STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

MR 115

Syntax-directed parsing of ALGOL 68 programs.

by

C.H.A. Koster

1969

Syntax-directed parsing of ALGOL 68 programs

Approximate text of a talk at the informal implementers conference in Vancouver, August 1969,

by C.H.A. Koster

Abstract: First it is shown how a well-known parsing method for context-free languages can be extended to languages defined by a van Wijngaarden two-level syntax.
Next, application of this technique to ALGOL 68 is explored.
Finally some conclusions are drawn concerning ALGOL 68 implementation.

1. Parsing according to a Context-Free grammar

The problem of parsing according to a Context-Free grammar $G = (V_n, V_t, E, F)$ is by now well understood, and can be attacked in a number of ways.

We take a variant of the "method of syntactic functions" [1], that is, we construct out of the CF grammar a program in ALGOL 60 by a process of (mechanical) transscription where a nonterminal in the syntax corresponds to a procedure in the parser. The parser consists of an "environment" and a "parser body". The environment contains a number of declarations for

1) a reading procedure *read* which converts each terminal symbol (word) to some unique integer
2) an integer array *I* containing the codes for the words of the sentence to be parsed, followed by a special code to mark the end of sentence
3) an integer variable *pin* pointing to the word in *I* presently under consideration, initially pointing to the first word in *I*
4) a boolean variable *b*
5) a reading procedure *R*, declared e.g. as

```
boolean procedure R(x); integer x;
if I [pin] = endmaker then R: = false else
if I [pin] = x then begin pin: = pin + 1; R: = true end
                   else R: = false;
```

The parser body is constructed as follows:

Let the rules be of the form lhs : rhs (lhs $\in V_n$)
Let rhs be of the form $a_1; a_2; \ldots ; a_k$. ($k \geq 0$)
Let a_i be of the form $b_1, b_2, \ldots , b_l$ ($l \geq 0$)
Let $\{x_i\} = V_n \cup V_t$. Choose for every (nonterminal of terminal) symbol x_i a corresponding ALGOL 60 identifier y_i.

Now each rule of the grammar is treated as follows:

1) replace the nonterminal symbol x_i which is its left hand side by

" *boolean procedure* y_i;
begin integer pi; pi:= pin; b:= false;"

2) replace each nonterminal x_j by " *if* ¬ y_j *then else* ".

3) replace each terminal x_j by " *if* ¬ $R(y_j)$ *then else* ".

4) remove all comma's and the colon.

5) replace each semicolon by "; *begin b:= true; goto end end; pin:= pi;*

6) replace the point by "; *begin b:= true; goto end end; pin:= pi;*
end : y_i:= b
end;"

The parser body now has to be concluded by the necessary read in statements and a call on the procedure corresponding to the special starting symbol E.

The parsing method described is reasonably efficiënt and applicable under reasonably general circumstances. For the following extension any other parsing method will work also provided it makes use of a correspondence between nonterminals and (recursive) parsing procedures.

2. Two-level grammars

In two-level grammars metanotions may be of three kinds

1) superfluous but convenient, e.g., THELSE.
These metanotions choose from a small set. By writing out for each alternative of that metanotion the rules in which it occurs, it can be removed.

2) counting, e.g., ROWSETY (counting the number of 'row-of's).
These metanotions choose from a countable set. They can not be removed, but they may be treated as numerical parameters to the parsing process.

3) recursive, e.g., MODE and its many poorer relations like MOOD and UNITED.

These metanotions choose from a recursive set. The two previous kinds are of course subsets of this one. We will investigate their treatment.

Chomsky has left a remarkable hiatus in his classification of automata : he forgot the weakest one, the class 4 automaton or "Finite Choice" automaton.
A FC automaton G = (V_n, V_t, E, F) has rules of the form a : A. where $a \in V_n$ and $A \in V_t$.

Using an obvious notation, a two-level grammar having only the first kind of metanotion is represented as a CF grammar G_1 with a FC grammar G_2 imposed, $(^{FC}_{CF})$, and is equivalent to some CF grammar which is probably much longer than the "underlying" CF grammar G_1.

An underlying CF grammar with a Finite State grammar imposed $(^{FS}_{CF})$ is more powerful than a CF grammar since it can describe the language $\{a^n b^n a^n \mid n \geq 0\}$.

Sintzoff [2] has proved a $(^{CF}_{CF})$ grammar to be equivalent to a Semi Thue system.
Later on we will make use of the concept of underlying CF grammar. From a two-level grammar many equivalent two-level grammars can be obtained with possible differing underlying CF grammars.

Therefore one should keep in mind that there are many underlying CF grammars for any given language.

3. Extension of the parsing method

The way metanotions are treated in the syntax of ALGOL 68 bears some resemblance to the passing on of parameters in ALGOL 60. The extension we attempt will therefore be adding parameters to the parsing procedures.

We consider a hypernotion like SORTETY unitary MOID clause as a "headword" unitary clause with two "affixes" SORTETY and MOID.
Thus we have to make a clear separation between headword and affixes, and, because of the properties of ALGOL 60, take care that the number of affixes of a specific headword always is the same.

For clarity sake we write SORTETY + unitary + MOID + clause.

In particular, we have to avoid "punning" [4], as e.g. in (8.1.1.a): SORTETY MOID confrontation may produce into (8.2.0.1.d): strong COERCEND.

Furthermore, in the left hand side we may only allow metanotions as affixes. A left hand side like (8.3.1.1.a) reference to MODE assignation, ensuring that the mode of the assignation begins with 'reference-to', is unacceptable.

In order to make these affixes all metanotions, we have to move the checking on affixes from the left hand side to the right hand side of the rule, by introduction of two special headwords equal and is plus, such that the terminal production of N_1 + equal + N_2 is empty if for N_1 and N_2 one same notion is substituted, and otherwise is some special redundant symbol. not present in any program.
Analoguously, N_1 + is + N_2 + plus + N_3 has as terminal production empty or the redundant symbol, depending whether the notion substituted for N_1 is the concatenation of those substituted for N_2 and N_3.
With some effort, equal and is plus can be defined by a two level syntax, but for our purpose it is important that their corresponding parsing procedures can be constructed such that they are terminating e.g., *equal* has two parameters (probably pointers to possibly circular lists). If those parameters are modes, the mode equivalence algorithm is involved (which terminates). If the modes turn out to be equivalent, parsing proceeds, otherwise it fails. It is interesting to remark that the changes we have to introduce to a two-level syntax bring it more nearly in the form advocated by Fraser Duncan [3]. In fact, the whole technique is without more applicable to his type of grammar.

4. Application to ALGOL 68.

Take the rule (8.3.1.1.a):

reference to MODE assignation : reference to MODE destination
becomes symbol , MODE source.

The first problem is how to split up this rule in a well defined headword and affixes.

Because of 8.3.0.1.a,we choose assignation as headword and MODE as affix. Secondly,the affix on the left has to be a metanotion. This leads to

MODE 1 + assignation : MODE 1 + destination, becomes symbol,
MODE 1 + is + reference to + plus + MODE 2, MODE 2 source.

Introducing special headwords dereference and same mode with obvious meaning,. we can write this

MODE 1 + assignation : MODE 1 + destination, becomes symbol,
MODE 2 source, dereference + MODE 1 + MODE 3, same mode +
MODE 2 + MODE 3.

An assignation is now recognized as follows:

1) a destination is sought and its à priori mode MODE 1 obtained.

2) a becomes-symbol is sought

3) a source is sought and its à priori mode MODE 2 obtained

4) from MODE 1 by dereferencing MODE 3 is obtained

5) it is checked whether MODE 2 and MODE 3 are the same mode.

Other special headwords which come to the mind are strongly coerceable to + MODE 1 + MODE 2 , etc.

The parsing procedures for these headwords need not be defined by syntax. Any semantic process involving affixes, defined in any manageable way, can be grafted onto the syntax.

If all those special headwords together with their affixes are left out, all remaining affixes have become superfluous, and can also be omitted. But then one has obtained an underlying context-free grammar of the two-level grammar of ALGOL 68. Any ALGOL 68 program is a program according to the underlying context-free grammar, because special headwords only limit the class of programs. Since the parser according to the underlying CF grammar terminates, and since we will take due care that the parsing procedures for our special headwords terminate, we can assure that we have a terminating parsing algorithm for ALGOL 68.

The frase in the Report about mode-independent parsing (0.1.4.3) in effect tells us that the underlying CF grammar of ALGOL 68 is unambiguous. This is not wholly true: is *x(1)* a call or a slice?

By a slight change in grammar (leaving the distinction between call and slice unresolved) this problem is resolved. The grammar of ALGOL 68 is reasonably mode-independent, apart from a small number of those trouble-spots.

5. Conclusions about implementation of ALGOL 68

Making use of the present technique for Syntax Directed parsing of ALGOL 68 programs, the following structure of a compiler is suggested.

pass 1 Get rid of comments, replace identifiers by unique integers, etc.
Find defining occurences of operators and mode-indications.
Find priorities.

pass 2 Mode independent parsing (according to underlying CF grammar).
Construct declarer list.
Find defining occurences of identifiers.
The result of this pass should not be a linear text but a tree, in which all structural information obtained is incorporated.
Thus, the following passes can make use of a simplified syntax.

intermediate Minimize size of declarer list.
Check for mode-indications without defining occurence, context conditions, etc.

pass 3 Mode dependent translation according to the simplified syntax of the tree obtained in pass 2.

pass 4 production of optimized code

The advantages of using the Syntax Directed approach are obvious:

1) instead of all the tedious detail of the implementation, the syntax used may be communicated, together with the algorithms for the special headwords used. Thus the machine-independent part of the implementation is communicable in a clear and concise form
2) it becomes easy to correct or change one's implementation
3) it becomes easy to experiment with variants of ALGOL 68

The advantage of getting a tree as output of pass 2 is that pass 3 and can work more efficiently when all problems of backtracking and local ambiguity have been removed in the relatively fast and cheap pass 2.

Vancouver, 28/8/69
Amsterdam, 16/10/69

References:

1. L. Bolliet, Compiler writing techniques, contained in Programming Languages, F. Genuys (Ed.), London, Academic Press, 1968

2. M. Sintzoff, Existence of a van Wijngaarden syntax for every recursively enumerable set, Annales de la Société scientifique de Bruxelles, Tome 81-II, 1967

3. F.G. Duncan, Notational abbreviations applied to the syntax of ALGOL, AB 26.3.5, august 1967

4. F.G. Duncan, An alternative presentation of the Formal Part of MR93, presented at the Tirrenia meeting of WG 2.1, june 1968