Addendum to M.L. Patrick Paper

Grover C. Simmons
Atlanta University Center

Key Words and Phrases: parallelism, polynomial root finding, real zeros
CR Categories: 5.15

May I suggest that eq. (2.7) in the M.L. Patrick paper [1] be modified to read:

$$r_1^{(j)} + r_2^{(j)} + \cdots + r_{m_j}^{(j)} = -a_{m_j-1}^{(2n-2j)} \quad \text{and}$$
$$\prod_{k=1}^{m_j} r_k^{(j)} = (-1)^{(m_j-nz)} a_{nz}^{(2N-2j)}, \quad r_k^{(j)} \neq 0, \tag{2.7}$$

where nz is the number of zero roots in the sequence $(r_2^{(j)}, r_3^{(j)}, \ldots, r_{m_j-1}^{(j)})$. The third degree polynomial, $p_3(x) = 2.5x^3 - 1.5x$ is a counterexample. In this case $m_1 = 1$ and $2n - 2 = 2$. Therefore, $p_3(x) = x$ and $r_2^{(2)} = 0$. No unique solution to the system of equations, $r_1^{(2)} + 0 + r_3^{(2)} = 0$ and $r_1^{(2)} \cdot 0 \cdot r_3^{(2)} = 0$, exists.

Received December 1972; revised October 1973

References

1. Patrick, M.L. A highly parallel algorithm for approximating all zeros of a polynomial with only real zeros. *Comm. ACM 15*, 11 (Nov. 1972), 952–955.

Author's address: Computation Center, Atlanta University Center, P.O. Box 324, Atlanta, GA 30314.

Some Remarks on Lookup of Structured Variables

Paul W. Abrahams
New York University

Key Words and Phrases: PL/I, symbol table, structured variables, qualified references, compilers
CR Categories: 4.12, 4.22

In [2], Gates and Poplawski present a technique for lookup of structured variables such as those used in COBOL and PL/I. Their technique is based on the construction of a deterministic finite-state machine whose input is a sequence of identifiers and whose final state designates a particular variable. As the author of a PL/I compiler [1], I would like to correct some of the statements made in the article and to compare their technique with one I have used.

Gates and Poplawski give the following example of a PL/I structure:

```
DECLARE  1  A,
            2    B,
                 3    C,
                 3    D,
            2    C,
                 3    D,
         1  B,
            2    A;
```

They then state that the reference A.C is ambiguous, although in fact, for PL/I, it is not. The rules of PL/I [4] state that if a reference exactly matches the fully qualified name of an item, then it refers unambiguously to that item; the existence of inexact matches is then irrelevant. In fact, one could not refer to the C component of A (which is a valid aggregate reference) unless PL/I had this rule. The Gates and Poplawski algorithm would easily be corrected, though, by flagging apparently ambiguous terminal states that represent exact matches.

Another difficulty with their algorithm is that the resolution of references must account for block structure. Since names in different blocks resolve differently, it is necessary, using the approach they suggest, to

These remarks are based on work supported by the U.S. Atomic Energy Commission under Contract AT(11-1)-3077. Author's address: Computer Science Department, New York University, Courant Institute of Mathematical Sciences, 251 Mercer St., New York, NY 10012.

create a separate deterministic machine for each block.

In the Gates and Poplawski method the number of states needed to represent a structure grows rapidiy with the depth of the structure. Since a programmer will generally use only one of the many possible names for an item, most of the time and space needed to construct the finite state machine will be wasted. For programs where most structures are not deeply nested (the usual case), almost any method will do.

An alternative technique is to construct a symbol table entry for each qualified reference that actually appears in the program, and then to resolve this reference exactly once. The symbol table entry records both the identifiers that appear in the name and the block where the reference occurs; references in different blocks generate distinct table entries. The symbol table search can be accomplished efficiently through hashing. With this approach, no effort is wasted on references that never actually occur. The one-time search for reference resolution can be done using the algorithm described in the first part of the Gates and Poplawski paper or using a similar algorithm described by Knuth [3, pp. 428–9].

Received October 1973; revised December 1973

References

1. Abrahams, Paul. A preliminary report on CIMS PL/I. Rep. COO-3077-31. AEC Comput. and Appl. Math. Center, New York U., New York.
2. Gates, Geoffrey, and Poplawski, David. A simple technique for structured variable lookup. *Comm. ACM 16*, 9 (Sept. 1973), 561–565.
3. Knuth, D.E. *The Art of Computer Programming, Vol. 1, Fundamental Algorithms*. Addison-Wesley, New York, 1969.
4. PL/I Language Specifications. Order No. GY-33-6003-2, IBM Corp., White Plains, N.Y.

An Alternative Approach to Mutual Recursion in Algol 60 Using Restricted Compilers

A. Balfour
Heriot-Watt University

Key Words and Phrases: Algol 60, mutual recursion, compiler restrictions
CR Categories: 4.12, 4.22

In any Algol 60 compiler containing a restriction on the sequence of declarations such that no identifier may be used before it is declared, difficulties arise when two or more mutually recursive procedures have to be declared in the same blockhead. Atkins [1] indicates one possible approach using an ingenious, but slightly artificial procedure GATE, which embraces not only the mutually recursive procedures involved but also what is in effect the main program.

An alternative, more natural approach is to create additional formal procedure parameters which can be used at the appropriate places in the various procedure bodies to avoid the problems referred to above. Consider, for example, three mutually recursive procedures A, B, C, each one of which calls the other two. The procedure definitions, in skeleton form, can be written as in Figure 1. In the main program A, B, or C can be called into action by writing A(B,C), B(C) or C, respectively.

Fig. 1.

```
procedure A(pB, pC);
procedure pB, pC;
begin
    ⋮
    pB(pC);
    ⋮
    pC;
    ⋮
end A;

procedure B(pC);
procedure pC;
begin
    ⋮
    A(B, pC)
    ⋮
    pC;
end B;

procedure C;
begin
    ⋮
    A(B, C);
    ⋮
    B(C);
    ⋮
end C;
```

The approach can obviously be generalized to handle any number of mutually recursive procedures. Normal parameters can also be associated with the procedures in the usual manner. For simplicity these were omitted in the example given. Mutually recursive function procedures can also be set up in a similar manner.

Received February 1973

References

1. Atkins, M.S. Mutual recursion in Algol 60 using restricted compilers. *Comm. ACM. 16*, 1 (Jan. 1973), 47–48.

Author's address: Department of Computer Science, Heriot-Watt University, 37–39 Grassmarket, Edinburgh EH1 2 HW, Scotland.