THE IMPORTANCE OF IMPLEMENTATION MODELS IN ALGOL 68, or HOW TO DISCOVER THE CONCEPT OF NECESSARY ENVIRONMENT

*Daniel M. Berry, General Electric Company, P.O. Box 8, Schenectady, New York 12301.**

Abstract *The need for implementation models in understanding languages, Algol 68 in particular, is stressed. The model is used to demonstrate the new concept of necessary environment of a procedure.*

Keywords *Algol 68, implementation model, pushdown stack, static link, environment, environment pointer, scope, procedures, pointers.*

CR Categories: *4.12, 4.2, 4.20, 4.22*

Traditionally computer languages have been specified independently of any implementation concerns. The effect of this divorce was felt even when the Algol 60 report [5] came out. A clear understanding of blocks, procedures, and the non-local and parameter mechanisms was best obtained from a conceptual model for its run time dynamics. The model used was that of a stack with activation records and static links [9]. The conceptual model did not have to be a full blown one with the exact mechanism for maintaining the stack, but rather just the notion of last-in first-out order of activation and deactivation of records and the static link sufficed for understanding Algol 60. The divorce of language specification from its implementation has reached its zenith with the Algol 68 report [8]. The report is quite elegant, but no real insight into the structure of Algol 68 is going to come unless there is use of a model.

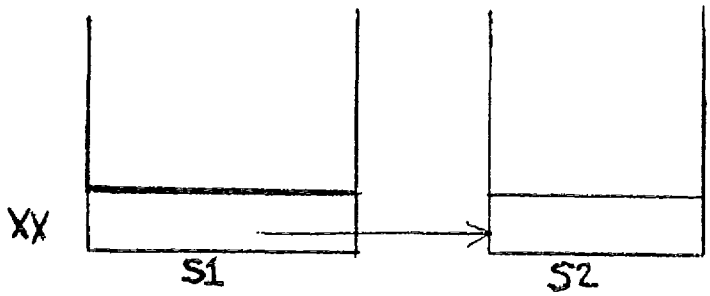
The use of such conceptual models of run time configurations arising from execution of a program can give much insight into the necessity of certain features and restrictions in the language. One example is the scope restrictions on assignations in Algol 68. The rules state that the scope of the right hand side of an assignation must be greater than or equal to the scope of the left hand side. Because of these rules, the assignation in line 5 of the following program is illegal.

**Present address: Department of Applied Mathematics, Brown University.*

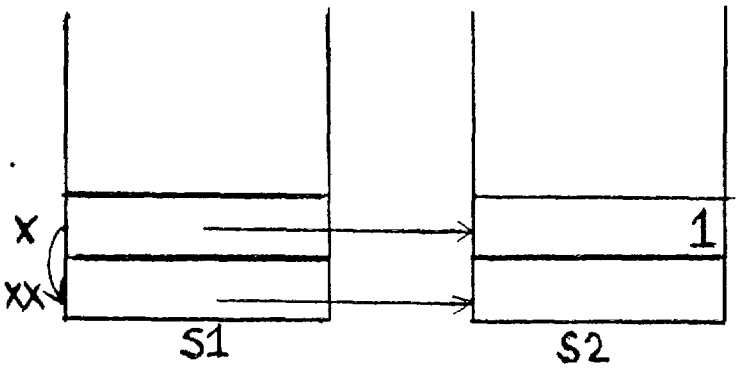
```
1   begin
2     ref ref int xx  = loc ref int;
3     begin
4       ref int x = loc int : = 1 ;
5       xx  : = x ;
6     end
7     print (xx);
8   end
```

The scope of the left hand side, xx, is the outer range, the scope of the right hand side, x, is the inner range, and is thus less than that of the left hand side. Clearly the scope rule for assignations is violated. The true nature of this violation and the catastrophic result of not obeying this rule is clearer if one considers a run time implementation model for Algol 68. The model used has two stacks, S1 and S2 [3, 10] S1 holds activation records containing cells for values possessed by identifiers and containing a static link. S2 hold locally generated cells.

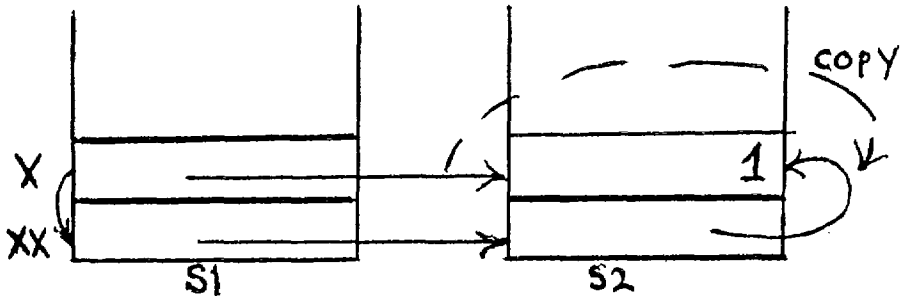
First the outer range is entered and the declaration in line 2 is elaborated. On S1 is placed an activation record with a cell for xx and a null static link. S2 has a cell for the reference of xx.



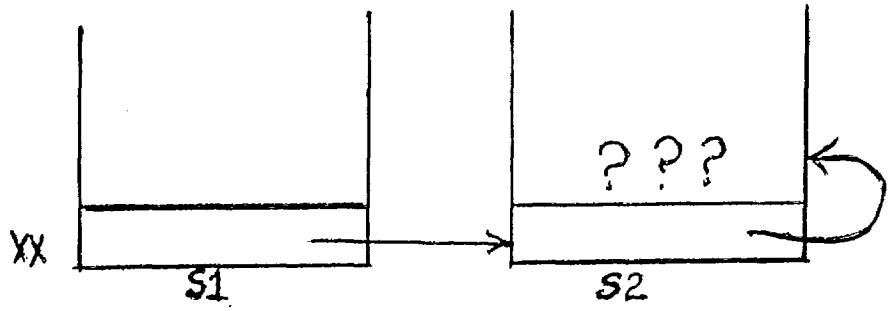
Then in line 3, a new range is entered and in line 4, x is declared, resulting in an activation record with a cell for x and a static link to the previous record on S1. S2 gets a cell for the reference of x and this cell is initialized to 1.



Now suppose the illegal assignation in line 5 were allowed. Then the reference of xx gets a copy of the value possessed by x.



When the range is ended, the top activation record is popped from S1 and its reference in S2 is deleted.



The pointer in the reference of xx now points to a non-existent cell. Any value accessing use of xx, such as the write (xx) in line 7, will have problems digging up an integer value. Thus, we see from the model, the necessity of the scope rule for assignation. It is clear that the scope rule prevents upwards pointers in the stack which might provide access to deallocated cells.

This paper has two main points. The first has been demonstrated and will continue to be demonstrated; it is that the use of the model can convey new insight into computational processes and language design. One such insight is recognition of a new concept, that of the necessary environment of a procedure, the introduction of which is the second main point of this paper. This concept was discovered by use of the model and trying to reconcile the Report's scope rules as applied to procedures with the implementation features of environment and environment pointers of procedure values. First we show the run time configuration of the program that lead to the discovery and the development of the concept. This concept has several implications for implementation of Algol 68 which are briefly listed. Finally, the application of this concept to other languages is briefly explored.

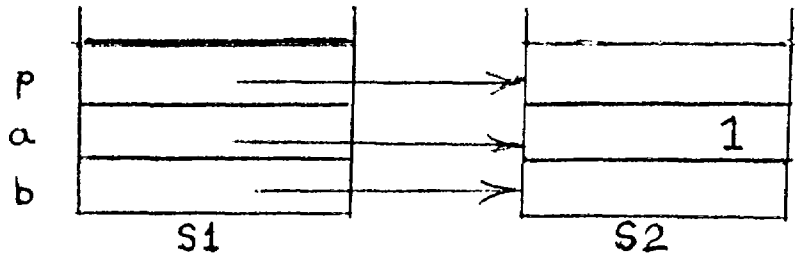
Let us consider the following brief Algol 68 program.

```
1   ┌   begin
2   |     ref int b = loc int,
3   |     ref int a = loc int,
4   |     ref proc (int) int p = loc proc (int) int;
5   |     a: = 1;
```

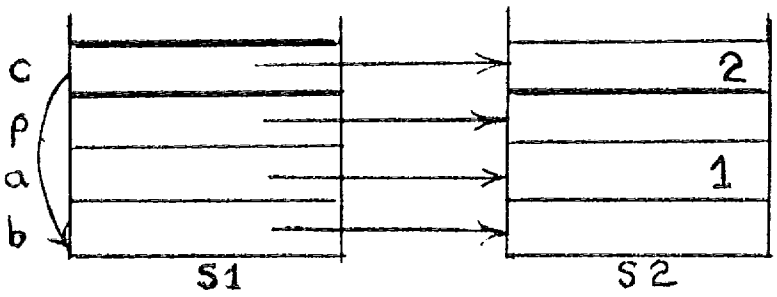
```
6        begin
7           ref int c = loc int: = 2
8           p: = ((int i) int: i + a);
9        end
10     b: = p(3);
11  end
```

The Algol 68 report defines the scope of a procedure value to be the smallest range containing a declaration of any non-local identifier in the denotation possessing the procedure value. In the program above the scope of the denotation in line 8 is the outer range because it has the declaration of the non-local a. Remember that Algol 68 has a rule which says that an assignation is legal only if the scope of the right hand side is greater than or equal to the scope of the left hand side. In line 8, the left hand side, p, has a scope of the outer range. Hence the assignation is legal. Observe that if the non-local in the denotation were c instead of a, the assignation would be illegal.

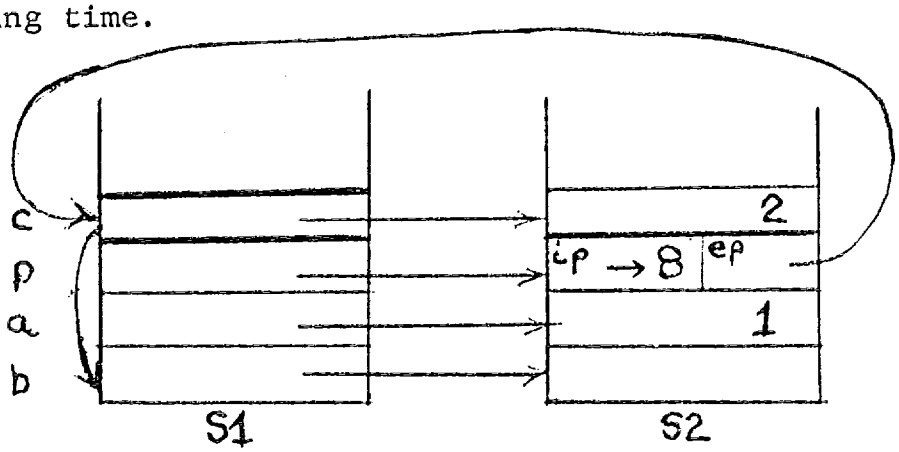
Now let us follow the run time configuration for elaboration of the program. After executing the declarations of the outer range in lines 1-4 and the assignation in line 5, we have an activation record on S1 and three cells on S2.



Then at lines 6-7 a new range is entered with the subsequent creation of an activation record on S1 and a cell on S2. The static link of this new activation record is shown on the left side of S1.

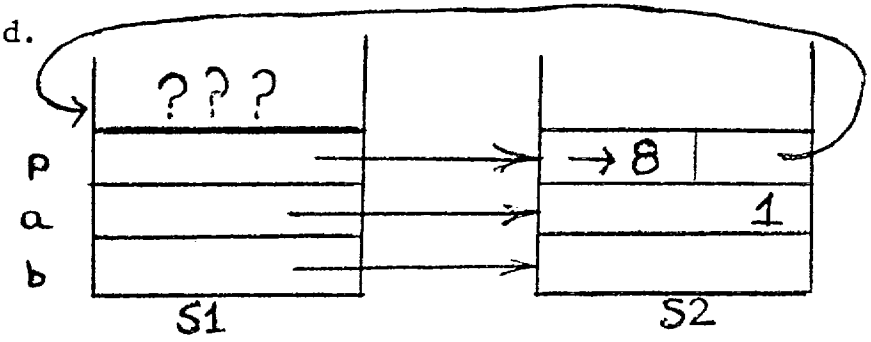


In line 8, a procedure value is computed. This value consists of an instruction pointer, ip, and an environment pointer, ep. The ip of the computed value points to the beginning of the text of the procedure in line 8. Traditionally*, one takes as the ep a pointer to the top activation record of the current environment, that is, to the top record of S1 (The current environment is the top record and all records linked to the top by a chain of static links). This value is then assigned to the reference of p . This process of creating and assigning a procedure value is called binding, and the environment traditionally used is the current environment at binding time.



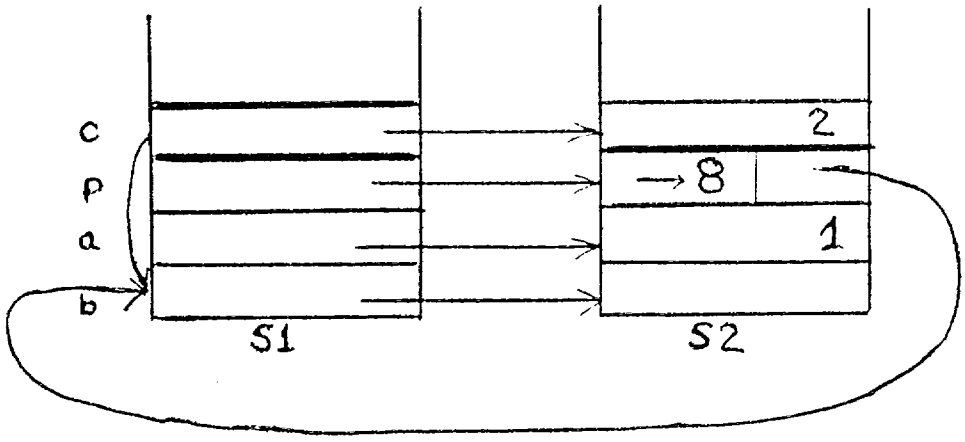
* The word "traditionally" merely indicates that the mentioned scheme for computing the ep of a procedure has been widely used with Algol 60 à la Randell and Russell [12] and with PL/1 a la the ULD formal definition [13].

Now at line 9, the top record is deallocated as the inner range is ended.

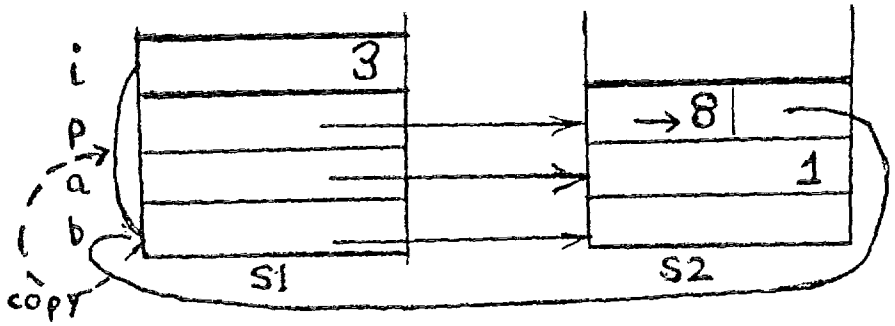


Observe that the ep of the procedure is now dangling; its reference has been deallocated. This is indeed a disaster because when the activation record for the call in line 10 is created its static link is to be a copy of the ep. There will be no activation record pointed to by this static link, making it difficult to access non-locals via the static chain.

The scope rules were apparently designed to prevent this disaster resulting from pointing too far up the stack. To regain correctness of implementation a modification in the definition of the ep of a procedure value is needed. The ep constructed at binding time will point to the top most activation record of the current environment in which any non-local of the procedure is declared. We call this record and those records of the current environment accessible from this record by the static chain the necessary environment of the procedure. In the above example, the ep would point to the bottom activation record. We show a modified diagram after elaboration of line 8.



This way the inner range can be deallocated and the call in line 10 may be elaborated. The activation record for the call has a static link pointing to the bottom record.



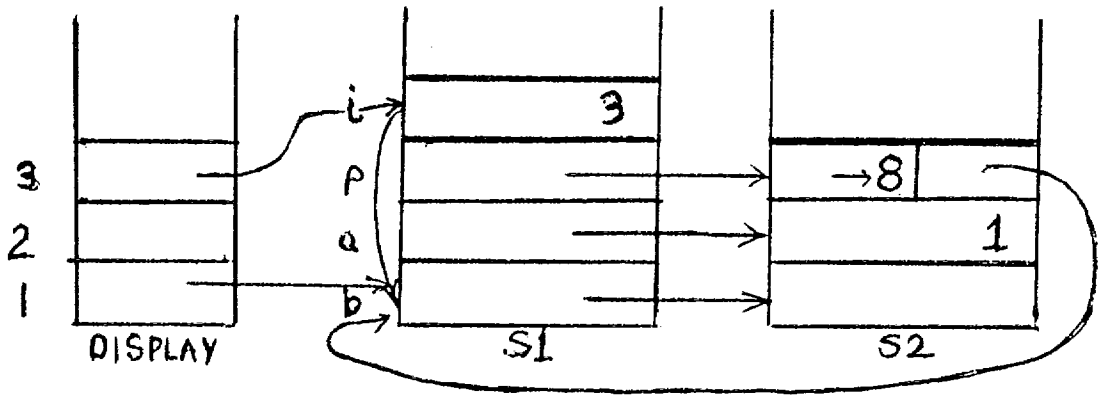
The new rule for determining the ep of a procedure value when elaborating a procedure denotation has several implications:

1) The ep points to an activation record of the range which constitutes the scope of the procedure value.

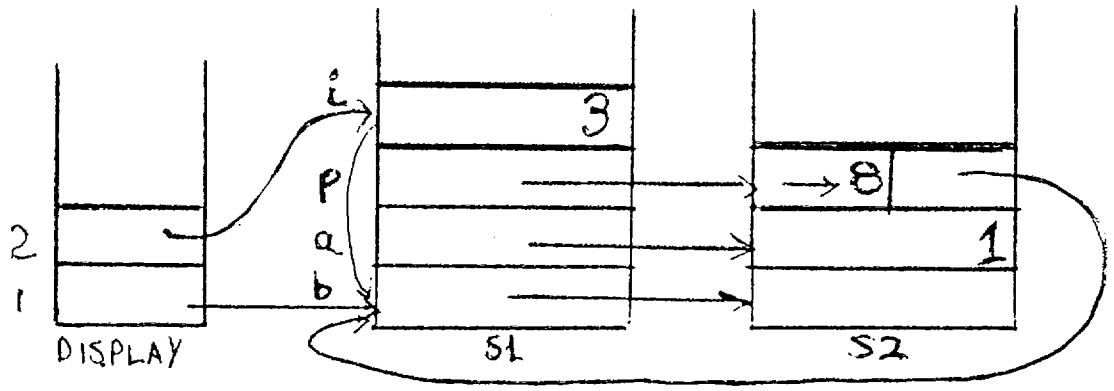
2) A procedure with no non-locals has a null ep.

3) Elaboration of a routine denotation will produce the same value no matter where in its scope the denotation is written.

4) There are at least two compiler/run time techniques which can be used to implement procedure values and calls with the necessary environment. At compile time, <i,j> pairs, that is <nesting height, relative position in record> pairs, are created for individual identifiers. At run time the ep and static links are used to maintain a display which provides immediate access to an activation record of a given nesting height. One technique is to use the normal nesting height conventions in compilation but to have a display with holes in it[14] In the example, the routine denotation has a nesting height of three (note brackets). So, the local i in the denotation would be compiled as <3,1>. However at call time, there is no activation record for a range of nesting height two because that range has been exited, and the ep of the procedure points to a record of height one.



This technique makes compiling relatively easy, but it forces a little extra work at run time. The second method retains the normal display management at the expense of the compiler. In compiling <i,j> pairs routine denotations are compiled to have a nesting height of one more than that of the range containing the declaration of the innermost non-local. In the example, the denotation would have nesting height two and the local i would be compiled as <2,1>. In this way, the static link arising from the ep can be used to construct a hole-free display stack.



This new concept of necessary environment may be applied to other languages which have procedure values treated as general values, and in which the non-locals are bound as in Algol 60 or 68. In

some of these languages such as GEDANKEN [11] and OREGANO, which this author is developing, the non-locals are kept as long as the procedure value still exists. The usual implementation technique is to save the entire binding time environment. However, saving only the necessary environment would insure correctness while conserving storage.*

BIBLIOGRAPHY

1. Goos, Gerhard, "Some Problems in Compiling Algol 68" Rechenzentrum der Technischen Hochschule, Munchen, Germany, paper delivered to ACM SIGPLAN Algol 68 Symposium, June, 1970.

2. Jorrand, Philippe, "Tutorial on Algol 68", AFIPS SJCC 1969 Proc., pp. 403-407.

3. Jorrand, P. and Wegner, Peter, Some Aspects of the Structure of Basel, Brown University, Providence, R. I., Jan., 1970.

4. Marshall, S. "An Algol 68 Garbage Collector", TM0111, Dartmouth College, Dec., 1969.

5. Naur, P. et al. "Revised Algol Report", CACM 6,1 Jan., 1963.

6. Peck, J. E. L. Draft of Algol 68 Companion, University of British Columbia, paper delivered at ACM SIGPLAN Symposium, June, 1970.

7. van der Meulen, S. G. and Lindsey, C. H. Informal Introduction to Algol 68. Amsterdam: Math. Centrum, 1969.

8. Van Wijngaarden, A.;Mailloux, B. J.; Peck, J. E. L.; and Koster, C. H. A. "Report on the Algoritmic Language Algol 68", Num. Math. 14, pp. 79-218, 1969.

9. Wegner, P. Programming Languages, Information Stuctures, and Machine Organization. New York: Mcgraw Hill, 1968.

*The help of John B. Johnston in the preparation of this paper is acknowledged with many thanks.

10. Wegner, P. "Three Computer Cultures; Computer Technology, Computer Mathematics and Computer Science," Advances in Computers, vol. 10, 1970.

11. Reynolds, John C. "GEDANKEN, A Simple Typeless Language Based on Principle of Completeness and the Reference Concept" CACM 13,5 May, 1970, pp. 308-319.

12. Randell, B. and Russell, L. J. Algol 60 Implementation, New York: Academic Press, 1964.

13. Alber, K. et al, Informal Introduction to Abstract Syntax and Interpretation of PL/1, IBM Lab., Vienna, TR25.099, June 30,1969.

14. Rosenkrantz, D. Private Communication

CONTRIBUTIONS