A SIMPLE MECHANISM MODELLING SOME FEATURES OF ALGOL60

E. W. Dijkstra

The purpose of this note is to describe an abstract machine, which has been designed primarily for explanatory purposes. As such a machine responds in a unique way to any given input character sequence one may hope that such a machine may serve a useful purpose as a tool for language definition. This does not pretend to provide an adequate tool for the definition of an arbitrary programming language. The presentation offers a clarification of some of the mechanisms essential for the understanding of ALGOL60. These are evaluation of an expression, assignment to a variable, the block structure, the possibly recursive call of a procedure and the parameter mechanism. It does not deal with labels and arrays.

In the following operators, constants, and what we usually call identifiers are all treated as single characters, individually recognizable. Inside the machine distinguishable characters of a further category will be generated; they may not occur in an input text and we call them "references". There will be a special reference, called the "dummy reference".

The machine can be visualised as containing three stacks. The first one is called "the working stack", the second is called "the correspondence stack". The third one is only used to control the sequencing of the character reading process, since this is defined recursively. We will not refer to it any further.

The basic cycle of the machine consists of "reading the next character", i.e. passing it by and reacting to it. This holds whether the character is read from the input text or internally. The machine begins by reading the first character of the input text, its three stacks being empty.

In addition to the stacks the machine has a store, in which may be held finite sequences of characters. A reference gives access to the first character of a stored sequence. The remaining characters of such a sequence can only be read in succession, just as is assumed for the input text.

For simplicity we shall deal only with the case that in the input text no identifier is declared more than once. We shall moreover assume that at the beginning of each block just one identifier is declared. Evidently this entails no loss of generality.

The machine is at any moment in one of two states, called "the active state" and "the passive state" respectively. The machine begins in the active state.

We shall now define the machine by describing its reaction to the various characters; first we shall do so for the active state.

[This acts as a "block begin", and also the next character will be read. If the next character is not an identifier, the machine gives a failure indication (and stops). In the case that the next character is indeed an identifier a new reference, unequal to the dummy reference, is generated and a new element is put on top of the correspondence stack. This element consists of the identifier paired with this reference; furthermore a single character sequence is generated, consisting of the single character "undetermined". Until further notice the reference will give access to this single character sequence.

] This acts as a "block end". If the correspondence stack is not empty, its top element will be removed; furthermore the sequence to which the reference in question refers is replaced by the character "undetermined". If the correspondence stack is already empty, or if the element removed contained the dummy reference, failure indication is given.

undetermined Failure indication is given.

a reference Read actively the sequence of characters referred to by the reference up to the special character "T", then return to the character next to the reference in question. (The special character "T" - apart from underlining I shall use capitals to denote special characters - acts as an end marker for linear sequences. All sequences in store will be terminated by the character "T", apart from a pathological one like "undetermined", where this is pointless.)

an identifier (not following "T") The correspondence stack is searched, starting at the top side, for the occurrence of the identifier in the elements. This search ends as soon as the identifier has been found, otherwise because the correspondence stack has been exhausted. In the latter case the failure indication is given, in the first case the machine continues as if it had read the reference passed with occurrence of the identifier detected in the correspondence stack.

Note 1 The distinction between "identifier" and "reference" is necessary because even without the "redeclaration" excluded, an ALGOL60 identifier does not identify a unique object: the local variables of a procedure, which is activated recursively, exist in multiplo during inner activations and an additional rule must specify to which copy the identifier refers. The correspondence stack and the way in which it is used, supplies such an additional rule.

a constant Constants are put on top of the working stack.

an operator — The operator is performed. Usually this implies some operations on the top of the working stack. Some examples of possible unary operators are:

- "neg" — the unary minus sign: if the top character of the working stack is a numerical character, its sign is inverted, otherwise a failure indication is given.
- "non" — the logical negation: if the top character of the working stack is a logical value, it is replaced by its inverse, otherwise failure.
- "sqrt" — if the top character of the working stack is a non-negative number, it is replaced by its square root, otherwise a failure indication is given.

Examples of possible binary operations are:

- "-" — if the top two characters of the working stack are numerical, the top character is subtracted from the top character but one. The answer replaces the top character but one, the top character itself is removed from the working stack. If not both top elements are numerical, a failure indication is given.
- "=" — Numerical equality. If the top character of the working stack and the top character but one are not both numerical, a failure indication is given. Otherwise they are removed; if the two characters removed are equal, the character "true" will subsequently be put on top of the stack, otherwise the character "false".

etc., etc..

{ — Put "(" on top of the stack.

} — Put ")" on top of the stack.

S — Put "T" on top of the stack.

Note 2 — The above three rules are only a trick. Reading a character "(" or "T" in a text will have other effects, so I need other means if I want to generate them on top of the working stack.

sel — The selection operator is a ternary operator. If the top character of the working stack is not a logical value, a failure indication is given. If the top character of the working stack is true, the two top characters of the working stack are removed from it. If the top character of the working stack is false, the three top characters are replaced by the middle one.

Thus the execution of the operator sel transforms the working stack top

"... a b c true" into "... a b"

whereas

"... a b c false" is transformed into "... a c"

:= The removing sequence assignment. If the top character of the working stack is not a reference, failure indication is given. Otherwise, the stack is scanned downwards until the first terminating character T is encountered. The characters passed are taken to be the new sequence to which the reference in question will refer until further notice (i.e. a new assignment or removal of the reference from the correspondence stack.). The new sequence "assigned to the reference" will be terminated by a T at the other end, the previous sequence referred to by the reference is deleted, and the top characters of the working stack are deleted "down to and including" the terminal T in the working stack. This may be clarified by the following example, giving the state of the top of the working stack and the referencing before and after the execution of the removing sequence assignment.

Before: "... a b c T d e f" and f -> "p q r T"

After: "... a b c" and f -> "d e T"

under the assumptions that "f" is a reference and d and e are both unequal to "T".

:- The retaining sequence assignment. As above, but for the fact that only the top character of the working stack is removed. Under the same initial conditions as above, the execution of the ":-" will give the final situation

"... a b c T d e" and f -> "d e T"

:= The removing character assignment. Here the length of the new sequence to be referred to by the reference is by definition a two character sequence, closing T included. The two top characters of the working stack are removed. Under the same assumptions as above, the ":=" creates

"... a b c T d" and f -> "e T"

:- The retaining character assignment. As the previous operation but for the fact that only the top character will be removed from the working stack. Under the same assumptions the execution of ":-" creates

"... a b c T d e" and f -> "e T"

T Reading will be resumed at the character following thethe occurrence of the reference (or paired identifier) on account of the reading of which the now terminating reading of the sequence in question has been initiated.

) Closing passivity bracket. When read while in the active state, a failure indication is given.

(Opening passivity bracket. When read while in the active state a "passivity bracket counter" is set to one and the transition to the passive state takes place. The stacks remain unaltered.

We shall now describe the actions of the machine in the passive state.

All characters except "(", ")", "T", "[", "]" are put unaltered on top of the working stack.

T If "T" is read in the passive state a failure indication is given.

[If "[" is read the next character is also read. If this is not an identifier, a failure indication is given, otherwise a next element is added to the correspondence stack, consisting of the identifier in question paired with the dummy reference.

] If the correspondence stack is not empty and the top element pairs an identifier with the dummy reference then the top element of the correspondence stack will be removed. Otherwise a failure indication will be given.

identifier (not following [ ) If an identifier is read in the passive state the stack of correspondences is searched as in the active state. If it is not found, a failure indication is given. If it is found, paired with a true reference, this reference will be put on top of the working stack; if it is found paired with the dummy reference, the identifier itself is put on top of the stack.

(The "passivity bracket counter" will be increased by one and the opening passivity bracket just read will be put on top of the working stack.

) The "passivity bracket counter" will be decreased by one. If the result is positive, the closing passivity bracket will be copied on top of the working stack, otherwise the working stack will remain unchanged and the transition to the active state takes place.

Note 3 As can be seen from the above description the passivity brackets show enough resemblance to string quotes - reading implies

the removal of the outer pair! - that it is worthwhile to point out the difference. During passive reading the identifier-reference replacement will take place whenever the identifier is found in the correspondence stack, but not paired with the dummy reference. This mechanism is used for two purposes.

When an ALGOL60 block is entered, in which a procedure is declared, this procedure may refer to quantities global and local to its body. It is intended to make at this moment a new copy of the procedure body in which

(a) all global identifiers are replaced by the correct referen ces in so far as this has not already happened,

(b) all local identifiers remain unchanged, which is achieved by the trick of the dummy reference. This reflects the attitude that the moment to introduce references for its local quantities will come whenever it is activated, i.e. whenever its character sequence will be read in the active state.

The other purpose is closely resembling: it enables us to specify actual parameters just prior to entry of the procedure body, but ehsuring that the identifier-reference replacement is performed in accordance with the situation at call side.

Note 4 Characters in the working stack are never "read" in the sense in which we used the word reading. The only characters which can be read in this sense are those of the input sequence and in sequences referred to by a reference. Sequences of such "readable" characters we have called text. The working stack does not contain text, it contains material out of which, finally, an assignment operation can construct a piece of readable text.

Note 5 When adding a new element with a non-dummy reference to the correspondence stack we said a "new" reference is generated. Intended is a reference not used before. When an element with a non-dummy reference was removed from the correspondence stack we took pains to ensure that from now on it would point to the single character "undetermined". We had to take this precaution because we have not excluded a priori that the sequence referenced by "a" contains as element the reference "b", where reference "b" will be removed earlier than reference "a".
(In ALGOL60 we have variables with a limited life-time: if they are normal variables, their values must be constants, the life-time of which embraces the life-time of the whole program. Formal parameters will be treated as variables, the value of which - viz. the corresponding actual parameter - may be built up from mortal components. But the structure of ALGOL60 ensures that the life-time of these components embraces that of the formal variable. Therefore the situation noted will not occur in ALGOL60.)

Note 6 In the above we have spoken about "the deletion from store of sequences of text", viz. when a reference will refer to a new sequence. We have to envisage the situation, however, that the third stack points to a character in this sequence, and that therefore the machine will try to continue reading it at some later moment. The most elegant solution is to keep the sequence in store anonymously as long as the third stack contains returns to it. A "usage counter" per sequence in store will do the job.

In the following examples small letters will be used as identifiers; a dash will be added - as long as no confusion arises - to denote the reference associated with it. Only under exceptional circumstances will we use the character assignments.

Example 1

Let us consider the following ALGOL60 program.

"begin integer x; x := 5; x := x + 2 end"

This will be represented as follows:

[x S 5 (x) := S x 2 + (x) :=]

We give the stack contents at successive characters.

1 and 2)	Establishment of the correspondence x - x'	x' → undetermined
3)	T	
4)	T 5	
5)	transition to passive state	
6)	T 5 x'	
7)	transition to active state	
8)		x' → 5 T
9)	T	
10)	T 5	
11)	T 5 2	
12)	T 7	
13)	transition to passive state	
14)	T 7 x'	
15)	transition to active state	
16)		x' → 7 T
17)	removal of correspondence x - x'	x' → undetermined

Example 2

A type procedure with value actual parameter.

```
"begin integer y;
      integer procedure sq(u); value u; integer u;
            sq := u × u;
      y := sq(3)
 end"
```

The top of the stack will be used to transmit the actual parameter to the procedure and to receive the result. Our text will be (with two superfluous passivity brackets):

```
[ y [ sq
S ( [ u ( u ) := u u × ] ) ( sq ) :=
S S 3 sq ( y ) := ] ]
```

The first line introduces two new references, the second line is the equivalent of the procedure declaration, the last line represents the function call and assignment, followed by two block ends. Explanatory pictures are:

	y - y'	y' → undetermined
	sq - sq'	sq' → undetermined
T [u	u - dummy	
T [u (u) := u u ×] sq'		
stack empty but:		sq' → [u (u) := u u ×] T
T T 3	u - u'	
T T 3 u'		
		u' → 3 T
T 3 3		
T 9	removal u - u'	u' → undetermined
T 9 y'		
stack empty		y' → 9 T

Example 3

In future we shall need the operator E which is applicable when the top character of the working stack is a reference, which removes this reference, and reads the sequence it refers to before going on. It amounts to reading the top character of the working stack in the active state after removing it.

We can accomplish this if we have in store the reference

E' -> [f (f) := f] T

From now on we shall regard E as one of our primitives, i.e. each program should be viewed as embedded in between

" [E S ([f (f) := f] E) := " and "] "

Example 4

In ALGOL60:

```
"begin integer b;
      procedure nega(u); integer u; u := -u;
      b := 3;
      nega(b)
 end "
```

Our text will be as follows (to be embedded as described above):

[b [nega

S ([u (u) := S u E neg u :=] nega) :=

S 3 (b) :=

S ((b)) nega]]

After the reading of the second line we have created:

nega' -> [u (u) := S u E' neg u :=] T

and the procedure declaration has been processed.
The next line creates:

b' -> 3 T

The last line starts by putting on the working stack:

T (b')

and then the text referenced by nega' will be read, giving:

T (b') u'

empty u' -> (b') T

T b'

T 3

T -3 b'

empty b' -> -3 T

Example 5

The conditional action: We consider what could be loosely written as

if B then "A1" else "A2"

"A1" and "A2" here represent alternative pieces of text. B represents a Boolean expression which according to the usual rules generates true or false on top of the stack. In the case that both A1 and A2 have matching activity brackets we can express this by means of

B S ("A1") S ("A2") if

provided the reference if' points to

if' -> [b [a1 [a2 (a2) := (a1) := (b) := (a1 a2) b sel E]]] T

In the above call it will create the relations

a2' -> "A2" T a1' -> "A1" T

Example 6

Partial evaluation of an actual parameter. The well known example, of course, is that in the case of an ALGOL60 call like "ncga(A["E1"])" we cannot prevent the expression "E1" from being evaluated once for each occurrence of the formal parameter. One would like to express that the value of E1 at the moment of call should be substituted in the actual parameter. As I do not deal with subscription at present I shall show how to transmit the actual parameter derived from "a + b" but where b should be replaced by its value at the moment of call

At call side we write:

S (a) b (+)

which generates on top of the stack, say,

T a' 7 +

The consuming procedure may start with

[u (u) := .....

generating the initialisation of the formal parameter in the form

u' -> a' 7 + T

Eindhoven,
April 1964.