Pass 6: Conversion to Polish notation. Type checking.
21. May 1962

Peter Naur
No. 130

INTRODUCTION.

The analysis of expressions and conversion to final machine form starts in pass 6. In this pass expressions are converted into so-called inverse Polish form. In this form the operations are chosen to conform to those available in the order code of the machine, but no account is taken of the use of the available machine registers. In pass 7 this string is converted into an operation string which refers directly to machine registers.

THE FORM AND MEANING OF THE INVERSE POLISH NOTATION.

In the inverse Polish notation expressions consist of a string of operands (identifiers etc.) and operators (+ - / > etc.). However, parentheses have been eliminated. Examples:

a b + c d - /
a ¬ b c > ∧

The meaning of an inverse Polish string, i.e. the rules for evaluating the result of the expression, assumes the existence of a stack for holding the actual values of operands, including intermediate results. Let us denote this as follows:

array OPERAND STACK[1:some unknown upper limit]

The Polish string never refers directly to a given location in the OPERAND STACK. Rather the references are implied in the structure of the Polish string itself. Thus the intermediate results of the evaluation are anonymous.

Now the evaluatinn can be described as follows: Proceed through the string from left to right in a strictly sequential fashion. When encountering an operand place this at the top of the OPERAND STACK by performing the following operations:

last used := last used + 1;
OPERAND STACK[last used] := value(operand);

When encountering an operator, perform the corresponding operation on the values found at the top of the stack and place the result of the operation also at the top of the stack. The exact action depends on the nature of the operator. Unary operators ("negative", ¬ and others to be introduced later) do not change "last used". Example: ¬ produces the following action:

OPERAND STACK[last used] := ¬ OPERAND STACK[last used]

Binary operators (+ / > ∧ etc.) remove one item from the top of the stack. Example: / produces the following action:

last used := last used - 1;
OPERAND STACK[last used] := OPERAND STACK[last used] /
OPERAND STACK[last used + 1]

As an illustration a step-by-step evaluation of the two above examples will be presented. In addition to the values held in the OPERAND STACK the mathematical expression for these values are given. It should be carefully noted, however, that the OPERAND STACK can only hold values (numbers, logical values) and the expressions given are only for the reader's convenience.

Example of evaluation: a b + c d - /
Assume a=1, b=2, c=3, d=4.

Input symbol	a	b	+	c	d	-	/
After processing of input symbol:							
OPERAND STACK [1]	1(a)	1 (a)	3 (a+b)	3 (a+b)	3 (a+b)	3 (a+b)	-3 ((a+b)/(c-d))
[2]		2 (b)		3 (c)	3 (c)	-1 (c-d)	
[3]					4 (d)		

Example 2: a ¬ b c > ∧
Assume a=false, b=2, c=3.

Input symbol	a	¬	b	c	>	∧
After processing						
OPERAND STACK [1]	false (a)	true (¬a)	true (¬a)	true (¬a)	true (¬a)	false (¬a∧b>c)
[2]			2 (b)	2 (b)	false (b>c)	
[3]				3 (c)		

CONVERSION OF ALGOL EXPRESSION INTO INVERSE POLISH FORM.

The conversion of a parenthesized ALGOL expression into inverse Polish form may be accomplished by means of a method which has been described by Dijkstra (APIC Bulletin no. 7, May 1961 and ALGOL Bulletin Supplement no. 10: Making a Translator for ALGOL 60). This method makes use of priority numbers associated with the operators and parentheses as follows:

Priority number	Delimiter
0	begin [(if for
1	end]) then else
2	:=
3	≡
4	⊃
5	∨
6	∧
7	¬
8	< ≤ = ≥ > ≠
9	+ -
10	"negative" × /
11	↑

The method works briefly as follows: The ALGOL expression is scanned from left to right. Operands are immediately transmitted to the output. Operators and left parentheses are entered into an OPERATOR STACK. This will at any one time hold those operators which have already occurred in the input but which have not yet been intered in the output (the Polish string) because the quantity on which the operator will operate has not yet been completed. Each new operator encountered in the input is compared with the operator waiting at the top of the OPERATOR STACK. If the priority of the operator waiting in the stack is higher than that of the new operator the operator waiting is removed from the stack and sent to

the output. This comparison of priorities is repeated until no operator having a higher priority than the new operator is found in the OPERATOR STACK. Then the new operator is placed at the top of the stack. Left parantheses are treated like other operators. Right parentheses (incl. end] ')' then else) on the other hand will not be placed in the OPERATOR STACK, but will remove the corresponding left parenthesis.

Example of conversion from ALGOL to inverse Polish notation: ((a+b)/(c-d))

Input	(	(	a	+	b	)	/	(	c	-	d	)	)
After processing:													
OPERATOR STACK [1]	(	(	(	(	(	(	(	(	(	(	(	(	
[2]		(	(	(	(		/	/	/	/	/	/	
[3]				+	+			(	(	(	(		
[4]										-	-		
Output (Polish)			a		b	+			c		d	-	/

A more complete discussion of the operators used in Gier Algol will be given below.

TYPE CHECKING.

The above conversion algorithm may very conveniently be extended with facilities for a complete type checking and a detection of types of the operands for each occurrence of the operators. This latter is of interest particularly for the power operator because a distinction is needed between integer and real exponents.

For this present purpose we perform a pseudo evaluation of the expression at the same time as it is generated in the inverse Polish form. This pseudo evaluation will of course not involve actual values but will only operate with the types of the values. This, however, is exactly what is necessary to perform a type checking. In order to illustrate this approach we give another example of a conversion which includes the behavior of the STACK FOR TYPE OF OPERAND:

Example of conversion with type development: $(\neg a \wedge b > c)$, Boolean a; integer b; real c;

Input:	(	¬	a	∧	b	>	c	)		
After processing										
OPERATOR STACK [1]	(	(	(	(	(	(	(			
[2]		¬	¬	∧	∧	∧	∧			
[3]						>	>			
Output (Polish)			a	¬	b		c	>	∧	
STACK FOR TYPE OF OPERAND [1]			Boo	Boo	Boo	Boo	Boo	Boo	Boo	
[2]					int	int	int	Boo		
[3]							real			

Note that the final) produces the output of two operators > and ∧.

By this method it is possible to check that the types of operands are consistent with the operators operating on them. The principal consistency rules may be stated as follows:

Operator	Required type of operands	Yield results of type
Arithmetic	real integer	real integer
Relational	real integer	Boolean
Boolean	Boolean	Boolean

For binary operators the handling of the STACK FOR TYPE OF OPERAND may be modified in the following manner. Instead of imitating the work of the OPERAND STACK exactly it is possible to extract and check the type information concerning the first operand belonging to a binary operator at the time when this operator is entered into the OPERATOR STACK. The advantage of this is that we do not have to check the consistency of an operator and both of its operands at the same time. A disadvantage is that in the case of the arithmetic operators, $+ - \times / \uparrow$ we have to attach the type of the first operand to that operator which is entered into the OPERATOR STACK. Thus for example there will be two possible + operators in the OPERATOR STACK, one "integer+" and the other "real+". The following example shows the use of this method:

Example of conversion to Polish notation with modified type handling.
integer i, j, k, n, m; real a;
((i + a)↑(j + k × n ↑ m))

Input:	(	(	i	+	a	)	↑	(	j	+	k	×	n	↑	m	)			)
After processing:																			
OPERATOR STACK [1]	(	(	(	(	(	(	(	(	(	(	(	(	(	(	(	(			
[2]		(	(	(	(		r↑	r↑	r↑	r↑	r↑	r↑	r↑	r↑	r↑	r↑			
[3]				i+	i+			(	(	(	(	(	(	(	(				
[4]										i+	i+	i+	i+	i+	i+				
[5]												i×	i×	i×	i×				
[6]														i↑	i↑				
Output (Polish)			i		a	+			j		k		n		m	"↑i"	×	+	"↑i"
TYPE CHECK STACK [1]			in		re	re			in		in		in		in			in	re

Note that in this example the type of n↑m is taken to be integer. Further that it is assumed that there exist distinct operators "↑i" (power to integer exponent) and "↑r" (power to real exponent). The details of the treatment of arithmetic types will be discussed below.