is made in such a way as to not change the results of the established system, in particular the defining system remains the one used for all internal references [1]. A proof is then given that, in any state of this "parallel operation" machine, a reference mechanism utilizing the components added above will yield the same result as the defining mechanism. The anticipated transition, which will include changing the reference method employed by the internal housekeeping operations (e.g. access to procedures), is then made. Finally, any state elements not required by the new method are discarded. Section 2.3.1 defines the twin machine.

The other idea behind this proof is the use of the freedom permitted by the language definition in the definition of the unique name generator. The only constraint of the definition was that the function should be one-one from identifiers and integers to unique names. In constructing the twin machine we define a particular function which has the property that the generated unique names are selectors leading to exactly the local denotations whose insertion is part of the mechanism for the new retrieval method.

[1] In this comment lies the key to the advantage of this style of proof construction: were the proof written based on the two systems unconnected to each other, we should be compelled to show that they follow the same course (e.g. executed the same procedures) before we could address the argument that at comparable points they yield the same result. Whereas a proof using the twin method is simplified by having both models "chained" together by virtue of the use of only one housekeeping system. Thereafter, when the equivalence of the reference techniques is established, the new method may be utilized for all references including those for housekeeping.

The correctness of the search model follows from the observation that the portions of text of the base model pointed to by the elements of BL are formed by successively modifying the base text in two ways: the names introduced in embracing blocks are changed to unique names as the blocks become active; all formal parameters are replaced by the unique names in the argument list (see 1-5, 1-4). The static chain traces through exactly those elements of BL pointing to the embracing blocks (2-2) and parameters are handled in a way equivalent to their insertion. The successive modification and the search from the innermost block both have the property of yielding the deepest occurence. Section 2.3.2 contains a proof by induction of the property that the search locates the same denotation as that pointed to by the unique name. Further, since the text components differ, the same induction is used to show that all unmodified names are equal. The induction for blocks is trivial, that for procedures is complicated by having to show that no changes have been made to the relevant components since the block was active in which the induction hypothesis was used (i.e. that containing the procedure declaration).

2.3.1 Twin model

There are two areas where this model differs from that of section 1.2, neither of which affects the result. First, the addition of new, unused, state components (all suffixed by -s), whose relation to the machine of section 2.1 should be obvious. The second change is the definition of a particular function (T22) for un which satisfies the property required in B17. The function chosen yields selectors which can be applied directly to the state to give the denotation (s-den-b is applied to this composite denotation to yield the base rather than the new "-s" component).

The selectors are generated so as to point to the local denotation components for the block local element corresponding to the block or procedure which introduced them. The global denotation component has therefore disappeared in favour of these local denotations. Furthermore the function un creates unique names whose first element is the original identifier (this observation justifies the use of the original identifiers in the updating of the local denotations in T15, T16).

Notice that these local denotations are deleted, with the rest of the block local list element, by epilogue on exit from their active scope. Since any variables in such a denotation are no longer referencable (and their counterparts in the global denotation would have been eligible for overwriting), no change has been made to the outcome.

2.3.1.1 State

T1 is-state = (<s-bl:is-bl-list>,<s-abn:is-lab-den ∨ is-Ω>,<s-c:is-c>,
<s-tx-s:is-block>)

T2 is-bl = (<s-tx:is-m-block>,<s-ptr:is-sel>,<s-ctr:is-int>,
<s-dn:({<id:(<s-den-b:is-den>,<s-den-s:is-den-s>)> || is-id(id)})>,
<s-epa-s:is-int>,<s-ptr-s:is-sel>,<s-ctr-s:is-int ∨ is-Ω>)

T3 is-den = is-proc-den ∨ is-lab-den ∨ is-other-den

T4 is-proc-den = is-sel

T5 is-lab-den = (<s-b:is-sel>,<s-dcl:is-int>)

T6 is-other-den = ...

T7 is-den-s = is-proc-den-s ∨ is-lab-den-s ∨ is-parm-den-s ∨ is-other-den-s

T8 is-proc-den-s = is-int

T9 is-lab-den-s = is-int

T10 is-parm-den-s = (<s-b:is-int>,<s-dcl:is-int>)

T11 is-other-den-s = ...
N.B. must not satisfy is-Ω

T12 is-c = see /2/
is-m-block etc. as in 1.1 except that all occurences of is-id have been changed to is-id'.

Abbreviations used:
BL = s-bl(ξ)
DN = s-dn(ξ)
ABN = s-abn(ξ)
B = last(BL)
TX = s-tx(B)
PTR = s-ptr(B)
CTR = s-ctr(B)
P = elem(CTR)∘s-sp∘PTR
ST = P(TX)
CTRs = s-ctr-s(B)
PTRs = s-ptr-s(B)

cont'd

```
TXs = s-tx-s($\xi$)
EPAs = s-epa-s(B)
Ps = CTRs $\neq \Omega$ $\longrightarrow$ elem(CTRs)$\circ$s-sp$\circ$PTRs
     T $\longrightarrow$ s-body$\circ$PTRs
STs = Ps(TXs)
cur(bl) = elem(length(bl))$\circ$s-bl
```

The initial state ξ_o for any t_o such that is-program(t_o)

```
T0   $\xi_o$ = $\mu_o$(<s-bl:[$\mu_o$(<s-tx:$\mu_o$(<s-sp:[$t_o$]>,<s-dp[]>)>,
                         <s-ptr:I>,
                         <s-ctr:0>,
                         <s-ptr-s:I>,
                         <s-ctr-s:0>)]>,
               <s-c:int-next-st>,
               <s-tx-s:$\mu_o$(<s-sp:[$t_o$]>,<s-dp:[]>)>)
```

2.3.1.2 State transition function

```
T13  int-next-st =
       CTR < length(s-sp$\circ$PTR(TX)) $\wedge$ ABN = $\Omega$ $\longrightarrow$ int-next-st;
                                                  int-st;
                                                    step-ctr

       s-b(ABN) = PTR $\longrightarrow$ int-next-st;
                           int-st;
                             set-ctr
       T $\longrightarrow$ null

T14  int-st =
       is-m-block(ST) $\longrightarrow$ epilogue;
                           int-next-st;
                             inst-bl

       is-m-call(ST) $\longrightarrow$ epilogue;
                          epilogue;
                            int-next-st;
                              inst-bl;
                                inst-proc(s-ap(ST),s-den-b$\circ$s-nm(ST)($\xi$))

       is-m-goto(ST) $\longrightarrow$ s-abn:s-den-b(ST($\xi$))
                      .
                      .
                      .
```

```
T15   inst-bl =

        s-bl:BL⌢[μ_0(<s-tx:μ(TX;<P:mod-b(ST,BL)>)>,
                     <s-ptr:P>,
                     <s-ptr-s:Ps>,
                     <s-ctr:0>,
                     <s-epa:length(BL)>,
                     <s-dn:μ_0({<s-id∘elem(i,s-dp(ST)):
                                 μ_0(<s-den-b:s-atr∘elem(i)∘s-dp∘P>,
                                     <s-den-s:i>)> | is-m-proc-atr(s-atr(elem(i,s-dp(ST))))}
                             ∪{<s-id∘elem(i,s-dp(ST)):
                                 μ_0(<s-den-b:μ_0(<s-b:P>,<s-dcl:i>)>,
                                     <s-den-s:i>)> |
                                        is-m-lab-atr(s-atr(elem(i,s-dp(ST))))} ∪ ...)>)]

T16   inst-proc(arg-l,s) =

        s-bl:BL⌢[μ_0(<s-tx:μ(TX;<P:s-body(mod-p(s(TX),arg-l))>)>,
                     <s-ptr:PTR>,
                     <s-ctr:CTR>,
                     <s-ptr-s:s-atr∘elem(dec)∘s-dp∘s-ptr-s(elem(ind,BL))>,
                     <s-epa-s:ind>,
                     <s-dn:μ_0({<elem(j,s-pp∘s(TX)):
                                  μ_0(<s-den-s:d(id-part(elem(j,arg-l)),length(BL),BL)>)>|
                                                         1 ≤ j ≤ length(arg-l)})>)]

      where: ind = (ιi)((∃j)(s-atr∘elem(j)∘s-dp∘s-ptr(elem(i,BL)) = s))
             dec = (ιi)((∃α)(s-atr∘elem(i)∘α = s))

T17   step-ctr = s-ctr∘cur(BL):CTR + 1
                 s-ctr-s∘cur(BL):CTRs + 1

T18   set-ctr = s-ctr∘cur(BL):s-atr∘elem(s-dcl(ABN))∘s-dp∘PTR(TX)
                s-ctr-s∘cur(BL):s-atr∘elem(s-dcl(ABN))∘s-dp∘PTR(TX)
                s-abn:Ω

T19   epilogue = cur(BL):Ω

T20   mod-b(blk,bl) =

        μ(blk;{<α:und(α(blk),length(bl))> | is-id(α(blk)) ∧ is-decl(α(blk),blk) ∧
                                                  ¬is-red(α(blk),α,blk)})
```

T21 $\text{mod-p}(\text{prc},\text{arg-1}) =$

$\mu(\text{prc};\{<\alpha:\text{elem}(i,\text{arg-1})> \mid$
$\text{is-id}(\alpha(\text{prc})) \wedge \text{elem}(i)\circ\text{s-pp}(\text{prc}) = \alpha(\text{prc}) \wedge \neg\text{is-red}(\alpha(\text{prc}),\alpha,\text{prc})\})$

T22 $\text{un}(\text{id},n) = \text{id}\circ\text{s-dn}\circ\text{elem}(n+1)\circ\text{s-bl}$

T23 $\text{is-red}(\text{id},\text{sel},t) = (\exists\alpha,\beta)(\text{sel} = \alpha\circ\beta \wedge \beta \neq I \wedge \text{is-intr}(\text{id},\beta(t)))$

T24 $\text{is-intr}(\text{id},t) = \text{is-m-block}(t) \longrightarrow \text{is-decl}(\text{id},t)$
$\text{is-m-proc-atr}(t) \longrightarrow \text{is-parmd}(\text{id},t)$
$T \longrightarrow F$

T25 $\text{is-decl}(\text{id},\text{blk}) = (\exists i)(\text{s-id}\circ\text{elem}(i)\circ\text{s-dp}(\text{blk}) = \text{id})$

T26 $\text{is-parmd}(\text{id},\text{prc}) = (\exists i)(\text{elem}(i)\circ\text{s-pp}(\text{prc}) = \text{id})$

T27 $\text{d-b}(\text{id},n,\text{bl}) = \text{d}(\text{id},n,\text{bl}) = \Omega \longrightarrow 0$
$T \longrightarrow \text{s-b}(\text{d}(\text{id},n,\text{bl}))$

T28 $\text{d-d}(\text{id},n,\text{bl}) = \text{s-dcl}(\text{id},n,\text{bl})$

T29 $\text{d}(\text{id},n,\text{bl}) = n \leq 1 \longrightarrow \Omega$
$\text{is-parm-den-s}(\text{s-den-s}\circ\text{id}\circ\text{s-dn}\circ\text{elem}(n,\text{bl})) \longrightarrow$
$\text{s-den-s}\circ\text{id}\circ\text{s-dn}\circ\text{elem}(n,\text{bl}))$
$\text{s-den-s}\circ\text{id}\circ\text{s-dn}\circ\text{elem}(n,\text{bl}) \neq \Omega \longrightarrow$
$\mu_0(<\text{s-b}:n>,<\text{s-dcl}:\text{s-den-s}\circ\text{id}\circ\text{s-dn}\circ\text{elem}(n,\text{bl})>)$
$T \longrightarrow \text{d}(\text{id},\text{s-epa-s}(\text{elem}(n,\text{bl})),\text{bl})$

We can now, given T22, specify actual functions for id-part and len-part (see B23,B24)

T30 $\text{id-part}(\text{un}) = (\iota\,\text{id})(\text{is-id}(\text{id}) \wedge (\exists\alpha)(\text{un} = \text{id}\circ\alpha))$

T31 $\text{len-part}(\text{un}) = (\iota\,i)(\text{un} = \text{id-part}(\text{un})\circ\text{s-dn}\circ\text{elem}(i)\circ\text{s-bl})$

2.3.2 Proof

Theorem 2-4 shows a relation between the unique names and the new state components which is stated with the aid of the following abbreviation.

T32 $f(\alpha,\text{BL}) = (\alpha\circ\text{PTRs}(\text{TXs}))\circ\text{s-dn}\circ\text{elem}(\text{d-b}(\alpha\circ\text{PTRs}(\text{TXs}),\text{BL}))\circ\text{s-bl}$

Corollaries 2-5 to 2-8 and Theorem 2-9 show how this relation justifies the correctness of the model of section 2.1.

<u>Theorem 2-4</u>: For any state ξ:

$is\text{-}un(\alpha \circ PTR(TX)) \supset \alpha \circ PTR(TX) = f(\alpha, BL)$

Proof: [1] An inductive proof is given that:

$1 \le i \le length(BL) \supset$

1 $is\text{-}un(\alpha \circ PTR_i(TX_i)) \supset \alpha \circ PTR_i(TX_i) = f(\alpha, BL_i) \wedge$

2 $is\text{-}id(\alpha \circ PTR_i(TX_i)) \supset \alpha \circ PTR_i(TX_i) = \alpha \circ PTRs_i(TXs)$

Basis for i = 1

3	$is\text{-}block(TX_1)$	TO,1-2
4	$PTR_1(TX_1) = I(TX_1)$	TO,1-2
5	$\neg(\exists \alpha)(is\text{-}un(\alpha \circ PTR_1(TX_1)))$	3,4
6	$PTRs_1(TXs) = I(TXs)$	TO
7	$TX_1 = TXs$	TO
8	$(\forall \alpha)(\alpha \circ PTR_1(TX_1) = \alpha \circ PTRs_1(TXs))$	3,4,6,7

Thus the required properties hold for any initial element 5,8

Induction from BL_i to BL_{i+1} by <u>inst-bl</u>

let $nm = \alpha(ST_i)$

$nm' = \alpha \circ PTR_{i+1}(TX_{i+1})$

$ids = \alpha(STs_i)$

$ids' = \alpha \circ PTRs_{i+1}(TXs)$

Property 1:

9 $is\text{-}un(nm')$

1) The properties given in 1-2 and 2-1 apply to the twin machine and will be used without separate proof. Further we shall use the obvious result that, after <u>inst-bl</u>, the CTR and CTRs components are equal, without proof.

Two case distinctions are made:

10	case: $\neg$is-un(nm)	
11	is-id(nm)	9,10,T15,T20
12	$(\exists\, i)$(s-id$\circ$elem(i)$\circ$s-dp(ST) = nm)	9,10,T15,T20,T24
13	$\alpha\circ$elem(CTR_i)$\circ$s-sp$\circ PTR_i(TX_i)$ = $\alpha\circ$elem(CTR_i)$\circ$s-sp$\circ PTRs_i$(TXs)	11,IH2
14	nm = ids	13
15	nm = ids'	13,T15
16	nm' = un(nm,length(BL_i))	11,12,T20
17	nm' = nm$\circ$s-dn$\circ$elem(length(BL_i) + 1)$\circ$s-bl	16,T22
18	nm' = nm$\circ$s-dn$\circ$elem(length(BL_{i+1}))$\circ$s-bl	17,T15
19	d-b(ids',BL_{i+1}) = d-b(nm,BL_{i+1})	15
20	d-b(ids',BL_{i+1}) = length(BL_{i+1})	19,12,T15,T27,T29
21	f(α,BL_{i+1}) = nm$\circ$s-dn$\circ$elem(length(BL_{i+1}))$\circ$s-bl	T32,15,20
22	nm' = f(α,BL_{i+1})	21,18

Which concludes the case given in 10.

23	case: is-un(nm) let idp = id-part(nm)	
24	$\neg(\exists\, i)$(s-id$\circ$elem(i)$\circ$s-dp(ST_i) = idp)	23,1-8a,T24
25	$\alpha\circ ST_i$ = f($\alpha\circ$elem(CTR_i)$\circ$s-sp,BL_i)	23,IH1
26	idp = ids	25,T32,T30
27	$\neg(\exists\, i)$(s-id$\circ$elem(i)$\circ$s-dp(ST_i) = ids)	26,24
28	d-b(ids,BL_{i+1}) = d-b(ids,BL_i)	T27,T29,27,T15
29	nm' = nm	23,1-9a
30	nm' = f($\alpha\circ$elem(CTR_i)$\circ$s-sp,BL_i)	29,25
31	f(α,BL_{i+1}) = f($\alpha\circ$elem(CTR_i)$\circ$s-sp,BL_i)	T32,28
32	nm' = f(α,BL_{i+1})	30,31

This concludes the case given in 23 and thus property 1 holds
Property 2:

33	is-id(nm')	
34	nm' = nm	33,T15,T20
35	is-id(nm)	33,34

36 ids' = ids T15

37 nm = ids 35,IH2

38 nm' = ids

This concludes the proof that property 2 holds and thus concludes the induction by <u>inst-bl</u>.

Induction from BL_i to BL_{i+1} by <u>inst-proc</u>

let ind = $(\iota j)((\exists k)(\text{s-den-b}\circ\text{s-nm}(ST_i)(\xi) = \text{s-atr}\circ\text{elem}(k)\circ\text{s-dp}\circ\text{s-ptr}(\text{elem}(j,BL_i))))$

dec = $(\iota j)((\exists \alpha)(\text{s-den-b}\circ\text{s-nm}(ST_i)(\xi) = \text{s-atr}\circ\text{elem}(j)\circ\alpha))$

nm = $\alpha\circ\text{s-body}\circ\text{s-den-b}(\text{s-nm}(ST_i)(\xi))(TX_i)$

nm' = $\alpha\circ PTR_{i+1}(TX_{i+1})$

ids = $\alpha\circ\text{s-body}\circ\text{s-atr}\circ\text{elem}(dec)\circ\text{s-dp}\circ PTRs_{ind}(TXs)$

ids' = $\alpha\circ PTRs_{i+1}(TXs)$

lenp = $\text{len-part}(\text{s-nm}(ST_i))$

39 nm = $\alpha\circ\text{s-body}\circ\text{s-den-b}(\text{s-nm}(ST_i)(\xi))(TX_{lenp})$ 1-4

40 lenp = ind 1-2,2-1

41 nm = $\alpha\circ\text{s-body}\circ\text{s-atr}\circ\text{elem}(dec)\circ\text{s-dp}\circ PTR_{ind}(TX_{ind})$ 1-2,T15,40

Property 1:

42 is-un(nm')

Two case distinctions are made:

43 case: $\neg$ is-un(nm)

44 is-id(nm) 42,43,T16,T21

45 $(\exists j)(\text{elem}(j)\circ\text{s-pp}\circ\text{s-den-b}(\text{s-nm}(ST_i)(\xi))(TX_i)) = nm)$ 42,44,T14,T16,T21

let j be that j satisfying 45

arg = $\text{elem}(j)\circ\text{s-ap}(ST_i)$

46 nm' = arg 45,T21

47 is-un(arg) 42,46

48 is-id($\alpha\circ\text{s-body}\circ\text{s-atr}\circ\text{elem}(dec)\circ\text{s-dp}\circ PTR_{ind}(TX_{ind})$) 41,44

49 $\alpha\circ\text{s-body}\circ\text{s-atr}\circ\text{elem}(dec)\circ\text{s-dp}\circ PTR_{ind}(TX_{ind})$ = ids 48,IH2

50 nm = ids 41,49

	let args = $\text{elem}(j)\circ\text{s-ap}(\text{STs})$	
51	arg = $f(\text{elem}(j)\circ\text{s-ap}\circ\text{elem}(\text{CTRs}_i)\circ\text{s-sp}\circ\text{PTRs}_i,\text{BL}_i)$	46,IH1
52	ids' = ids	T16
53	$\text{elem}(j,\text{s-pp}\circ\text{s-den-b}(\text{s-nm}(\text{ST}_i)(\xi))(\text{TX}_i))$ = ids'	45,50,52
54	$\text{d-b}(\text{ids}',\text{BL}_{i+1}) = \text{d-b}(\text{args},\text{BL}_i)$	52,T27,T29
55	$f(\text{elem}(j)\circ\text{s-ap}\circ\text{elem}(\text{CTRs}_i)\circ\text{s-sp},\text{BL}_i) = f(\alpha,\text{BL}_{i+1})$	T32,54
56	$\text{nm}' = f(\alpha,\text{BL}_{i+1})$	51,55

Which concludes the case given in 43.

57	case: is-un(nm)	
58	$\neg(\exists i)(\text{elem}(i)\circ\text{s-pp}\circ\text{s-den-b}(\text{s-nm}(\text{ST}_i)(\xi))(\text{TX}_i))$ = idp)	57,1-8b
59	$\text{is-un}(\alpha\circ\text{s-body}\circ\text{s-atr}\circ\text{elem}(\text{dec})\circ\text{s-dp}\circ\text{PTR}_{ind}(\text{TX}_{ind})$	51,41
60	$\alpha\circ\text{s-body}\circ\text{s-atr}\circ\text{elem}(\text{dec})\circ\text{s-dp}\circ\text{PTR}_{ind}(\text{TX}_{ind})$	59,IH1
61	$\text{nm} = f(\alpha,\text{BL}_{ind})$	41,60
62	idp = ids	61,T32
63	ids = ids'	T16
64	$\neg(\exists i)(\text{elem}(i)\circ\text{s-pp}\circ\text{s-den-b}(\text{s-nm}(\text{ST}_i)(\xi))(\text{TX}_i))$ = ids')	62,63,58
65	$\text{d-b}(\text{ids}',\text{BL}_{i+1}) = \text{d-b}(\text{ids},\text{BL}_{ind})$	63,64,T27,T29,2-1
66	nm' = nm	51,1-9b
67	$\text{nm}' = f(\alpha,\text{BL}_{i+1})$	66,61,65,T32

This concludes the proof of both the case given in 57 and Property 1.

Property 2:

68	id-is(nm')	
69	nm' = nm	T16,T21
70	$\text{is-id}(\alpha\circ\text{s-body}\circ\text{s-atr}\circ\text{elem}(\text{dec})\circ\text{s-dp}\circ\text{PTR}_{ind}(\text{TX}_{ind}))$	68,41
71	$\alpha\circ\text{s-body}\circ\text{s-atr}\circ\text{elem}(\text{dec})\circ\text{s-dp}\circ\text{PTR}_{ind}(\text{TX}_{ind})$ = ids	70,IH2
72	nm = ids	41,71
73	ids' = ids	T16,T21
74	ids' = nm'	73,72,69

This concludes the proof of property 2 and thus the induction according to <u>inst-proc</u>.

Since inst-bl and inst-proc are the only instructions which create new elements of BL, and they preserve the property which we know holds for all initial elements, the property holds for all elements of BL. This concludes the proof of the theorem.

Theorem 2-4 shows that the denotation entry for a name can be located, in the extended machine, by a particular searching method. The retrieval of the denotation is expresssed in the corollary 2-5. However, further information is found during the search process (i.e. the block in which the denotation is located) which can be used to supplement the more meagre information contained in the s-den-s components to fulfil the purposes of s-den-b, this is expressed in corollaries 2-6 to 2-8.

Corollary 2-5: Shows that the same denotation can be found by either applying the unique name to the state or by searching with the identifier.

$\text{is-un}(\alpha \circ \text{PTR}(\text{TX})) \supset$ 2-4,T32
$\alpha \circ \text{PTR}(\text{TX})(\xi) = (\alpha \circ \text{PTRs}(\text{TXs})) \circ \text{s-dn} \circ \text{elem}(\text{d-b}(\alpha \circ \text{PTRs}(\text{TXs}),\text{length}(\text{BL}),\text{BL}),\text{BL})$

Corollary 2-6: Shows that the block referred to by a unique name can be found by the search.

$\text{is-un}(\alpha \circ \text{PTR}(\text{TX})) \supset$ 2-4,T32
$\text{d-b}(\alpha \circ \text{PTRs}(\text{TXs}),\text{length}(\text{BL}),\text{BL}) = (\iota i)((\exists\, \text{id})(\alpha \circ \text{PTR}(\text{TX}) =$
$\text{id} \circ \text{s-dn} \circ \text{elem}(i) \circ \text{s-bl} \wedge \text{is-id}(\text{id})))$

Corollary 2-7: Shows that the original procedure denotation can be constructed from the search method.

$\text{is-m-proc-den}(\text{s-den-b}(\alpha \circ \text{PTR}(\text{TX})(\xi))) \supset$ T15,2-5,2-6
$\text{s-den-b}(\alpha \circ \text{PTR}(\text{TX})(\xi)) =$
$\text{s-atr} \circ \text{elem}(\text{d-d}(\alpha \circ \text{PTRs}(\text{TXs}),\text{length}(\text{BL}),\text{BL})) \circ \text{s-dp} \circ (\text{s-ptr}(\text{elem}(\text{d-b}(\alpha \circ \text{PTRs}(\text{TXs}),$
$\text{length}(\text{BL}),\text{BL}),\text{BL})))$

Corollary 2-8: a) Shows that the block located by a search will be the one whose pointer is stored in the original label denotation.

$\text{is-m-lab-den}(\text{s-den-b}(\alpha \circ \text{PTR}(\text{TX})(\xi))) \supset$ T15,2-6
$\text{s-b} \circ \text{s-den-b}(\alpha \circ \text{PTR}(\text{TX})(\xi)) = \text{s-ptr}(\text{elem}(\text{d-b}(\alpha \circ \text{PTRs}(\text{TXs}),\text{length}(\text{BL}),\text{BL}),\text{BL}))$

b) Shows that the index part of a label denotation can be found using either reference mechanism.

$\text{is-m-lab-den}(\text{s-den-b}(\alpha \circ \text{PTR}(\text{TX})(\xi))) \supset$ T15,2-5
$\text{s-dcl} \circ \text{s-den-b}(\alpha \circ \text{PTR}(\text{TX})) = \text{d-d}(\alpha \circ \text{PTRs}(\text{TXs}),\text{length}(\text{BL}),\text{BL})$

Theorem 2-9: The search model of section 2.1 is equivalent to the twin machine and thus to the base model of section 1.2.

Proof: It is noticed that the following changes to the twin machine do not change its function from text to results:

a) The second argument of inst-proc(T14) is-changed to 2-7

s-atr∘elem(d-d(s-nm(STs),length(BL),BL))∘s-dp∘(s-ptr(elem(d-b(s-nm(STs), length(BL),BL),BL)))

b) The implicit evaluation of the s-epa component (T16) is removed by passing an extra argument from int-st (T14) whose value is d-b(s-nm(STs),length(BL),BL) 2-6

c) The location of the parameter list (T16) is changed from s-pp(s(TX)) to s-pp∘s-atr∘elem(dec)∘s-dp∘s-ptr-s(elem(ind,BL)) 2-7

d) The value stored in s-b(ABN) is changed to be of type is-int and is calculated (T14) by d-b(STs,length(BL),BL) and int-next-st (T13) compares s-b(ABN) with length(BL) 2-8a

e) The value stored in s-dcl(ABN) is calculated (T14) by d-d(STs,length(BL),BL) 2-8b

f) d) & e) are simplified (T14) by setting ABN to d(STs,length(BL),BL) T29

g) All references to denotations are changed to use search and the s-den-s components (including unspecified parts of the interpreter). In particular, the first argument to inst-proc (T14) becomes

$\mathop{\mathrm{LIST}}_{i=1}^{length(s\text{-}ap(STs))}$ d(elem(i,s-ap(STs)),length(BL),BL) 2-5

h) The s-tx and s-ptr block local components and the s-den-b sub-component are no longer used and are deleted.

i) The decomposition of argument s (T16) can be avoided by passing (T14) s-atr∘elem(d-d(s-nm(ST),length(BL),BL)∘s-dp∘(s-ptr(elem(d-b(s-nm(ST), length(BL),BL),BL)))

j) The now superfluous s-den-s selector can be systematically removed.

Steps a) - j) have created the search model and the justifications on the right have shown it to be equivalent to the twin model.

2.4 Further Properties

This section presents some properties, of the model given in section 2.1, which were not given in section 2.2 since their proofs are most easily seen by appealing to the equivalence result proved in section 2.3. In fact we could present analogues of most of the properties 1-3 to 1-9 as corollaries of the equivalence, but only 2-10 and 2-11 will be presented as such. Property 2-12 is also required below.

Property 2-10: This property shows that, under the assumption of a proper program, for any block pointed to by (the PTR component of) an element of BL, a search using the function d applied to one of its identifiers not contained in a nested block will not yield Ω.

$$1 \leq i < \text{length}(BL) \supset$$
$$(\text{is-dir-cont}(\alpha, PTR_i(TX_i)) \wedge \text{is-id}(\alpha \circ PTR_i(TX_i)) \supset d(\alpha \circ PTR_i(TX_i), i, BL) \neq \Omega)$$

A proof follows immediately from 1-6 and 2-4.

Property 2-11: a) This property states that if the activation of a block causes the value of d applied to an identifier to change, then that identifier must have been introduced.

$$d(\alpha \circ PTR_{i+1}(TX_{i+1}), i+1, BL) \neq d(\alpha(ST_i), i, BL) \supset \text{is-decl}(\alpha(ST_i), ST_i) \vee$$
$$d(\alpha \circ PTR_{i+1}(TX_{i+1}), i+1, BL) \neq d(\alpha \circ \text{s-body}(\text{s-nm}(ST_i)(DN_i)), i, BL) \supset \text{is-parmd}(\alpha \circ \text{s-body}(\text{s-nm}(ST_i)(DN_i)), \text{s-nm}(ST_i)(DN_i))$$

A proof follows from 1-8 and 2-4.

b) The following is a generalisation of a):

$$1 < i \leq \text{length}(BL) \supset d(id, i, BL) \neq \Omega \supset (\exists \alpha, \beta)(PTR_i = \alpha \circ \beta \wedge \text{is-intr}(id, \beta(TX)))$$

A proof follows from the observation that for all elements of BL either is-block(ST_i) or is-call(ST_i) and the above result.

Property 2-12: This property shows that the integers stored in components of BL which are indexes of BL will always be less than the index of the point at which they are stored.

$1 < i \leq \text{length}(BL) \supset$

a) $\text{s-epa}(BL_i) < i$

b) $\text{is-parm-den}(\text{id}(DN_i)) \supset \text{s-b}\circ\text{id}\circ DN_i < i$

c) $\text{d-b}(\text{id},i,BL) \leq i$

A proof by induction on BL can be constructed to show that the property is true when the elements are installed and 2-1 assures us that they remain true until deleted.

3. IMPLEMENTATION USING A DISPLAY OF THE STATIC CHAIN

We recall that the introduction to section 2 explained that the purpose of a display is to provide a vector of pointers which supply the link from some index, appended to references in a prepass, to the dynamic area in which the denotation of the referenced name is to be found. The knowledge gained from our search implementation of section 2.1, that the only referencable variables are on the static chain, prompts the idea of having a display pointing to the elements of this chain.

The definition of an E.P.A. pointer (i.e. for some block, it points to the dynamic area whose pointer points to the block or procedure containing it) gives the key as to how the indexes to the display are computed by the prepass: each reference to a variable must be supplemented with the lexicographic depth of the declaration of the variable (i.e. number of embracing blocks). With the elements of the display set to point to the dynamic areas found on the current static chain (the first element of the display being that of the outermost block), its "n"th element will point to the dynamic area containing the denotation of any variable whose active reference has a depth element n. This is the subject of the proof in section 3.3. Thus any variable can be referenced with one indirect step.

As was pointed out in section 2, such a display must be updated dynamically, and the model given in section 3.1 employs an extremely wasteful rule: each time the length of the block local list is changed, the entire static chain is traced and stored in the display. The subject of optimizations to this basic method is pursued in section 4.

This display method is described in /4/ and, with some improvements to the updating mechanism, in /5/.

3.1 The Model

The transition from the model of section 2.1 to the model of this section is accomplished by the addition of, and provision for maintenance and use of, a global display vector (DISP).

The maintenance is performed by replacing the display by a completely new trace of the static chain (formed by an invocation of upd-disp) each time the length of the list of dynamic areas (BL) changes.

The function dd accepts a reference and uses its depth part to select the element of DISP which points to the dynamic area for this reference.

3.1.1 State

```
SD1   is-state = (<s-bl:is-bl-list>,<s-tx:is-p-block>,
                  <s-abn:(<s-b:is-int>,<s-dcl:is-int>) v is-Ω>,
                  <s-c:is-c>,<s-disp:is-int-list>)

SD2   is-bl = (<s-epa:is-int v is-Ω>,
               <s-ptr:is-sel>,
               <s-ctr:is-int v is-Ω>,
               <s-dn:({<id:is-den> || is-id(id)})>)

SD3   is-den = is-proc-den v is-lab-den v is-parm-den v is-other-den

SD4   is-proc-den = is-int

SD5   is-lab-den = is-int

SD6   is-parm-den = (<s-b:is-int>,<s-decl:is-int>)

SD7   is-other-den = ...

SD8   is-c = see /2/
```

is-p-block etc. as is-block except:

a) is-id (except within is-dec or s-pp) changed to is-ref
   is-ref = (<s-id:is-id>,<s-dep:is-int>)

b) is-p-block and is-p-proc-atr have the additional component <s-dep:is-int>

Abbreviations used:

```
BL = s-bl(ξ)
TX = s-tx(ξ)
ABN = s-abn(ξ)
DISP = s-disp(ξ)
B = last(BL)
EPA = s-epa(B)
PTR = s-ptr(B)
CTR = s-ctr(B)
DN = s-dn(B)
DEP = s-dep(PTR(TX))
P = CTR ≠ Ω ——→ elem(CTR)∘s-sp∘PTR
    T ——→ s-body∘PTR
ST = P(TX)
cur(bl) = elem(length(bl))∘s-bl
```

The initial state ξ_o for any text t_o satisfying is-program:

```
SD0   $\xi_o$ = $\mu_o$(<s-bl:[$\mu_o$(<s-ptr:I>,
                          <s-ctr:0>)]>,
            <s-tx:$\mu_o$(<s-sp:[prep($t_o$)]>,<s-dep:0>,
                        <s-dp:[]>)>,
            <s-disp:[]> ,
            <s-c:int-next-st>)
```

Where prep must satisfy 3-1.

3.1.2 State transition function

```
SD9   int-next-st =
         CTR < length(s-sp∘PTR(TX)) ∧ ABN = $\Omega$ $\longrightarrow$ int-next-st;
                                                     int-st;
                                                       step-ctr
         s-b(ABN) = length(BL)  $\longrightarrow$ int-next-st;
                                    int-st;
                                      set-ctr
         T $\longrightarrow$ null

SD10  int-st =
         is-p-block(ST) $\longrightarrow$ rev-disp;
                              epilogue;
                                int-next-st;
                                  rev-disp;
                                    inst-bl
         is-p-call(ST) $\longrightarrow$ rev-disp;
                             epilogue;
                               epilogue;
                                 int-next-st;
                                   rev-disp;
                                     inst-bl;
                                       inst-proc( $\mathop{LIST}\limits_{i=1}^{length(s\text{-}ap(ST))}$ dd(elem(i,s-ap(ST)),BL,DISP),
                                         s-atr∘elem(d)∘s-dp∘(s-ptr(elem(b,BL))),b)
         is-p-goto(ST) $\longrightarrow$ s-abn:dd(ST,BL,DISP)
                          ⋮

      where: d = dd-d(s-nm(ST),BL,DISP)
             b = dd-b(s-nm(ST),BL,DISP)
```

```
SD11  inst-bl =
          s-bl:BL⌒[μ_o(<s-epa:length(BL)>,
                       <s-ptr:P>,
                       <s-ctr:0>,
                       <s-dn:μ_o({<s-id∘elem(i,s-dp(ST)):i> |
                              ¬is-p-other-atr(s-atr(elem(i,s-dp(ST))))}∪...)>)]

SD12  inst-proc(arg-l,s,epa) =
          s-bl:BL⌒[μ_o(<s-epa:epa>,
                       <s-ptr:s>,
                       <s-dn:μ_o({<elem(i,s-pp(s(TX))):elem(i,arg-l)> |
                                          1 ≤ i ≤ length(arg-l)})>)]

SD13  step-ctr = s-ctr∘cur(BL):CTR + 1

SD14  set-ctr = s-ctr∘cur(BL):s-ctr(elem(s-dcl(ABN),s-dp(PTR(TX))))
                s-abn:Ω

SD15  epilogue = cur(BL):Ω

SD16  rev-disp = s-disp:upd-disp(BL,DEP)
```

SD17 upd-disp(bl,dep) = $\overset{dep}{\underset{i=1}{\mathrm{LIST}}}$ find-epa(dep - i,bl)

```
SD18  dd-b(ref,bl,disp) = s-b(dd(ref,bl,disp))

SD19  dd-d(ref,bl,disp) = s-dcl(dd(ref,bl,disp))

SD20  dd(ref,bl,disp) =
          is-parm-den(s-id(ref)∘s-dn∘elem(ind,bl) ——→ s-id(ref)∘s-dn∘elem(ind,bl)
          T ——→ μ_o(<s-b:ind>,
                    <s-dcl:s-id(ref)∘s-dn∘elem(ind,bl)>)

      where ind = elem(s-dep(ref),disp)
```

3.2 Properties

This section states the axiom required for any prep function and gives a property of the machine of section 3.1.

Axiom 3-1: This axiom states that any function proposed for prep must add to each block an index, which is the depth of the block, and add to each reference an index which is the depth of the deepest surrounding block which introduces the identifier (see B19, S22, S24 for is-intr, nth-blk, cnt-blk respectively).

$\text{is-block}(t_o) \wedge pt = \text{prep}(t_o) \supset$

a) $\text{is-p-block}(\alpha(pt)) \vee \text{is-p-proc-atr}(\alpha(pt)) \supset \text{s-dep}\circ\alpha(pt) = \text{count-blk}(\alpha,pt) \wedge$

b) $\text{is-ref}(\alpha(pt)) \supset \text{s-dep}\circ\alpha(pt) = \underset{i}{\text{MAX}}\ (\text{is-intr}(\text{s-id}\circ\alpha(pt),\text{nth-blk}(i,\alpha,pt)(pt)))$

Property 3-2:

This property is identical with 1-2 and 2-1 except that it applies to the machine of section 3.1. Its proof again relies on 1-1.

3.3 Justification

We now prove that the machine of section 3.1 is equivalent to that of section 2.1. The proof is relatively simple. We know that the search mechanism locates the first dynamic area of the static chain which contains a declaration of a referenced variable. Since the display is updated in such a simple way (see SD10) it is obvious, or provable by induction, that it will always contain a trace of the static chain. Therefore, it is only necessary to show that the element of the display indexed by the depth component of a reference corresponds to this first element. We know that the depth component of a reference is the deepest surrounding block of the text which declares this entity. It follows from the relation between blocks of the text and the elements of the static chain that this corresponds to the highest index in the display which points to a dynamic area containing an entity of this name. The equivalence of "furthest from the end" and "nearest to the beginning" completes the proof.

For our formal proof we shall again use the twin machine idea (as explained in section 2.3) but, in view of the detailed presentation in that section and the extreme simplicity of the current changes, the definition of the twin machine is not written out. Apart from the extensions, the only change required to the machine of section 2.1 permits it to operate on text which has been preprocessed by the function prep. Such a text has all the original information, only an extra "s-id" selector must be placed before any use of "s-id". The machine is extended by the addition and maintenance of a display (DISP). The functions d and d-b are limited to the domain over which they are actually used (2-10) by deleting the first case distinction of each.

Since in our twin machine, the d-b function of S16 is used to supply the entry in the denotation for a parameter, we have only to show that the selected element of the display points to the element of the block local list in which d-b locates an entry, whether this entry is a parameter or variable denotation. In order to specify this block we introduce an auxiliary definition which locates this block local element:

SD21 $\text{df-intr}(id,n,bl) = id \circ \text{s-dn} \circ \text{elem}(n,bl) \neq \Omega \longrightarrow n$
$\qquad T \longrightarrow \text{df-intr}(id,\text{s-epa}(\text{elem}(n,bl)),bl)$

<u>Theorem 3-3</u>: This states that for any variable which is referencable, the component of DISP indexed by the depth of the reference is the index of the element of BL which would be located by applying df-intr to the identifier of the reference.

ξ is any state immediately after <u>rev-disp</u>
$\text{is-ref}(\alpha(TX)) \wedge \text{is-refbl}(\alpha,PTR,TX) \supset$
$\quad \text{elem}(\text{s-dep} \circ \alpha(TX),DISP) = \text{df-intr}(\text{s-id} \circ \alpha(TX),\text{length}(BL),BL)$

Proof:

1 $\text{is-ref}(\alpha(TX))$

2 $\text{is-refbl}(\alpha,PTR,TX)$

3 $\text{count-blk}(\alpha,TX) = \text{count-blk}(PTR,TX)$ — 2,S20

4 $\text{count-blk}(PTR,TX) = \text{s-dep} \circ PTR(TX)$ — 3-1a,3-2

5 $\text{count-blk}(PTR,TX) = DEP$ — 4

6 $\text{count-blk}(\alpha,TX) = DEP$ — 3,5

7 $PTR = \text{nth-blk}(DEP,\alpha,TX)$ — 2,6,S20,S21

8 $PTR_{\text{find-epa}(k,BL)} = \text{nth-blk}(DEP-k,\alpha,TX)$ for $k \leq DEP$ — 7,2-2b

9 $PTR_{\text{find-epa}(DEP-i,BL)} = \text{nth-blk}(i,\alpha,TX)$ for $i \geq 0$ — 8

10 $\text{is-intr}(\text{s-id} \circ \alpha(TX),\text{nth-blk}(i,\alpha,TX)(TX)) \equiv$ — 9,S19,SD11,SD12
$\qquad \text{is-intr-s}(\text{s-id} \circ \alpha(TX),\text{front}(\text{find-epa}(DEP-i,BL),BL))$

11 $\underset{i}{\text{MAX}}\ \text{is-intr}(\text{s-id} \circ \alpha(TX),\text{nth-blk}(i,\alpha,TX)(TX)) =$ — 10
$\qquad \underset{i}{\text{MAX}}\ \text{is-intr-s}(\text{s-id} \circ \alpha(TX),\text{front}(\text{find-epa}(DEP-i,BL),BL))$

12 $\text{elem}(\text{s-dep} \circ \alpha(TX),DISP) = \text{find-epa}(DEP-\text{s-dp} \circ \alpha(TX),BL)$ — SD16,SD17

13 $\text{elem}(\text{s-dep} \circ \alpha(TX),DISP) = \text{find-epa}(DEP-i,BL)$ — 12,3-1b
$\qquad$ where $i = \underset{i}{\text{MAX}}\ \text{is-intr}(\text{s-id} \circ \alpha(TX),\text{nth-blk}(i,\alpha,TX)(TX))$

14 $\text{elem}(\text{s-dep} \circ \alpha(TX),DISP) = \text{find-epa}(DEP-i,BL)$ — 13,11
$\qquad$ where $i = \underset{i}{\text{MAX}}\ \text{is-intr-s}(\text{s-id} \circ \alpha(TX),\text{front}(\text{find-epa}(DEP-i,BL),BL))$

15 $\text{elem}(\text{s-dep} \circ \alpha(TX),DISP) = \text{find-epa}(j,BL)$ — 14
$\qquad$ where $j = \underset{j}{\text{MIN}}\ \text{is-intr-s}(\text{s-id} \circ \alpha(TX),\text{front}(\text{find-epa}(j,BL),BL))$

16 $\text{df-intr}(\text{s-id} \circ \alpha(TX),\text{length}(BL),BL) = \text{find-epa}(i,BL)$ — SD21,S19,S26
$\qquad$ where $i = \underset{i}{\text{MIN}}\ \text{is-intr-s}(\text{s-id} \circ \alpha(TX),\text{front}(\text{find-epa}(i,BL),BL))$

17 $\text{elem}(\text{s-dep}\circ\alpha(TX),DISP) = \text{df-intr}(\text{s-id}\circ\alpha(TX),\text{length}(BL),BL)$ 15,16

This concludes the proof that the two functions are equivalent over the stated domain.

<u>Theorem 3-4</u>: The machine of section 3.1 is equivalent to that of section 2.1.

Proof: First it is observed that the only places in our (unwritten) twin machine where references are made, have the same values of the critical state vectors as after a <u>rev-disp</u> instruction (3-2).

Now:

1 $\text{d-b}(\text{s-id}\circ\alpha(TX),\text{length}(BL),BL) =$ S16,S18,SD21

$$\begin{array}{l} \text{is-parm-den}((\text{s-id}\circ\alpha(TX))(DN_{intr})) \longrightarrow \text{s-b}((\text{s-id}\circ\alpha(TX))(DN_{intr})) \\ T \longrightarrow intr \end{array}$$

where $intr = \text{df-intr}(\text{s-id}\circ\alpha(TX),\text{length}(BL),BL)$

2 $\text{dd-b}(\alpha(TX),BL,DISP) =$ SD18,SD20

$$\begin{array}{l} \text{is-parm-den}((\text{s-id}\circ\alpha(TX))(DN_{intr})) \longrightarrow \text{s-b}((\text{s-id}\circ\alpha(TX))(DN_{intr})) \\ T \longrightarrow intr \end{array}$$

where $intr = \text{elem}(\text{s-dep}\circ\alpha(TX),DISP)$

3 $\text{dd-b}(\alpha(TX),BL,DISP) = \text{d-b}(\text{s-id}\circ\alpha(TX),\text{length}(BL),BL)$ 1,2,3-3

Since the change of reference mechanisms, justified by 3, is the only change we require to create the machine of 3.1, that machine must be equivalent to the twin machine and thus equivalent to the machine of section 2.1.

4. OPTIMIZATION OF THE STATIC CHAIN DISPLAY

Continuing our discussion of the display from the introduction to sections 2 and 3, we already have a display whose one indirect step for references is as good as can be hoped for in view of the dynamic nature of procedure calls. The direction of our optimization effort must therefore be to simplify the updating of the display whilst preserving this desirable property.

Section 4.1 shows that all entries on the static chain for simple blocks can be considered to be superfluous and section 4.2 shows how the updating of the already shortened static chain can be simplified for some cases. Section 4.3 proposes an alternative which, even more so than other ideas, is not a clear cut optimization since its relative efficiency depends on the instructions available for tracing lists or storing the display.

4.1 Omission of Simple Blocks from the Static Chain

The model given in section 4.1.1 has dynamic storage areas (procedure local elements - PL) only for procedure activations. Each such element contains the storage for all simple blocks nested within the procedure body (section 1 explained that aggregates would be handled by treating a dope vector as a simple entity, so such a scheme is not wasting much store). This change utilizes the knowledge that within a procedure activation simple blocks cannot be reactivated (see 2-3). The further knowledge, that parallel simple blocks (i.e. those of equal, relative, depth) cannot be simultaneously active, is employed by the prepass when it allocates the same series of addresses (relative to the procedure) to variables in parallel simple blocks (see 4-1).

The static chain joins these procedure activations and the correspondingly shortened display is recomputed only when the length of PL changes.

The optimization described here became known to the authors from /10/.

4.1.1 Model

4.1.1.1 State

```
SD1.1 is-state = (<s-pl:is-pl-list>,<s-tx:is-p-proc-atr>,
                  <s-abn:(<s-p:int>,<s-dcl:is-lab-den>) v is-Ω>,
                  <s-c:is-c>,<s-disp:is-int-list>)
```

SD1.2 is-pl = (<s-epa:is-int v is-Ω>,
<s-ptr:is-sel>,
<s-ctr:is-int v is-Ω>,
<s-btr:is-sel>,
<s-dn:is-den-list>)

SD1.3 is-den = is-proc-den v is-lab-den v is-parm-den v is-other-den

SD1.4 is-proc-den = is-sel

SD1.5 is-lab-den = (<s-btr:is-sel>,<s-dec:is-int>)

SD1.6 is-parm-den = (<s-p:is-int>,<s-dcl:is-proc-den v is-lab-den v is-other-den>)

SD1.7 is-other-den = ...

SD1.8 is-c = see /2/

is-p-block etc. as is-block except:
a) is-id changed to is-ref
is-ref = (<s-off:is-int>,<s-dep:is-int>)
b) is-p-proc-atr has the additional component
<s-dep:is-int>

Abbreviations used:
PL = s-pl(ξ)
TX = s-tx(ξ)
ABN = s-abn(ξ)
DISP = s-disp(ξ)
PR = last(PL)
EPA = s-epa(PR)
PTR = s-ptr(PR)
CTR = s-ctr(PR)
BTR = s-btr(PR)
DN = s-dn(PR)
DEP = s-dep(PTR(TX))
Q = CTR $\neq \Omega \rightarrow$ elem(CTR)$\circ$s-sp$\circ$BTR
T $\rightarrow$ s-body$\circ$BTR
P = Q$\circ$PTR
ST = P(TX)
cur(pl) = elem(length(pl))$\circ$s-pl

Initial State ξ_0 for any text t_0 satisfying is-block(t_0):

```
SD1.0 $\xi_0 = \mu_0$(<s-pl:[$\mu_0$(<s-ptr:s-body>,
                         <s-btr:I>,
                         <s-ctr:0>,
                         <s-dn:[]>)]>,
           <s-tx:$\mu_0$(<s-body:[prep($t_0$)]>,<s-dep:0>,
                        <s-pp:[]>)>,
           <s-disp:[]>,
           <s-c:int-next-st>)
```

where prep must satisfy 4-1.

4.1.1.2 State transition function

```
SD1.9 int-next-st =
        s-p(ABN) = length(PL) $\rightarrow$ int-next-st;
                                       int-st;
                                         set-btr
        CTR < length(s-sp∘BTR∘PTR(TX)) ∧ ABN = $\Omega$ $\rightarrow$ int-next-st;
                                                         int-st;
                                                           step-ctr
        CTR ≠ $\Omega$ ∧ ABN = $\Omega$ $\rightarrow$ int-next-st;
                                      reset-btr
        T $\rightarrow$ null

SD1.10 int-st =
        is-p-block(ST) $\rightarrow$ inst-bl
        is-p-call(ST) $\rightarrow$ rev-disp;
                               epilogue;
                                 int-next-st;
                                   inst-bl;
                                     rev-disp;
                                       inst-proc( $\mathrm{LIST}_{i=1}^{length(s\text{-}ap(ST))}$ dd(elem(i,s-ap(ST)),PL,DISP),
                                                      s-atr∘d∘s-ptr(elem(p,PL)),p))
        is-p-goto(ST) $\rightarrow$ s-abn:dd(ST,PL,DISP)
                          ⋮
        where:  d = dd-d(s-nm(ST),PL,DISP)
                p = dd-p(s-nm(ST),PL,DISP)
```

```
SD1.11 inst-bl =

          s-ctr∘cur(PL):0
          s-btr∘cur(PL):Q
          s-dn∘cur(PL):μ(DN;{<elem(s-off∘s-id∘elem(i,s-dp(ST))):elem(i)∘s-dp∘Q> |
                                       is-p-proc-atr(s-atr(elem(i,s-dp(ST))))} ∪
                             {<elem(s-off∘s-id∘elem(i,s-dp(ST))):μ₀(<s-btr:Q>,<s-dec:i>)> |
                                       is-p-lab-atr(s-atr(elem(i,s-dp(ST))))} ∪ ...)

SD1.12 inst-proc(arg-l,s,epa) =
          s-pl:PL⁀[μ₀(<s-epa:epa>,
                     <s-ptr:s>,
                     <s-btr:I>,
                     <s-dn:μ₀({<elem(i):elem(i,arg-l)> | 1 ≤ i ≤ length(arg-l)})>)]

SD1.13 reset-btr = s-btr∘cur(PL):(ια)((∃ i)(BTR = elem(i)∘s-sp∘α))
                   s-ctr∘cur(PL):(ι i)((∃ α)(BTR = elem(i)∘s-sp∘α))

SD1.14 step-ctr = s-ctr∘cur(PL):CTR + 1

SD1.15 set-btr = s-btr∘cur(PL):s-btr∘s-dcl(ABN)
                 s-ctr∘cur(PL):s-atr(elem(s-dec∘s-dcl(ABN),s-dp∘(s-btr∘s-dcl(ABN))∘PTR(TX)))
                 s-abn:Ω

SD1.16 epilogue = cur(PL):Ω

SD1.17 rev-disp = s-disp:upd-disp(PL,DEP)

                            dep
SD1.18 upd-disp(pl,dep) =  LIST find-epa(dep-i,pl)
                            i=1

SD1.19 dd-p(ref,pl,disp) = s-p(dd(ref,pl,disp))

SD1.20 dd-d(ref,pl,disp) = s-dcl(dd(ref,pl,disp))

SD1.21 dd(ref,pl,disp) = is-parm-den(elem(s-off(ref),s-dn(elem(ind,pl)))) ——→
                                          elem(s-off(ref),s-dn(elem(ind,pl)))
                         T ——→ μ₀(<s-p:ind>,<s-dcl:elem(s-off(ref),s-dn(elem(ind,pl)))>)

       where ind = elem(s-dep(ref),disp)
```

4.1.2 Axiom of the prepass

Axiom 4-1: This axiom states the required properties of the function prep:

a) An index is added to each procedure attribute indicating the number of embracing procedures.

b) Each name introduced is also supplemented by a depth index which is that of its deepest containing procedure.

An offset is added to each name introduced which is

c) Without gaps within a block

d) Unique within the block

e) Larger than any of a surrounding block within the same procedure

f) Each use of a name is replaced by the reference to which that name refers.

$$\text{is-block}(t_0) \wedge pt = \text{prep}(t_0) \supset$$

a) $$\text{is-proc-atr}(\alpha(t_0)) \supset \text{s-dep}\circ\alpha(pt) = \text{count-proc}(\alpha, t_0)$$

b) $$\text{is-dec}(\text{elem}(i)\circ\text{s-dp}\circ\alpha(t_0)) \supset \text{s-dep}\circ\text{elem}(i)\circ\text{s-dp}\circ\alpha(pt) = \text{count-proc}(\alpha, t_0)$$
$$\text{is-id}(\text{elem}(i)\circ\text{s-pp}\circ\alpha(t_0)) \supset \text{s-dep}\circ\text{elem}(i)\circ\text{s-pp}\circ\alpha(pt) = \text{count-proc}(\alpha, t_0)$$

c) $$\text{is-block}(\alpha(t_0)) \supset \quad \underset{i}{\text{MAX}}\,(i = \text{s-off}\circ\text{elem}(j)\circ\text{s-dp}\circ\alpha(t_0)) -$$
$$\underset{i}{\text{MIN}}\,(i = \text{s-off}\circ\text{elem}(k)\circ\text{s-dp}\circ\alpha(t_0)) + 1 = \text{length}(\text{s-dp}\circ\alpha(t_0))$$
$$\text{where } j,k \le \text{length}(\text{s-dp}\circ\alpha(pt))$$
$$\text{is-proc-atr}(\alpha(t_0)) \supset \underset{i}{\text{MAX}}\,(i = \text{s-off}\circ\text{elem}(j)\circ\text{s-pp}\circ\alpha(pt)) = \text{length}(\text{s-pp}\circ\alpha(t_0))$$
$$\text{where } j \le \text{length}(\text{s-pp}\circ\alpha(t_0))$$

d) $$\text{is-block}(\alpha(t_0)) \supset (\text{s-off}\circ\text{elem}(i)\circ\text{s-dp}\circ\alpha(pt) = \text{s-off}\circ\text{elem}(j)\circ\text{s-dp}\circ\alpha(pt) \supset i = j)$$
$$\text{is-proc-atr}(\alpha(t_0)) \supset (\text{s-off}\circ\text{elem}(i)\circ\text{s-pp}\circ\alpha(pt) = \text{s-off}\circ\text{elem}(j)\circ\text{s-pp}\circ\alpha(pt) \supset i = j)$$

e) $$\text{is-block}(\alpha(t_0)) \wedge (\text{is-block} \vee \text{is-proc-atr})(\beta(t_0)) \wedge (\exists\gamma)(\alpha = \gamma\circ\beta \wedge \gamma \neq I) \wedge$$
$$\text{count-proc}(\alpha, pt) = \text{count-proc}(\beta, pt) \supset$$
$$(\forall ij)(\text{s-off}\circ\text{elem}(i)\circ\text{s-dp}\circ\alpha(pt) > \text{s-off}\circ\text{elem}(i)\circ\text{s-dp}\circ\beta(pt))$$

f) $$\text{is-ref}(\alpha(pt)) \wedge i = \underset{i}{\text{MAX}}\,(\text{is-intr}(\alpha(t_0), \text{nth-blk}(i,\alpha,t_0)(t_0))) \supset$$
$$((\text{is-block}(\text{nth-blk}(i,\alpha,t_0)(t_0)) \supset$$
$$\alpha(pt) = \text{elem}((\iota j)(\alpha(t_0) = \text{elem}(j)\circ\text{s-dp}\circ\text{nth-blk}(i,\alpha,t_0)(t_0)),$$
$$\text{s-dp}\circ\text{nth-blk}(i,\alpha,t_0)(pt)))$$
$$\wedge\ (\text{is-proc-atr}(\text{nth-blk}(i,\alpha,t_0)(t_0)) \supset$$
$$\alpha(pt) = \text{elem}((\iota j)(\alpha(t_0) = \text{elem}(j)\circ\text{s-pp}\circ\text{nth-blk}(i,\alpha,t_0)(t_0)), \text{s-pp}\circ\text{nth-blk}$$
$$(i,\alpha,t_0)(pt)))$$

4.1.3 Justification

Theorem 4-2: The model given in section 4.1.1 is equivalent to that of section 3.1.

Proof: A series of changes to the model of section 3.1 which yield the machine of section 4.1.1. Justifications are given that they preserve equivalence.

a) The use of an object for the s-dn sub-components can be replaced by a list, and all identifiers changed to indexes of this list within the introducing block. The function dd is modified to use indices. 4-1c,4-1d

b)1) The denotations can be stored relative to a procedure. The information stored in such denotations is extended accordingly. 2-3,4-1c

c) Immediate access to a reference which is separated from its declaration by only simple blocks requires only the address of the procedure containing both. Furthermore, knowledge of the position of the procedure declaration, containing the block in which an arbitrary reference was declared, is enough to locate the denotation. Thus, the E.P.A. entries need only be made for each procedure call, and the display only contain a chain of these entries.

d) The need for the PTR component to be block local is obviated by the fact that it contains a history of the index of the statement at which a temporarily suspended block is to be resumed. (In fact in the model of 4.1.1 a split, into the pointer locating the procedure and that locating the block therein, is made.)

e) Steps a)-d) have eliminated the BL entries for simple blocks and the updating on simple block entry/exit consists solely of denotation and address adjustments.

f) int-next-st is modified to reduce the work required to interpret a goto within a procedure call.

4.2 Simplification of Maintenance of the Display

The model of section 4.1 has reduced the length of the static chain by relating all entities to their procedures, however, the updating of the chain is still performed wastefully in many cases. This section comments on how to simplify the updating in all cases except that of a procedure invoked via a parameter. No model nor any formal proof is offered since the transition is small and the argument simple. However Appendix I contains a non-recursive model which embodies the idea given here.

1) Certain errors can now go undetected which the application of a name to DN would have shown. In particular, the prepass must ensure that no references to variables outside their scope are permitted.

Property 1-5 shows that if any block is active, its statically embracing block is on the dynamic chain. We now observe that for any procedure to be invoked in its own right (as opposed to via a parameter) its statically containing procedure must still be on the static chain. This is the case since for it to have been invoked, the search through the static chain must have found the block in which it was declared. Now, this statically preceding block must be made the element in the static chain to proceed the newly invoked procedure. This is achieved by simply setting the E.P.A. pointer of the new dynamic area to equal the entry in the display at depth one less than the depth of this procedure. In parallel, the display is updated by putting the address of the new dynamic area in the display at the point indicated by the depth of the invoked procedure.

Notice that the above argument does not apply to procedures invoked via parameters because, although the dynamic area of their statically embracing block is still in the dynamic chain, we are not assured that it is in the current static chain since only the parameter denotation was located in our search - not the declaring block. In the introduction to the language we attempted to explain the complex character of passing procedures as parameters by what amounted to a block name style of display as described in section 6. Although it would have been more difficult at that time to explain, it can now be seen that invocation of procedures via parameters also poses significant problems for implementations based on displays of the static chain.

We are now left in the position that we are only forced to update the display by retracing the static chain in the case of procedure exit and the special case of entry via a parameter.

4.3 Updating the Display without the Static Chain

This section shows that the housekeeping for the display can be performed using only the display itself. The desirability of doing this depends on a trade-off between the searches of the static chain and the storage required in all dynamic areas for the suggested storing of display elements, as well as the instructions available for performing both methods. Again neither a model nor a formal proof is presented.

The above section has shown how the display can be installed directly except in the case of invocation via a parameter. The storage, with a procedure parameter, of the display at the time of declaration would facilitate a similar direct installation in the final case. The justification is similar to that given above. To avoid retracing the static chain on exit it is necessary that the contents of the display, from the overwritten element to the end of the active display, are stored in the dynamic area for the procedure call. These stored elements can then be used to reconstitute the display on exit from the procedure. Notice that the necessity to step back through each procedure closed by a goto statement would lose the directness possible in our preceding implementations of goto.

5. IMPLEMENTATION USING F-SEARCH

As explained in the introduction to section 2, we are interested in ways of locating the dynamic area where a referenced variable is to be found.

This section gives a search algorithm which, where section 2.1 uses the static chain, uses a linking we shall refer to as the F-chain. The technique prerequires (5-1) that no two variables declared in statically embracing blocks have the same name [1]. (We shall fulfill this in a convenient way for the subsequent discussion by adding to each identifier declared, and all its corresponding references, the depth of the declaring block.)

The constructed chain is coincident with the dynamic chain (i.e. that which points to the invoking block) in all cases except where a procedure has been invoked via a parameter, in which case the F-chain pointer of the dynamic area of the invoked procedure points to the dynamic area of the first block which passed this procedure as an argument.

The ability to create the F-chain entries necessitates the storage of a pointer, to the block which first passed a procedure parameter, with that parameter.

This technique is essentially that used by the PL/I F-level compiler (see /9/) in updating the display described in the next section. The derivation of this algorithm is contained in /8/.

5.1 The Model

This section indicates how some facets of the above chaining idea appear in the model given below. We do this by relating our new model to that of section 2.1, from which the only differences are the prepass and the setting up of the pointers. The role of the prepass is expressed in axiom 5-1.

The F-chain is indicated by the s-f component where this is undefined(Ω) the preceding element in the dynamic chain is also the next in the F-chain. Procedure parameter denotations are supplemented with a component which points to the dynamic area of the first block which passed this procedure as an argument. This indicator is inserted by the m-p (make parameter) function. The s-f component is updated as follows: for blocks it is left as Ω; for procedures it is set to the s-f component of the denotation, thus automatically inserting Ω for normal invocations and the pointer to the first passing block for parameters. The function "f" operates on the F-chain exactly as did d on the static chain.

[1] This restriction applies only to the search model and will be lifted in the display model of section 6.

5.1.1 State

F1 is-state = (<s-bl:is-bl-list>,<s-tx:is-block>,
<s-abn:(<s-b:is-int>,
s-dcl:is-int>) v is-Ω>,<s-c:is-c>)

F2 is-bl = (<s-f:is-int v is-Ω>,
<s-ptr:is-sel>,
<s-ctr:is-int v is-Ω>,
<s-dn:({<id:is-den> || is-id(id)})>)

F3 is-den = is-proc-den v is-lab-den v is-parm-den v is-other-den

F4 is-proc-den = is-int

F5 is-lab-den = is-int

F6 is-other-den = ...

F7 is-parm-den = (<s-b:is-int>,<s-dcl:is-int>,<s-f:is-int v is-Ω>)

F8 is-c = see /2/

Abbreviations used:
BL = s-bl(ξ)
TX = s-tx(ξ)
ABN = s-abn(ξ)
B = last(BL)
PTR = s-ptr(B)
CTR = s-ctr(B)
F = s-f(B)
DN = s-dn(B)
P = CTR $\neq \Omega \longrightarrow$ elem(CTR)$\circ$s-sp$\circ$PTR
T $\longrightarrow$ s-body$\circ$PTR
ST = P(TX)
cur(bl) = elem(length(bl))$\circ$s-bl

Initial state:
F0 $\xi_o = \mu_o$(<s-bl:[μ_o(<s-ptr:I>,<s-ctr:0>)]>,
<s-tx:μ_o(<s-sp:[t_o]>,<s-dp:[]>)>,
<s-c:$\underline{\text{int-next-st}}$>)

where t_o satisfies is-block and axiom 5-1

5.1.2 State transition function

```
F9   int-next-st =
        CTR < length(s-sp∘PTR(TX)) ∧ ABN = $\Omega$ ──▶ int-next-st;
                                                  int-st;
                                                    step-ctr

        s-b(ABN) = length(BL) ──▶ int-next-st;
                                    int-st;
                                      set-ctr

        T ──▶ null

F10  int-st =
        is-block(ST) ──▶ epilogue;
                           int-next-st;
                             inst-bl

        is-call(ST) ──▶ epilogue;
                          epilogue;
                            int-next-st;
                              inst-bl;
                                inst-proc( $\underset{i=1}{\overset{length(s-ap(ST))}{LIST}}$ m-p(f(elem(i,s-ap(ST)),length(BL),BL),
                                                                                     BL,TX),
                                      s-atr∘elem(d)∘s-dp∘(s-ptr∘elem(b,BL)),f)

        is-goto(ST) ──▶ s-abn:f(ST,length(BL),BL)
                  ⋮

     where: b = f-b(s-nm(ST),length(BL),BL)
            d = f-d(s-nm(ST),length(BL),BL)
            f = f-f(s-nm(ST),length(BL),BL)

F11  inst-bl =
        s-bl:BL⁀[$\mu_0$(<s-ptr:P>,
                       <s-ctr:0>,
                       <s-dn:$\mu_0$({<s-id∘elem(i,s-dp(ST)):i > |
                                 ¬ is-other-atr(s-atr∘elem(i,s-dp(ST)))} ∪ ...)>)]

F12  inst-proc(arg-1,s,f) =
        s-bl:BL⁀[$\mu_0$(<s-f:f>,
                       <s-ptr:s>,
                       <s-dn:$\mu_0$({<elem(i,s-pp∘s(TX)):elem(i,arg-1)> |
                                                      1 ≤ i ≤ length(arg-1)})>)]
```

F13 $\underline{\text{step-ctr}}$ = s-ctr∘cur(BL):CTR + 1

F14 $\underline{\text{set-ctr}}$ = s-ctr∘cur(BL):s-atr∘elem(s-dcl(ABN))∘s-dp∘PTR(TX)
s-abn: Ω

F15 $\underline{\text{epilogue}}$ = cur(BL): Ω

```
F16   f(id,n,bl) = n ≤ 1 ⟶ Ω
                   is-parm-den∘id∘s-dn∘elem(n,bl) ⟶ id∘s-dn∘elem(n,bl)
                   id∘s-dn∘elem(n,bl) ≠ Ω ⟶ μ₀(<s-b:n>,<s-dcl:id∘s-dn∘elem(n,bl)>)
                   s-f∘elem(n,bl) ≠ Ω ⟶ f(id,s-f∘elem(n,bl),bl)
                   T ⟶ f(id,n-1,bl)

F17   f-b(id,n,bl) = f (id,n,bl) = Ω ⟶ 0
                     T ⟶ s-b∘f(id,n,bl)

F18   f-d(id,n,bl) =  s-dcl∘f(id,n,bl)

F19   f-f(id,n,bl) = s-f∘f(id,n,bl)

F20   m-p(den,bl,tx) = s-f(den) ≠ Ω  ∨ ¬ is-proc-atr∘s-atr∘elem(s-dcl(den))∘s-dp∘
                                       s-ptr∘elem(s-b(den),bl)(tx) ⟶ den
                       T ⟶ μ(den;<s-f:length(bl)>)
```

5.1.3 Axiom of the text

Axiom 5-1: This axiom imposes the restriction that there do not exist two declarations of the same identifier in two blocks where one embraces the other.

$$\alpha \neq I \wedge \text{is-intr}(id,\beta(t)) \supset \neg\,\text{is-intr}(id,\alpha\circ\beta(t))$$

5.2 Justification

This section establishes the equivalence of the model of section 5.1 to that of 2.1.

The correctness of the adherence to the dynamic chain, which is only difficult to understand in the case of a procedure invocation, is established by noting that a procedure invoked in its own right (as opposed to a parameter invocation) is invoked, either from its surrounding block, or from a block whose dynamic area has as the first dynamic area preceding it in the dynamic chain with lower level that surrounding block.

In either case a name not declared in the invoked procedure, by virtue of its lower level, can not possibly be found in the chain earlier than that surrounding block since no names of that level are introduced. The argument to justify the pointing of the F-chain to the block first passing the procedure rather than that declaring the procedure also relies on the level numbers preventing a premature halt to the search. We now proceed to the formal proof (where 5-1 is used as the basis of the argument instead of our informal depths) which is again based on the twin machine method.

5.2.1 Twin model

The machine given below is based on that of section 2.1. New components are added to the state which, since they are not used by the referencing functions, do not influence the result. In this category are s-f of the elements of BL and s-f and s-id of each parameter denotation. The function m-p is used to insert the new entries.

5.2.1.1 State

```
T1     is-state = (<s-bl:is-bl-list>,<s-tx:is-block>,
                   <s-abn:(<s-b:is-int>,
                           <s-dcl:is-int>) v is-Ω>,<s-c:is-c>)

T2     is-bl = (<s-epa:is-int v is-Ω>,
                <s-f:is-int v is-Ω>,
                <s-ctr:is-int v is-Ω>,
                <s-ptr:is-sel>,
                <s-dn:({<id:is-den> || is-id(id)})>)

T3     is-den = is-proc-den v is-lab-den v is-parm-den v is-other-den

T4     is-proc-den = is-int

T5     is-lab-den = is-int

T6     is-other-den = ...
          N.B. must not be is-Ω

T7     is-parm-den = (<s-b:is-int>,<s-dcl:is-int>,<s-f:is-int>,<s-id:is-id>)

T8     is-c = see /2/
```

Abbreviations used:

```
BL = s-bl(ξ)
TX = s-tx(ξ)
ABN = s-abn(ξ)
B = last(BL)
EPA = s-epa(B)
PTR = s-ptr(B)
CTR = s-ctr(B)
F = s-f(B)
DN = s-dn(B)
P = CTR ≠ Ω ——→ elem(CTR)∘s-sp∘PTR
    T ——→ s-body∘PTR
ST = P(TX)
```

Initial state:

```
T0    ξ_0 = (<s-bl:[μ_0(<s-ptr:I>,<s-ctr:0>)>,
            <s-tx:μ_0(<s-sp:[t_0]>,<s-dp:[]>)>,
            <s-c:int-next-st>)
```

where t_o satisfies is-block and axiom 5-1.

5.2.1.2 State transition function

```
T9    int-next-st =
        CTR < length(s-sp∘PTR(TX)) ∧ ABN = Ω ——→ int-next-st;
                                                  int-st;
                                                   step-ctr

        s-b(ABN) = length(BL) ——→ int-next-st;
                                   int-st;
                                    set-ctr

        T ——→ null

T10   int-st =
        is-block(ST) ——→ epilogue;
                          int-next-st;
                           inst-bl
```

cont'd

```
    is-call(ST) ——► epilogue;
                       epilogue;
                         int-next-st;
                           inst-bl;
                             inst-proc( $\underset{i=1}{\overset{length(s-ap(ST))}{LIST}}$ parm$_i$,s,b,f)
    is-goto(ST) ——► s-abn:d(ST,length(BL),BL)
                   .
                   .
                   .

 where: parm$_i$ = m-p(d(elem(i,s-ap(ST)),length(BL),BL),BL,TX,elem(i,s-ap(ST)))
        s = s-atr∘elem(d)∘s-dp∘(s-ptr∘elem(b,BL))
        b = d-b(s-nm(ST),length(BL),BL)
        d = d-d(s-nm(ST),length(BL),BL)
        f = d-f(s-nm(ST),length(BL),BL)

T11  inst-bl =
      s-bl:BL⌒[$\mu_0$(<s-epa:length(BL)>,
                 <s-ptr:P>,
                 <s-ctr:0>,
                 <s-dn:$\mu_0$({<s-id∘elem(i)∘s-dp(ST):i> |
                              ¬ is-other-atr(s-atr∘elem(i,s-dp(ST)))}∪...)>)]

T12  inst-proc(arg-l,s,epa,f) =
        s-bl:BL⌒[$\mu_0$(<s-epa:epa>,
                   <s-f:f>,
                   <s-ptr:s>,
                   <s-dn:$\mu_0$({<elem(i)∘s-pp∘s(TX):elem(i,arg-l)> |
                                               1 ≤ i ≤ length(arg-l)})>)]

T13  step-ctr = s-ctr∘cur(BL):CTR + 1

T14  set-ctr = s-ctr∘cur(BL):s-atr∘elem(s-dcl(ABN))∘s-dp∘PTR(TX)
               s-abn:Ω
T15  epilogue = cur(BL):Ω

T16  d(id,n,bl) = n ≤ 1 ——► Ω
                  is-parm-den∘id∘s-dn∘elem(n,bl) ——► id∘s-dn∘elem(n,bl)
                  id∘s-dn∘elem(n,bl) ≠ Ω ——► $\mu_0$(<s-b:n>,<s-dcl:id∘s-dn∘elem(n,bl)>)
                  T ——► d(id,s-epa∘elem(n,bl),bl)

T17  d-b(id,n,bl) = d(id,n,bl) = Ω ——► 0
                    T ——► s-b(d(id,n,bl))
```

T18 $d\text{-}d(id,n,bl) = s\text{-}dcl \circ d(id,n,bl)$

T19 $d\text{-}f(id,n,bl) = s\text{-}f \circ d(id,n,bl)$

T20 $d\text{-}id(id,n,bl) = s\text{-}id \circ d(id,n,bl)$

T21 $m\text{-}p(den,bl,tx,id) = s\text{-}f(den) \neq \Omega \longrightarrow den$
$\quad T \longrightarrow \mu(den;\langle s\text{-}f:length(bl)\rangle,\langle s\text{-}id:id\rangle)$

5.2.2 Properties of the twin machine

In the following it is assumed that ξ is any state during a computation by TWIN.

Property 5-2: This property is identical with 1-2 but applies to the machines of section 5.1 and 5.2.1.

Property 5-3: This property is supplementary to property 5-2. It states that between the installation and use of state components (except s-abn, s-ctr, or components of type is-other-den) the components cannot be changed.

$$\alpha \neq s\text{-}ctr \wedge \neg is\text{-}other\text{-}den(\alpha \circ elem(k,BL^i)) \wedge k \leq j \supset$$
$$\alpha \circ elem(k,BL^j) = \alpha \circ elem(k,BL^i)$$

where $j = \underset{j}{MAX}(length(BL^j) \leq length(BL^i) \wedge length(BL^{j-1}) + 1 = length(BL^j))$

A proof by induction (using 5-2) can be constructed on those lists of states which comprise the computation beginning with the element where the conditions for j are fulfilled.

Property 5-4: This property supplements 2-12 in respect to the F-model and TWIN-model. It states the relation between the s-f sub-components and the index of the element of BL containing them.

a) $1 < k \leq length(BL) \supset is\text{-}parm\text{-}den(id(DN_k)) \supset s\text{-}f \circ id(DN_k) < k$
b) $1 < k \leq length(BL) \supset d\text{-}f(id,k,BL) \neq \Omega \supset d\text{-}f(id,k,BL) < k$

The proof of a) follows from the fact that the evaluation of the argument passed to the parameter id was performed in the preceding block. Note that either the argument was not a parameter in which case the s-f component was set to k-1 or, using induction and 5-3, a) was true for the argument and consequently it is true for id.

The property b) is only a generalization of a) and follows immediately from the definition of d-f (T19) together with a).

Property 5-5: This property states that the s-b and s-dcl components of a parameter are equal to the corresponding components of $d(id_o,k_o,bl)$ where id_o was the argument passed in its own right (i.e. id_o was not a parameter) from the element of the block-list bl, whose index is k_o.

$$\begin{aligned}&1 < k \le \text{length}(BL) \supset\\ &\quad \text{is-parm-den}(d(id,k,BL)) \wedge id_o = \text{d-id}(id,k,BL) \wedge k_o = \text{d-f}(id,k,BL) \supset\\ &\qquad \neg\text{is-parm-den}(d(id_o,k_o,BL)) \wedge\\ &\qquad \text{d-b}(id,k,BL) = \text{d-b}(id_o,k_o,BL) \wedge\\ &\qquad \text{d-d}(id,k,BL) = \text{d-d}(id_o,k_o,BL)\end{aligned}$$

This property follows

a) from the definition of the function m-p which, in the case that id_o is not a parameter, adds id_o and k_o to $d(id_o,k_o,bl)$ otherwise makes no change

b) from property 5-3 which states that parameter denotations are not changed.

Lemma 5-6: This lemma shows that assuming axiom 5-1 there do not exist two entries of the same name in the denotation elements of a static chain.

$$1 < k \le \text{length}(BL) \supset (d(id,EPA_k,BL) \ne \Omega \supset id(DN_k) = \Omega)$$

Proof: $d(id,EPA_k,BL) \ne \Omega$ implies according to 2-11c that there exist α and β such that is-intr$(id,\beta(TX))$ and $\alpha\circ\beta = PTR_{EPA_k}$.

Since, according to 2-2a, there exists an α' with $\alpha' \ne I$ and $\alpha'\circ PTR_{EPA_k} = PTR_k$, it follows from 5-1 that

$$\text{is-intr}(id,\underbrace{\alpha'\circ\alpha\circ\beta}_{PTR_k}(TX)) \text{ cannot be true.}$$

$\neg$is-intr$(id,PTR_k(TX))$ is equivalent to $id(DIN_k) = \Omega$ (T11,T12) and this concludes the proof.

Lemma 5-7: This lemma shows that the result of a search of the static chain, from some element of BL, can also be obtained by beginning the search at a previous element of any static chain passing through that element of BL.

$$\begin{aligned}&1 < k \le \text{length}(BL) \supset\\ &\quad \neg\text{is-parm-den}\circ d(id',k,BL) \wedge d(id,\text{d-b}(id',k,BL),BL) \ne \Omega \supset\\ &\qquad\qquad d(id,\text{d-b}(id',k,BL),BL) = d(id,k,BL)\end{aligned}$$

Proof:

1 If $k = 1$ then the lemma is vacously true.

2 We assume that the lemma is true for all k_1 with $1 \leq k_1 < k \leq \text{length(BL)}$. We will show that it holds also for k,

3 $\neg$is-parm-den$\circ$d(id',k,BL)

4 $\text{d(id,d-b(id',k,BL),BL)} \neq \Omega$

5 $\text{d-b(id',k,BL)} > 1$ 4,T17,T16

6 $\text{d(id, d-b(id',k,BL),BL)} = \text{id'(DN}_k) \neq \Omega \longrightarrow \text{d(id,k,BL)}$ T17,5,3
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{T} \longrightarrow \text{d(id,d-b(id',EPA}_k\text{,BL),BL)}$

Now we make a case distinction on whether $\text{id'(DN}_k)$ is Ω or not.

case:

7 $\text{id'(DN}_k) = \Omega$

8 $\text{d(id,d-b(id',EPA}_k\text{,BL),BL)} \neq \Omega$ 7,6,4

9 $\neg$is-parm-den(d(id',EPA$_k$,BL)) 7,6,3

10 $\text{d(id,d-b(id',EPA}_k\text{,BL),BL)} = \text{d(id,EPA}_k\text{,BL)}$ IH2,8,9,2-12a

11 $\text{d(id,EPA}_k\text{,BL)} \neq \Omega$ 8,10

12 $\text{id(DN}_k) = \Omega$ 11,5-6

13 $\text{d(id,EPA}_k\text{,BL)} = \text{d(id,k,BL)}$ 11,12,T16

14 $\text{d(id,d-b(id',k,BL),BL)} = \text{d(id,k,BL)}$ 6,7,10,13

case:

15 $\text{id'(DN}_k) \neq \Omega$

16 $\text{d(id,d-b(id',k,BL),BL)} = \text{d(id,k,BL)}$ 15,6

14 and 16 are the last lines of the two cases and both prove the property, therefore the lemma is proved.

5.2.3 Proof

This section contains a formal proof of the relation of the functions d and f in the twin machine (5-8), followed by the argument of the transition from the twin machine to that in section 5.1 (5-9).

<u>Theorem 5-8</u>: $1 < k \leq \text{length}(BL) \supset d(id,k,BL) \neq \Omega \supset d(id,k,BL) = f(id,k,BL)$

1 $\quad 1 < k \leq \text{length}(BL)$

Assumption 1, property 5-3 and the definition of <u>inst-bl</u> and <u>inst-proc</u> imply that each EPA-pointer has following value:
Let $nm = \text{s-nm}(ST_{k-1})$

2 $\quad EPA_k = (\text{is-call}(ST_{k-1}) \longrightarrow \text{d-b}(nm,k-1,BL)$
$\qquad\qquad T \longrightarrow k-1)$

Consider the case that $\text{is-call}(ST_{k-1})$:

Assumption 1, property 5-3 and the definition of <u>int-st</u> and <u>inst-proc</u> imply that F_k is defined by d-f:

3 $\quad \text{is-call}(ST_{k-1}) \supset F_k = \text{d-f}(nm,k-1,BL)$ T19

4 $\quad \text{d-f}(nm,k-1,BL) \neq \Omega \equiv \text{is-parm-den}(d(nm,k-1,BL))$ T7,T19

Let $id_o = \text{d-id}(nm,k-1,BL)$

5 $\quad \text{is-parm-den}(d(nm,k-1,BL)) \supset \neg\,\text{is-parm-den}(d(id_o,F_k,BL)) \wedge$
$\qquad\qquad \text{d-b}(id_o,F_k,BL) = \text{d-b}(nm,k-1,BL)$ 3,4,5-5

Now we can rewrite the fomula for EPA_k:

6 $\quad EPA_k = (\text{is-call}(ST_{k-1}) \longrightarrow (F_k \neq \Omega \longrightarrow \text{d-b}(id_o,F_k,BL),$
$\qquad\qquad\qquad T \longrightarrow \text{d-b}(nm,k-1,BL)),$
$\qquad\quad T \longrightarrow k-1)$ 2,4,5

7 $\quad F_k \neq \Omega \supset \text{is-call}(ST_{k-1})$ T12

Using these results we can transform the function d :

8 $\quad d(id,k,BL) = (k \leq 1 \longrightarrow \Omega,$
$\qquad\qquad \text{is-parm-den}(id(DN_k)) \longrightarrow id(DN_k),$ T16
$\qquad\qquad id(DN_k) \neq \Omega \longrightarrow \mu(\langle s\text{-}b{:}k\rangle,\langle s\text{-}dcl{:}id(DN_k)\rangle),$
$\qquad\qquad T \longrightarrow (F_k \neq \Omega \longrightarrow d(id,\text{d-b}(id_o,F_k,BL),BL),$ 6,7
$\qquad\qquad\qquad \text{is-call}(ST_{k-1}) \longrightarrow d(id,\text{d-b}(nm,k-1,BL),BL),$
$\qquad\qquad\qquad T \longrightarrow d(id,k-1,BL)))$

With the hypothesis $d(id,k,BL) \neq \Omega$ and 5-6 we get a simplification of 8. Note that if $F_k \neq \Omega$ then $\neg\,\text{is-parm-den}(d(id_o,F_k,BL))$ (see 5) and if $F_k = \Omega$ then $\neg\,\text{is-parm-den}(d(nm,k-1,BL))$ (see 4).

9 $d(id,k,BL) = (k \leq 1 \longrightarrow \Omega$ 8,7,5,4,5-7

$\quad is\text{-}parm\text{-}den(id(DN_k)) \longrightarrow id(DN_k)$

$\quad id(DN_k) \neq \Omega \longrightarrow \mu(<s\text{-}b:k>,<s\text{-}dcl:id(DN_k)>)$

$\quad F_k \neq \Omega \longrightarrow d(id,F_k,BL)$

$\quad is\text{-}call(ST_{k-1}) \longrightarrow d(id,k\text{-}1,BL)$

$\quad T \longrightarrow d(id,k\text{-}1,BL))$

Noting, that the last two alternatives are identical, we see that the algorithm of the transformed function d is the same as f. Therefore, the two functions are equal on the domain given by the hypothesis.

Theorem 5-9: The TWIN and F-search models are equivalent.

Proof: Using theorem 5-8 we change in TWIN all occurences of function d or its derivates to f or its derivates. This is possible since as it is stated in chapter one all programs are proper (see 2-10). That means all occurences of d are defined therefore d never yields Ω. This change removes the need for the s-epa components which are deleted.

After this change in TWIN we have almost the F-search model. The last difference is is-parm-den and the instructions and function handling it. It can easily be seen that the s-id component is never used. This justifies the last transition to the F-search model thus:

1) we omit the last argument in m-p

2) we add the s-f component to the passed object only in the case of procedures.

6. IMPLEMENTATION USING A DISPLAY OF THE F-CHAIN

The reader will recall that a display, which is a vector of pointers to the dynamic areas, provides the link between some index appended to the references and the dynamic area in which the storage for the referenced value is located. In section 3 the static chain suggested a display which used the static depth of the declaring block as an index. We now use a display which will store the elements of the F-chain which was described in the preceding section. The index to be used on this occasion will be unique block names assigned in a prepass.

As has been explained, any display must be updated dynamically and the technique given in this section will adopt the wastefull principle of retracing the complete F-chain and storing it into the display each time the list of dynamic areas changes its length. A considerably economy over this is illustrated in section 7.

6.1 The Model

The transition from the model of section 5.1 to that given below is accomplished by the addition of, and provision for maintenance and use of, a global display vector (DISP). The maintenance is performed by replacing the display be a completely new trace of the F-chain (formed by invoking rev-disp) each time the length of BL changes.

The function df accepts a reference and uses its block name part to select the element of DISP which points to the dynamic area for this reference.

6.1.1 State

```
FD1    is-state = (<s-bl:is-bl-list>,
                   <s-tx:is-p-block>,
                   <s-disp:({<bn:is-int> || is-bn(bn)})>,
                   <s-abn:(<s-b:is-int>,<s-dcl:is-int>) v is-Ω>,<s-c:is-c>)

FD2    is-bl = (<s-f:is-int v is-Ω>,
                <s-ptr:is-sel>,
                <s-bn:is-bn>,
                <s-ctr:is-int v is-Ω>,
                <s-dn:( { <id:is-den> || is-id(id)})>)

FD3    is-den = is-proc-den v is-lab-den v is-parm-den v is-other-den
```

FD4 is-proc-den = is-int

FD5 is-lab-den = is-int

FD6 is-other-den ...

FD7 is-parm-den = (<s-b:is-int>,<s-dcl:is-int>,<s-f:is-int v is-Ω>)

FD8 is-c = see /2/

FD9 is-p-block etc. as is-block except:

a) is-id (except in s-dp or s-pp) changed to is-ref
   is-ref = (<s-id:is-id>,<s-bn:is-bn>)

b) is-block and is-proc-atr have the additional component <s-bn:is-bn>

Abbreviations used:

```
BL = s-bl(ξ)
TX = s-tx(ξ)
DISP = s-disp(ξ)
ABN = s-bn(ξ)
B = last(BL)
PTR = s-ptr(B)
CTR = s-ctr(B)
F = s-f(B)
DN = s-dn(B)
BN = s-bn(B)
P = CTR ≠ Ω  ——→ elem(CTR)∘s-sp∘PTR
    T  ——→ s-body∘PTR
ST = P(TX)
cur(bl) = elem(length(bl))∘s-bl
```

Initial state:

```
FD0  ξ₀ = μ₀(<s-bl:[μ₀(<s-ptr:I>,
                        <s-ctr:0>)]>,
            <s-tx:μ₀(<s-sp:[t₀]>,
                     <s-dp:[]>)>,
            <s-c:int-next-st>)
```

where t_o satisfies the two axioms 6-1 and 6-2.

6.1.2 State transition function

```
FD10  int-next-st =

         CTR < length(s-sp∘PTR(TX)) ∧ ABN = Ω  ──→ int-next-st;
                                                      int-st;
                                                        step-ctr

         s-b(ABN) = length(BL)  ──→ int-next-st;
                                       int-st;
                                         set-ctr

         T ──→ null

FD11  int-st =

         is-p-block(ST) ──→ rev-disp;
                               epilogue;
                                 int-next-st;
                                   rev-disp;
                                     inst-bl

         is-p-call(ST)  ──→ rev-disp;
                              epilogue;
                                epilogue;
                                  int-next-st;
                                    rev-disp;
                                      inst-bl;    length(s-ap(ST))
                                        inst-proc( LIST   m-p(df(elem(i,s-ap(ST)),BL,DISP),
                                                   i=1                                  BL,TX),
                                       s-atr∘elem(d)∘s-dp∘(s-ptr(elem(b,BL))),f)

         is-p-goto(ST) ──→ s-abn:df(ST,BL,DISP)
                          ⋮

      where: d = df-d(s-nm(ST),BL,DISP)
             b = df-b(s-nm(ST),BL,DISP)
             f = df-f(s-nm(ST),BL,DISP)

FD12  inst-bl =

         s-bl:BL⁀[μ_0(<s-ptr:P>,
                      <s-bn:s-bn∘P(TX)>,
                      <s-ctr:0>,
                      <s-dn:μ_0({<s-id∘elem(i,s-dp(ST)):i> |
                                    ¬is-p-other-atr(s-atr∘elem(i,s-dp(ST)))}∪...)>)]
```

FD13 <u>inst-proc</u>(arg-1,s,f) =

s-bl:BL $\frown$ [μ_o(<s-f:f>,
<s-ptr:s>,
<s-bn:s-bn∘s(TX)>,
<s-dn:μ_o({<elem(i,s-pp(s(TX))):elem(i,arg-1)> | 1 ≤ i ≤ length(arg-1)})>)]

FD14 <u>step-ctr</u> = s-ctr∘cur(BL):CTR + 1

FD15 <u>set-ctr</u> = s-ctr∘cur(BL):s-atr∘elem(s-dcl(ABN),s-dp(PTR(TX)))
s-abn:Ω

FD16 <u>epilogue</u> = cur(BL):Ω

FD17 <u>rev-disp</u> = s-disp:upd-disp(BL,length(BL))

FD18 upd-disp(bl,k) = k ≤ 1 $\rightarrow$ Ω
s-f∘elem(k,bl) ≠ Ω $\rightarrow$ μ(upd-disp(bl,s-f∘elem(k,bl));
<s-bn∘elem(k,bl):k>)
T $\rightarrow$ μ(upd-disp(bl,k-1);<s-bn∘elem(k,bl):k>)

FD19 df-b(ref,bl,disp) = s-b(df(ref,bl,disp))

FD20 df-d(ref,bl,disp) = s-dcl(df(ref,bl,disp))

FD21 df-f(ref,bl,disp) = s-f(df(ref,bl,disp))

FD22 df(ref,bl,disp) = adr(s-id(ref),s-bn(ref)(disp),bl)

FD23 adr(id,b,bl) = is-parm-den∘id∘s-dn∘elem(b,bl) $\rightarrow$ id∘s-dn∘elem(b,bl),
T $\rightarrow$ μ_o(<s-b:b>,<s-dcl:id∘s-dn∘elem(b,bl)>)

FD24 m-p(den,bl,t) = s-f(den) ≠ Ω ∨ ¬is-proc-atr∘elem(s-dcl(den))∘s-dp∘s-ptr∘
elem(s-b(den),bl)(t) $\rightarrow$ den,
T $\rightarrow$ μ(den;<s-f:length(bl)>)

<u>6.1.3 Properties</u>

This model assumes that blocknames occur in the text. These names are not in the language, but must be inserted by a prepass. Two conditions must be fulfilled by the prepass. The first states the uniqueness of the names:

Axiom 6-1:

$$(\text{is-p-block} \vee \text{is-p-proc-atr})(\alpha(t)) \wedge (\text{is-p-block} \vee \text{is-p-proc-atr})(\beta(t)) \supset$$
$$\text{s-bn}\circ\alpha(t) = \text{s-bn}\circ\beta(t) \supset \alpha = \beta$$

The second shows the relation between the name in the reference and the name of the block where the identifier of the reference is declared.

Axiom 6-2:

$$(\text{is-ref}(\alpha(t)) \supset (\exists\beta)((\text{is-p-block} \vee \text{is-p-proc-atr})(\beta(t)) \wedge$$
$$\text{s-bn}\circ\alpha(t) = \text{s-bn}\circ\beta(t) \wedge (\exists\gamma)(\alpha = \gamma\circ\beta \wedge$$
$$\neg\text{is-red}(\text{s-id}\circ\alpha(t),\gamma,\beta(t)) \wedge \text{is-intr}(\text{s-id}\circ\alpha(t),\beta(t))$$

6.2 Justification

We shall now show that the model of section 6.1 is equivalent to that of section 5.1. Since the display always represents the current chain, it is only necessary to show that the unique block name assigned to each reference corresponds to the first place that the F-search would have found a denotation of the variable. This follows from the observation already made in 5.2 about the F-chain.

6.2.1 Twin model

In this chapter it is not necessary to explain the TWIN model in detail. Only the difference between the TWIN model and display model is stated:

All occurences of df, df-b, df-d, df-f are replaced by f1, f1-b, f1-d, f1-f. f1 is a version of a modified function of f, which accepts objects of type is-ref instead of type is-id.

T1 $\text{f1}(\text{ref},k,\text{bl}) = k \leq 1 \longrightarrow \Omega$
$\text{s-bn}(\text{ref}) = \text{s-bn}\circ\text{elem}(k,\text{bl}) \longrightarrow \text{adr}(\text{s-id}(\text{ref}),k,\text{bl})$
$\text{s-f}\circ\text{elem}(k,\text{bl}) \neq \Omega \longrightarrow \text{f1}(\text{ref},\text{s-f}\circ\text{elem}(k,\text{bl}),\text{bl})$
$T \longrightarrow \text{f1}(\text{ref},k-1,\text{bl})$

T2 $\text{f1-b}(\text{ref},k,\text{bl}) = \text{s-b}\circ\text{f1}(\text{ref},k,\text{bl})$

T3 $\text{f1-d}(\text{ref},k,\text{bl}) = \text{s-dcl}\circ\text{f1}(\text{ref},k,\text{bl})$

T4 $\text{f1-f}(\text{ref},k,\text{bl}) = \text{s-f}\circ\text{f1}(\text{ref},k,\text{bl})$

In this new model the F-search model is embedded. Since,

1) the display is always updated but never referenced,

2) the difference that we use a composed name of type is-ref instead of a simple name of type is-id, does not affect the result. We have only to show that a similar axiom to 5-1 holds for the programs accepted by TWIN and that f is similar to f1.

Axiom 6-3:

$$\alpha \neq I \wedge \text{is-p-intr}(\text{s-id}(ref),\beta(t)) \wedge \text{s-bn}(ref) = \text{s-bn}\circ\beta(t) \supset \neg(\text{is-p-intr}(\text{s-id}(ref),\alpha\circ\beta(t)) \wedge \text{s-bn}(ref) = \text{s-bn}\circ\alpha\circ\beta(t))$$

Lemma 6-4: This lemma shows that any text with unique block names will necessarily satisfy 6-3.
6-1 implies 6-3.

Proof: Consider the counter example to 6-4 α_o, β_o, ref_o and t_o:
It must follow $\text{s-bn}(ref_o) = \text{s-bn}\circ\beta_o(t_o)$ and
$\text{s-bn}(ref_o) = \text{s-bn}\circ\alpha_o\circ\beta_o(t_o)$ when $\alpha_o\circ\beta_o(t_o)$ and $\beta_o(t_o)$ are blocks but $\text{s-bn}\circ\beta_o(t_o) = \text{s-bn}\circ\alpha_o\circ\beta_o(t_o)$ with $\alpha_o \neq I$ contradicts 6-1. Thus the implication is true.

The next lemma shows that it is possible to ask only for the s-bn of a reference and not also the s-id component.

Lemma 6-5: $\text{s-bn}(ref) = \text{s-bn}\circ\text{elem}(k,BL) \supset \text{s-id}(ref)\circ\text{s-dn}\circ\text{elem}(k,BL) \neq \Omega$

Proof:

1	$\text{s-bn}(ref) = BN_k$	
2	$BN_k = \text{s-bn}\circ PTR_k(TX)$	FD11,FD12
3	$(\exists\beta)((\text{is-p-block} \vee \text{is-p-proc-atr})(\beta(TX)) \wedge \text{s-bn}(ref) = \text{s-bn}\circ\beta(TX) \wedge \text{is-p-intr}(\text{s-id}(ref),\beta(TX))$	6-2
4	$\text{is-intr}(\text{s-id}(ref),PTR_k(TX))$	2,3, 6-1
5	$\text{s-id}(ref)(DN_k) \neq \Omega$	4,FD11,FD12

This completes the proof.

Lemma 6-5 shows that adr(s-id(ref),k,BL) is defined if $\text{s-bn}(ref) = BN_k$. Therefore the transformation of the first and second alternative of function f into the second alternative of f1 is valid.

The two lemmas 6-4, 6-5 show that the F-search model is embedded in the TWIN model. The next lemmas reflect the relation between f1 and upd-disp. For this purpose, an auxiliary function is used.

T5 $f2(ref,k,bl) = (k \leq 1 \longrightarrow \Omega,$
$s\text{-}bn(ref) = BN_k \longrightarrow k,$
$F_k \neq \Omega \longrightarrow f2(ref,F_k,bl),$
$T \longrightarrow f2(ref,k\text{-}1,bl))$

<u>Lemma 6-6</u>: $f1(ref,k,BL) \neq \Omega \supset f1(ref,k,BL) = adr(s\text{-}id(ref),f2(ref,k,BL),BL)$

Proof:

1 $f1(ref,k,BL) \neq \Omega$

2 $adr(s\text{-}id(ref),f2(ref,k,BL),BL) = s\text{-}bn(ref) = BN_k \longrightarrow adr(s\text{-}id(ref),k,BL),$
$F_k \neq \Omega \longrightarrow adr(s\text{-}id(ref),f2(ref,F_k,BL),BL),$
$T \longrightarrow adr(s\text{-}id(ref),f2(ref,k\text{-}1,BL),BL)$
1,FD23,T5

On the subset of the domain of f1, where f1 is not Ω, both functions f1(ref,k,BL) and adr(s-id(ref),f2(ref,k,BL),BL) are the same algorithms, therefore they are equal on this subset.

<u>Lemma 6-7</u>: $s\text{-}bn(ref)\circ upd\text{-}disp(BL,k) = f2(ref,k,BL)$

Proof:

1 $s\text{-}bn(ref)\circ upd\text{-}disp(BL,k) = k \leq 1 \longrightarrow \Omega$ FD18
$F_k \neq \Omega \longrightarrow s\text{-}bn(ref)\circ\mu(upd\text{-}disp(BL,F_k);<BN_k:k>)$
$T \longrightarrow s\text{-}bn(ref)\circ\mu(upd\text{-}disp(BL,k\text{-}1),<BN_k:k>)$

2 $s\text{-}bn(ref)\circ upd\text{-}disp(BL,k) = k \leq 1 \longrightarrow \Omega$ 1
$F_k \neq \Omega \longrightarrow (BN_k = s\text{-}bn(ref) \longrightarrow k,$
$T \longrightarrow s\text{-}bn(ref)\circ upd\text{-}disp(BL,F_k))$
$T \longrightarrow (BN_k = s\text{-}bn(ref) \longrightarrow k,$
$T \longrightarrow s\text{-}bn(ref)\circ upd\text{-}disp(BL,k\text{-}1))$

3 $s\text{-}bn(ref)\circ upd\text{-}disp(BL,k) = k \leq 1 \longrightarrow \Omega$ 2
$BN_k = s\text{-}bn(ref) \longrightarrow k$
$F_k \neq \Omega \longrightarrow s\text{-}bn(ref)\circ upd\text{-}disp(BL,F_k)$
$T \longrightarrow s\text{-}bn(ref)\circ upd\text{-}disp(BL,k\text{-}1)$

The domain of f2(ref,k,bl) is equal to the domain of s-bn(ref)∘upd-disp(BL,k) the algorithms of both functions are equal, therefore they are equivalent functions.

6.2.2 Main theorem

Theorem 6-8: In the TWIN model f1 and its derivatives can be replaced by df and its derivatives.

$$f1(ref,length(BL),BL) \neq \Omega \supset f1(ref,length(BL),BL) = df(ref,BL,DISP)$$

Proof: Immediately after the execution of rev-disp it is true that

$$\begin{aligned} f1(ref,length(BL),BL) &= adr(s\text{-}id(ref),f2(ref,length(BL),BL),BL) && \text{6-6,T1}\\ &= adr(s\text{-}id(ref),s\text{-}bn(ref)\circ upd\text{-}disp(BL,length(BL)),BL) && \text{6-7}\\ &= adr(s\text{-}id(ref),s\text{-}bn(ref)(DISP),BL) && \text{FD17}\\ &= df(ref,BL,DISP) && \text{FD22} \end{aligned}$$

Now we must show that for any case of df, none of the critical elements (i.e., DISP, F pointers and the constant parts of the denotations) can have been changed since the last call to rev-disp. This follows immediately from an inspection of TWIN.

The proof of this theorem concludes the justification that the F-search and the F-display model are equivalent.

7. OPTIMIZATION OF THE F-CHAIN DISPLAY

This section explains how the display can be updated, without retracing the F-chain, in all cases except a procedure invocation via a parameter. Thus, the display can be updated as follows:

a) direct invocation of a non-recursive block - replace the relevant element of the display with the address of the new dynamic area (7-3)

b) exit from same - do nothing (7-4,7-5)

c) direct invocation of a recursive block (7-3) - copy the old value of, and replace by the address of the new dynamic area, the relevant display element

d) exit from same - replace the relevant element by the saved copy (7-4)

e) entry or exit to a procedure invoked via a parameter or after goto statement - retrace the F-chain.

Notice that the display now contains more information than one trace through the F-chain. It in fact has the latest invocation, if any, of all blocks even if they are not in the F-chain. Here "latest" takes into account that a call of a procedure via a parameter may cause other than the last invoked to be required as "latest". This links back to our elementary discussion of the problem in section 1.

No model is given in this section but Appendix II contains a non-recursive version of the fully optimized F-chain.

7.1 Properties of the FD-Model

For optimization purposes a new block-local component is added to the state. This component, called display-field (s-df), gets the old value of that display component bn on entering the block or procedure identified by bn. When leaving an activation the display field is restored. Now let ξ be any state and ξ' its successor.

Property 7-1: This property shows that s-f and s-df components are not changed, except by deletion or insertion.

$$1 \le k \le \min(\text{length}(BL),\text{length}(BL')) \Rightarrow \\ \text{s-f}\circ\text{elem}(k,BL) = \text{s-f}\circ\text{elem}(k,BL') \\ \text{s-df}\circ\text{elem}(k,BL) = \text{s-df}\circ\text{elem}(k,BL'))$$

The proof is a corollary of 5-2.

Property 7-2: DISP = upd-disp(BL,length(BL)) if ξ is not a state ready for rev-disp or a state after the execution of epilogue, inst-proc or null.

Proof follows immediately from property 7-1 and the inspection of the model.

Lemma 7-3: This lemma shows that the DISP component of a dynamic area, which was not installed via a parameter, differs from the preceding DISP component by only one element.

$$F' = \Omega \wedge \text{length}(BL') = \text{length}(BL) + 1 \supset \text{upd-disp}(BL',\text{length}(BL')) = \mu(DISP;<BN':\text{length}(BL')>)$$

Proof:

$$\begin{aligned}\text{upd-disp}(BL',\text{length}(BL')) &= \text{length}(BL') \leq 1 \longrightarrow \ldots && \text{FD18}\\ &\quad F' \neq \Omega \longrightarrow \ldots\\ &\quad T \longrightarrow \mu(\text{upd-disp}(BL',\text{length}(BL') - 1); <BN':\text{length}(BL')>)\\ &= \mu(\text{upd-disp}(BL,\text{length}(BL));<BN':\text{length}(BL')>) && \text{7-1}\\ &= \mu(DISP;<BN':\text{length}(BN')>) && \text{7-2}\end{aligned}$$

Lemma 7-4: This lemmas shows that on deleting a dynamic area which was not installed via a parameter, the DISP differs by only one element.

$$F = \Omega \wedge \text{length}(BL') = \text{length}(BL) - 1 \supset \text{upd-disp}(BL',\text{length}(BL')) = \mu(DISP;<BN:\text{s-df}(B)>)$$

Proof: s-df(b) = BN(disp) where disp was the display on entering this activation. The s-f and s-df components are unchanged from entering this activation until leaving it (7-1). Therefore

let upd-disp(BL',length(BL')) = disp

$$\begin{aligned}\mu(DISP;<BN:\text{s-df}(B)>) &= \mu(\text{upd-disp}(BL,\text{length}(BL));<BN:BN(disp)>)\\ &= (k \leq 1 \longrightarrow \ldots && \text{FD18}\\ &\quad F \neq \Omega \longrightarrow \ldots\\ &\quad T \longrightarrow \mu(\mu(\text{upd-disp}(BL',\text{length}(BL'));<BN:\text{length}(BL)>); <BN:BN(disp)>))\\ &= \mu(\mu(disp;<BN:\text{length}(BL)>);<BN:BN(disp)>)\\ &= disp\end{aligned}$$

The remainder of the optimization makes use of the knowledge of whether a block or procedure is used recursively or not.

If a block or a procedure cannot occur twice in the dynamic chain, its display field in its only activation is always Ω. Storing and restoring is not necessary. But if restoring is not done we get a display which has not only name occurring in the current F-chain but also dead portions of an old F-chain. This does not matter, if we use a modified update-display function.

```
FD25  upd-disp1(bl,k,disp) =
        (k ≤ 1 ⟶ disp
         s-f∘elem(k,bl) ≠ Ω ⟶ μ(upd-disp1(bl,s-f∘elem(k,bl),disp);
                                                   <s-bn∘elem(k,bl):k>),
         T ⟶ μ(upd-disp1(bl,k-1,disp);<s-bn∘elem(k,bl):k>))
```

We have to show that the above two lemmas remain true for this function.

Lemma 7-5: $bn \circ upd\text{-}disp(bl,k) \neq \Omega \Rightarrow bn \circ upd\text{-}disp(bl,k) = bn \circ upd\text{-}disp1(bl,k,d_o)$, where d_o is any object.

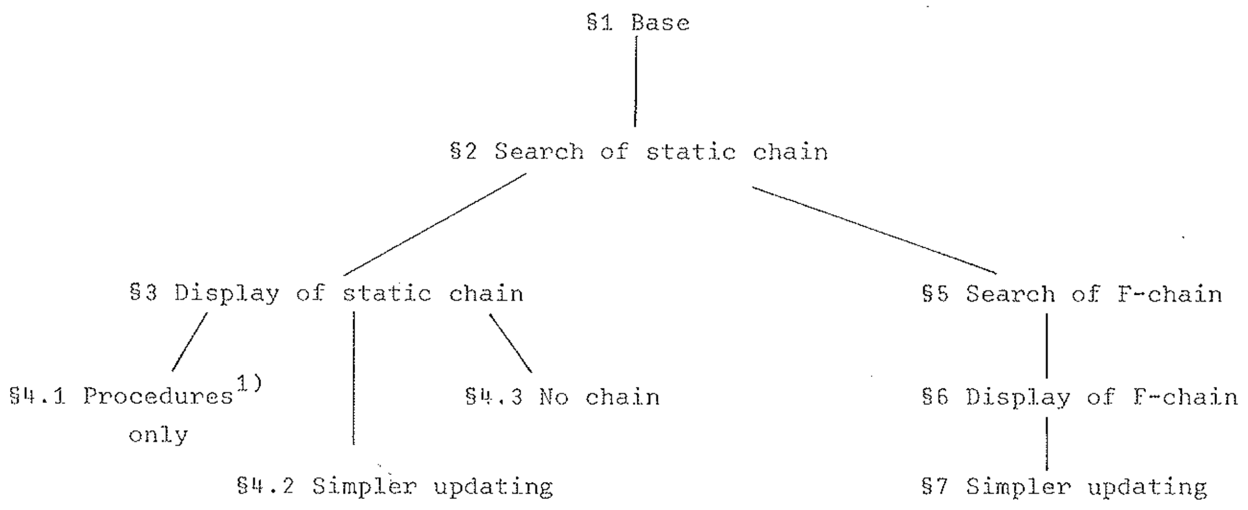
Proof: $bn \circ upd\text{-}disp(bl,k) = (k \leq 1 \longrightarrow bn(\Omega),$
$s\text{-}f \circ elem(k,bl) \neq \Omega \longrightarrow bn \circ \mu(\ldots)$
$T \longrightarrow bn \circ \mu(\ldots)$

Since $bn \circ upd\text{-}disp(k,bl) \neq \Omega$ the first alternative is never true and we can write instead of $bn(\Omega)$ any object, e.g., d_o

$= (k \leq 1 \longrightarrow d_o,$
$s\text{-}f \circ elem(k,bl) \longrightarrow bn \circ \mu(\ldots)$
$T \longrightarrow bn \circ \mu(\ldots)$
$= bn \circ upd\text{-}disp1(bl,k,d_o)$

8. SUMMARY

This section attempts to give an overview, and some comments on the efficiency of the models contained in this report. First a figure is given showing how the models were derived above.



Not only was each model derived from its predecessor, but its equivalence to that predecessor was also justified. Thus we are assured that all of the models are equivalent. With this knowledge an implementor will naturally wish to know which model is the most efficient. There is no clearcut answer to this question, because of its dependence on the operating environment of the object program, but the following informal comments attempt to show some of the parameters on which a decision could be based.

The first consideration must be the difference in the chaining technique which splits the methods into two sets. The static chain is, in general, the shortest chain which will locate all of the dynamic variables referencable in a given block, whereas the F-chain would appear to be the longest such chain. Thus if implementing a search based model one is unlikely to seak further than the static chain.

However, as was observed in section 2, display type implementations tend to provide more efficient object time reference methods, since using such methods permits reference to any variable with just one indirect step. In particular, if sufficient hardware index registers are available for the display, the reference would be very fast. Given the limited number of registers on many machines, this is most likely to be practicable, if at all, with a technique based on that of section 4.1 (first the static chain display is the shortest possible, second the maximum number of elements is only the static procedure depth).

---

[1] The three optimizations of section 4 can be, and for the purposes of this chapter are considered as separate optimizations on the model of section 3.

Assuming that the greater length of the F-chain display does not make a critical difference such as preventing an index register implementation, the implementor would be interested in the comparitive amount of work required to update both displays. The results, given in detail in section 3.1 columns F and I of the following table, can be summarized by saying that the updating of the F-display requires less steps on exit from non-recursive or recursive procedures; more steps for entry or exit from proce-dures invoked via a parameter and for the goto case; a roughly equal number of steps in other cases.

Source	A Pure Copy Section 1	B Complete Environment /7/	C Local Environment /1/ [1)]
1. Prepass	none	none	none
2. Directories:			
ENV: names $\longrightarrow$ unique names	none	local/ complete	local/ partial
DEN: (unique) names $\longrightarrow$ values	global/ complete	as A	as A
3. Operations:			
3.1 Block Transition [3)]			
open block/proc { non-recursive, recursive, parameter }	change text	form complete environment	store partial environments/ update chain
close block/proc { non-recursive, recursive, parameter, goto }	revert to old text	revert to old environment	delete last local environment
3.2 Reference	un(DEN)	n(ENV)(DEN)	ss(n)

Key: global - immediate component of ξ
local - component of each element of s-bl(ξ)
partial - only contains local entries
complete - contains any entries required at this time
n - name
un - unique name
ss - function for searching static chain } if no argument given returns a
sf - function for searching F-chain } vector of values
i - index of the new dynamic area

8.

D Search static chain Section 2	E Display static chain Section 3	F Optimize update E Section 4.2[2)]	G Search F-chain Section 5	H Display F-chain Section 6	I Optimize update H Section 7
none	add depths to names	as E	make names on static chain unique	add block names to names	as H
none	as D	as D	as D	as D	as D
local/partial					
store partial denota-tions/ update chain	as D + DISP←ss	as D + DISP[dep]←i	store partial denota-tions/ update chain	as G + DISP←sf	as G + DISP[bn]←i
		as D + DISP[dep]←i			as G + ST←DISP[bn] DISP[bn]←i
		as D + DISP←ss			as G + DISP←sf
delete last local denota-tion	as D + DISP←ss	as D + DISP←ss	delete last local denota-tion	as G + DISP←sf	as G
		as D + DISP←ss			as G + DISP[bn]←ST
		as D + DISP←ss			as G + DISP←sf
		as D + DISP←ss			as G + DISP←sf
ss(n)	n($DEN_{DISP[dep]}$)	as E	sf(n)	n($DEN_{DISP[bn]}$)	as H

Footnotes from preceding table:

1) The inclusion of entries B and C in the table permits a comparison of the proof given in section 2 with that given in /1/. Apart from the lack of goto statements, in the language, the proof of /1/ goes from the complete environment to the local environment models, thus tackling the problem of showing the equivalence of a reference to a complete environment to a search through a chain of local partial environments. Such a proof could utilize the equality of the unique names found as its criterion of correctness. Making the step given in section 2 posed the problem of specifying the correctness criteria in a convenient form. This has occasioned the use of the special unique name generator which supplies the criterion of location of the same denotation element.

2) The optimization described in section 4.1 is omitted for the purposes of this table.

3) The requirement to install or update the DEN component is not mentioned in the table.

REFERENCES

/1/ LUCAS,P.: Two Constructive Realizations of the Block Concept and their Equivalence.- IBM Laboratory Vienna, Techn.Report TR 25.085, 28 June 1968.

/2/ LUCAS,P., WALK,K.: On the Formal Description of PL/I.- Annual Review in Automatic Programming 6 (1969), Part 3, pp.105-182.

/3/ NAUR,P. (Ed.): Revised Report on the Algorithmic Language ALGOL 60.- Comm. ACM 6 (1963), No.1, pp.1-23.

/4/ DIJKSTRA,E.W.: Recursive Programming.- Num.Math. 2 (1960), pp.312-318.

/5/ RANDELL,B., RUSSELL,L.J.: ALGOL 60 Implementation.- London: Academic Press 1964.

/6/ HENHAPL,W., JONES,C.B.: On the Interpretation of Goto Statements in the ULD.- IBM Laboratory Vienna, Lab. Note LN 25.3.065, 3 March 1970.

/7/ WALK,K., ALBER,K., BANDAT,K., BEKIĆ,H., CHROUST,G., KUDIELKA,V., OLIVA,P., ZEISEL,G.: Abstract Syntax and Interpretation of PL/I - ULD-Version 2.- IBM Laboratory Vienna, Techn.Report TR 25.082, 28 June 1968.

/8/ HENHAPL,W.: A Proof of Correctness for the Reference Mechanism to Automatic Variables in the F-Compiler.- IBM Laboratory Vienna, Lab. Note 25.3.048, 19 November 1968.

/9/ IBM System 360/Operating System, PL/I (F): Programmers' Guide.- Form C28-6594.

/10/ LUCAS,P., BEKIĆ,H.: Compilation of Algol: Part I - Organization of the Object Program. IBM Laboratory Vienna, Lab. Rep. LR 25.3.001, May 1962.

ACKNOWLEDGEMENT

The idea for this report was suggested by P.Lucas, to whom the authors are also indepted, along with H.Bekić and other members of the Vienna Laboratory, for encouraging discussions. Thanks are also offered to N.Gard and K.Zimmermann for their careful readings of the manuscript.

APPENDIX I:

This appendix contains a model which, instead of the recursive approach used above, uses an interative control routine which should be more immediately useable for implementation purposes. The transition made here is discussed more fully in /6/. The model presented is that which was described in section 4.2.

```
is-state = (<s-pl:is-pl-list>,<s-tx:is-p-proc-atr>,
            <s-c:is-c>,<s-disp:is-int-list>)

is-pl = (<s-epa:is-int v is-Ω>,
         <s-ptr:is-sel>,
         <s-ctr:is-int v is-Ω>,
         <s-btr:is-sel>,
         <s-dn:is-den-list>)

is-den = is-proc-den v is-lab-den v is-parm-den v is-other-den

is-proc-den = is-sel

is-lab-den = (<s-btr:is-sel>,<s-dec:is-int>)

is-other-den = ...

is-parm-den = (<s-p:is-int>,<s-dcl:is-proc-den v is-lab-den v is-other-den>)

is-c = see /2/

is-p-block etc. as is-block except:

a) is-id changed to is-ref
   is-ref = (<s-off:is-int>,<s-dep:is-int>)
b) is-proc-atr has the additional component
   <s-dep:is-int>

Abbreviations used:
PL = s-pl(ξ)
TX = s-tx(ξ)
DISP = s-disp(ξ)
PR = last(PL)
EPA = s-epa(PR)
PTR = s-ptr(PR)
CTR = s-ctr(PR)
BTR = s-btr(PR)
```

```
DN = s-dn(PR)
DEP = s-dep(PTR(TX))
Q = CTR ≠ Ω  → elem(CTR)∘s-sp∘BTR
    T  → s-body∘BTR
P = Q∘PTR
ST = P(TX)
cur(pl) = elem(length(pl))∘s-pl
```

Initial state:

```
ξ_0 = μ_0(<s-pl:[μ_0(<s-ptr:s-body>,
                      <s-btr:I>,
                      <s-ctr:1>,
                      <s-dn:[]>)]>,
          <s-tx:μ_0(<s-body:[prep(t_0)]>,<s-dep:0>,
                    <s-pp:[]>)>,
          <s-disp:[]>,
          <s-c:int-next-st>)
```

where prep must satisfy 4-1.

State transition function:

```
int-st =
   is-p-block(ST)  → int-st;
                       inst-bl
   is-p-call(ST)  → int-st;
                      inst-bl;
                        inst-proc( LIST_{i=1}^{length(s-ap(ST))} dd(elem(i,s-ap(ST)),PL,DISP),
                                   ,s-atr∘d∘s-ptr(elem(p,PL)),p,parm-inf)
   is-p-goto(ST)  → int-st;
                      set-btr(dd-d(ST,PL,DISP));
                        goto(dd-p(ST,PL,DISP))
                  : 1)
   CTR ≠ Ω  → int-st;
                step-ctr;
                  reset-btr
```

cont'd

1) any other statement must call step-ctr after execution.

```
    length(PL) > 1 ⟶ int-st;
                       step-ctr;
                         epilogue
    T ⟶ null

 where: d = dd-d(s-nm(ST),PL,DISP)
        p = dd-p(s-nm(ST),PL,DISP)
        parm-inf = is-parm-den(elem(s-off(ref),s-dn(elem(elem(s-dep(ref),DISP),PL))))

 inst-bl =

    s-ctr∘cur(PL):1
    s-btr∘cur(PL):Q
    s-dn∘cur(PL):μ(DN;{<elem(s-off∘s-id∘elem(i,s-dp(ST))):elem(i)∘s-dp∘Q> |
                                  is-p-proc-atr(s-atr(elem(i,s-dp(ST))))} ∪
                 {<elem(s-off∘s-id∘elem(i,s-dp(ST))):μ_0(<s-btr:Q>,<s-dec:i>)> |
                                  is-p-lab-atr(s-atr(elem(i,s-dp(ST))))} ∪...)

 inst-proc(arg-1,s,epa,parm) =

    s-pl:PL⌒ [μ_0(<s-epa:epa>,
                 <s-ptr:s>,
                 <s-btr:I>,
                 <s-dn:μ_0({<elem(i):elem(i,arg-1)> | 1 ≤ i ≤ length(arg-1)})>)]
    s-disp:(parm-inf ⟶ upd-disp(PL,s-dep∘s(TX) - 1,epa,DISP)⌒ [length(PL) + 1]
            T ⟶ μ(DISP;<elem(s-dep∘s(TX)):length(PL) + 1>))

 reset-btr = s-btr∘cur(PL):(ια)((∃ i)(BTR = elem(i)∘s-sp∘α))
             s-ctr∘cur(PL):(ιi)((∃α)(BTR = elem(i)∘s-sp∘α))

 step-ctr = s-ctr∘cur(PL):CTR + 1

 set-btr = s-btr∘cur(PL):s-btr∘s-dcl(ABN)
           s-ctr∘cur(PL):s-atr(elem(s-dec∘s-dcl(ABN),
                                 s-dp∘(s-btr∘s-dcl(ABN))∘PTR(TX)))

 epilogue = cur(PL):Ω
            s-disp:upd-disp(PL,s-dep∘s-ptr∘elem(length(PL) - 1,PL)(TX),
                                               length(PL) - 1,DISP)

 goto(p) = s-bl:front(p,PL)
           s-disp:upd-disp(PL,s-dep∘s-ptr∘elem(p,PL)(TX),p,DISP)
```

```
upd-disp(pl,dep,p,disp) =
   dep ≥ 1 ──→ upd-disp(pl,dep-1,s-epa∘elem(p,pl),μ(DISP;<elem(dep):p>))
   T ──→ disp

dd-p(ref,pl,disp) = s-p(dd(ref,pl,disp))

dd-d(ref,pl,disp) = s-dcl(dd(ref,pl,disp))

dd(ref,pl,disp) = is-parm-den(elem(s-off(ref),s-dn(elem(ind,pl)))) ──→
                                       elem(s-off(ref),s-dn(elem(ind,pl)))
                  T ──→ μ₀(<s-p:ind>,<s-dcl:elem(s-off(ref),s-dn(elem(ind,pl)))>)

where ind = elem(s-dep(ref),disp)
```

APPENDIX II:

This appendix contains an iterative model of the implementation described in section 7.

Properties of the model

Axiom: This axiom states that if in the text a procedure can be recursively used then all blocks embraced by it can be recursively used

$$\text{is-rec}(\alpha(t)) \supset (\text{is-block} \lor \text{is-proc-atr})(\beta\circ\alpha(t)) \supset \text{is-rec}(\beta\circ\alpha(t))$$

Axiom: This axiom states that only blocks allowed to be recursive can be used recursively.

$$1 \leq i < k \leq \text{length}(BL) \supset BN_i = BN_k \supset \text{is-rec}(PTR(TX))$$

State of the DF1-model

```
is-state = (<s-tx:is-p-block>,
            <s-bl:is-bl-list>,
            <s-disp:({<bn:is-int> || is-bn(bn)})>,
            <s-c:is-c>)

is-bl = (<s-ptr:is-sel>,
         <s-ctr:is-int ∨ is-Ω>,
         <s-f:is-int ∨ is-Ω>,
         <s-df:is-int ∨ is-Ω>,
         <s-dn:({<id:is-den> || is-id(id)})>)

is-den = is-proc-den ∨ is-lab-den ∨ is-parm-den ∨ is-other-den

is-proc-den = is-int

is-lab-den = is-int

is-other-den = ...

is-parm-den = (<s-b:is-int>,<s-dcl:is-int>,<s-f:is-int ∨ is-Ω>)

is-p-block  as the definition FD8 except that blocks and procedures have a mark
            indicating whether the block can be used recursively or not
            (is-rec(t) is true or not).
```

Abbreviations used:

```
BL = s-bl(ξ)
TX = s-tx(ξ)
DISP = s-disp(ξ)
B = last(BL)
PTR = s-ptr(B)
CTR = s-ctr(B)
F = s-f(B)
DN = s-dn(B)
BN = s-bn(B)
P = CTR ≠ Ω ——→ elem(CTR)∘s-sp∘PTR
    T ——→ s-body∘PTR
ST = P(TX)
DF = s-df(B)
cur(bl) = elem(length(bl))∘s-bl
```

Initial state:

```
ξ_o = μ_o(<s-tx:μ_o(<s-sp:[t_o]>,
                    <s-dp:[]>)>,
        <s-bl:[ μ_o(<s-ptr:I>,
                    <s-ctr:1>)]>,
        <s-c:int-st>)
```

where t_o again satisfies 6-1, 6-2 and the above two axioms.

State transition function

```
int-st =
   is-p-block(ST) ——→ int-st;
                        inst-bl
   is-p-proc(ST) ——→ int-st;
                       inst-bl;      length(s-op(ST))
                         inst-proc(  LIST   m-p(df(elem(i,s-ap(ST)),BL,DISP),BL,TX),
                                     i=1
                                 s-atr∘elem(d)∘s-dp∘s-ptr∘elem(b,BL),f)
   is-p-goto(ST) ——→ inst-st;
                       set-ctr(df-d(ST,BL,DISP));
                         goto(df-b(ST,BL,DISP))
                  ⋮
```

cont'd

```
   length(BL) > 1 ⟶ int-st;
                     step-ctr;
                       epilogue
   T ⟶ null

where: b = df-b(s-nm(ST),BL,DISP)
       d = df-d(s-nm(ST),BL,DISP)
       f = df-f(s-nm(ST),BL,DISP)

inst-bl =
   s-bl:BL⌢[$\mu_0$(<s-ptr:P>,
              <s-bn:s-bn∘P(TX)>,
              <s-df:(is-rec∘s(TX) ⟶ s-bn∘P(TX)(DISP)
                     T ⟶ Ω)>,
              <s-ctr:1>,
              <s-dn:$\mu_0$({<s-id∘elem(i,s-dp(ST):i> |
                              ¬is-p-other-atr(s-atr∘elem(i,s-dp(ST))}>)]
   s-disp: (DISP;<s-bn∘P(TX):length(BL) + 1>)

inst-proc(arg-l,s,f) =
   s-bl:BL⌢[$\mu_0$(<s-f:f>,
              <s-ptr:s>,
              <s-bn:s-bn∘s(TX)>,
              <s-df:(is-rec∘s(TX) ⟶ s-bn∘p(TX)(DISP),
                     T ⟶ Ω )>,
              <s-dn:$\mu_0$({<elem(i,parm-l):elem(i,arg-l)> |
                              1 ≤ i ≤ length(arg-l ∧ parm-l = s-pp∘s(TX)}>)]
   s-disp:μ((f ≠ Ω ⟶ upd-disp1(f,BL,DISP),
             T ⟶ DISP);<s-bn∘s(TX):length(BL) + 1>)

step-ctr = s-ctr∘cur(BL):CTR + 1

set-ctr(d) = s-ctr∘cur(BL):s-atr∘elem(d)∘s-dp(PTR(TX))

epilogue = cur(BL): Ω
           s-disp:(F ≠ Ω ⟶ upd-disp1(length(BL) - 1,BL,DISP),
                   is-rec∘PTR(TX) ⟶ μ(DISP:<BN:DF>),
                   T ⟶ DISP)

goto(b) = s-bl:front(b,BL)
          s-disp:upd-disp1(b,BL,DISP)
```

```
adr(id,b,bl) = is-parm-den∘id∘s-dn∘elem(b,bl) ⟶ id∘s-dn∘elem(b,bl),
                   T ⟶ μ(<s-b:b>,<s-dcl:id∘s-dn∘elem(b,bl)>)

m-p(den,bl,t) = s-f(den) ≠ Ω  v ¬ is-proc-atr∘elem(s-dcl(den)∘s-dp∘s-ptr∘
                                           elem(s-b(den),bl)(t) ⟶ den,
                   T ⟶ μ(den;<s-f:length(bl)>)

upd-disp1(k,bl,disp) = k ≤ 1 ⟶ disp
                       s-f∘elem(k,bl) ≠ Ω ⟶ μ(upd-disp1(s-f∘elem(k,bl),bl,disp);
                                             <s-bn∘elem(k,bl):k>)
                       T ⟶ μ(upd-disp1(k-1,bl,disp);<s-bn∘elem(k,bl):k>)

df-b(ref,bl,disp) = s-b(df(ref,bl,disp))

df-d(ref,bl,disp) = s-dcl(df(ref,bl,disp))

df-f(ref,bl,disp) = s-f(df(ref,bl,disp))

df(ref,bl,disp) = adr(s-id(ref),s-bn(ref)(disp),bl)
```

APPENDIX III:

This appendix indicates the meaning of notations and terms used in the current report. For the most part this consists of referencing /2/ (the section numbers of which are distinguished by the symbol §). Notation peculiar to this report is defined in lines marked with an asterisk.

§ 1 - computation
abstract syntax
$\circ$ - functional composition
* $\forall \exists$ quantifiers - whose bound variables indicate their domain as follows:
$i, j \ldots n$ = is-int
$\alpha, \beta, \gamma, \varkappa$ = is-sel
other = is-other
ι descriptors
* $\underset{i}{\text{MAX}}\ p(i) \underset{\text{Df}}{=} (\iota i)(p(i) \wedge \neg(\exists j)(j > i \wedge p(j)))$
* $\underset{i}{\text{MIN}}$ similar

§ 2.1, § 2.2 - objects
selectors

§ 2.3, § 2.4 - μ-operator [1)]
I

§ 2.5 Lists - elem(i)
elem(i,x)
* $[x_1,\ldots,x_n] \underset{\text{Df}}{=} \mu_o(\langle elem(1):x_1\rangle, \ldots, \langle elem(n):x_n\rangle)$
length
$\frown$
$\underset{i=1}{\overset{n}{\text{LIST}}}$
* last(list) = elem(length(list),list)
* front(n,list) = $\underset{i=1}{\overset{n}{\text{LIST}}}$ elem(i,list)

§ 2.6 Predicates - $(pe_1 \vee pe_2 \vee \ldots \vee pe_n)(x)$
$(\langle se_1:pe_1\rangle, \langle se_2:pe_2\rangle, \ldots \langle se_n:pe_n\rangle)$
$(\ \langle se:pe\rangle \ ||\ prop\ )$
pr-list

1) The properties of the μ-operator are used throughout this report without reference.

§ 2.7 Assumptions on EO and S -

$\text{is-id}(s) \underset{Df}{=} (s \in S_{id})$

* $\text{is-un}(s) \underset{Df}{=} (s \in S_n)$

* $S_{id} \cap S_n = \{\}$

* is-int defines the set of integers

* is-sel defines the set of selectors

§ 6.3, § 6.4 Instructions

* card(set) number of elements in set

* $SC^i \underset{Df}{=} \text{s-sc} \circ \text{s}(\xi^i)$ where SC is the abbreviation of $\text{s-sc} \circ \text{s}(\xi)$

* $SC_i \underset{Df}{=} \text{s-sc} \circ \text{elem}(i) \circ \text{s}(\xi)$ where SC is the abbreviation of s-sc(last(SL)) and SL is the abbreviation of $\text{s}(\xi)$

APPENDIX IV:

This appendix lists the main abbreviations used in the report. Whilst no meaning is drawn from the names used, the offered text may be suggestive for the reader. Each term is listed in lower case letters but may be used in either case.

abn	abnormal termination
adr	address
ap	argument part
arg	argument
atr	attributes
b	block
bl	block local element
blk	block
bn	block name
btr	counter (of statements)
c	control part
cnt	count
cont	containing
ctr	counter (of statements)
cur	current
dcl	declaration index
dec	declaration
den	denotation
dep	depth
dir	directly
disp	display
dn	denotation component
dp	declaration part
epa	environmentally preceding activation
F	PL/I F-level compiler
id	identifier
inst	install
int	integer - syntax
int	interpret - instruction name
intr	introduced

lab	label
len	length
m	modified
mod	modify
m-p	modify parameter
nm	name
off	offset
p	procedure index
P	pointer to a statement
parmd	parameter introduced
pl	procedure local element
pp	parameter part
proc	procedure
ptr	pointer (to a block)
Q	pointer to a statement
red	redeclared
ref	reference
reflb	referencable
rev	revise
s	selector
sp	statement part
st	statement
t	text
tx	text
un	unique name
upd	update