A SYNTAX MACRO PROCESSOR

P.L. Wodon
MBLE Research Laboratory, Brussels

1. The Context

Since syntax macros are not new, a few words of justification for still another experiment are necessary. The origin is the following question : "In which language to program the software of electronic telephone exchanges?" Roughly, this is the known problem of choosing a language for software programming, i.e. a language which should be "high-level" to ensure reliability and hardware-oriented to ensure efficiency. Telephone exchanges however, have particularities of their own, two of which are worth mentioning. Firstly, no two exchanges are exactly alike, and this rules out plain assembly code. Secondly, the exchange builders know what they want but seem to have difficulties in expressing it in terms of algorithms. This means that designing at once another special purpose language was out of the question.

Something flexible was needed. To provide for an immediately usable programming tool, a GPM-like macro generator [4] was put into service and is being used for producing software.

In fact, a set of macros which expand into assembly code define a programming "language". This "language" is of course extensible and very flexible so that the user can adapt it to his needs when he discovers them. This gives an important side-effect : the properties of a possible specialized programming language can be investigated simply by looking at the macros which are being used.

A processor like GPM, however, has three obvious disadvantages : the structure of programs does not appear in the text, there is no syntactic control and the system is an inefficient string processor.

In the present case, expansion efficiency has a secondary importance : it is the efficiency of the assembly code constituting the result of expansion which matters. It therefore remains to take care of program structure and syntactic control. This leads to syntax macros, seen as a system in which a language (i.e. a set of macro definitions and call formats) and its translation (the associated macro bodies) are defined together and thereafter used.

Such a system, which has obvious similarities with extensible languages and compiler compilers, is justified only if two conditions hold. Firstly, nothing better should be available. This is the case and in fact one of the aims is to find something better using the system itself. The second condition is that software programmers should be able to use it and this remains to be seen.

2. The System

The user of the system has to define formats and contexts for macro calls, he may consider this as the syntax of his language, and corresponding expansion rules, the associated "semantics". For example, he may define a syntax for arithmetic expressions and expansion into some assembly code. The call 'A+B' then expands into 'LOAD,A ; ADD, B;'. The intended use is along these lines but the system is more general and similar things have been made long ago (see e.g. [3]).

The problem is to have something to be put into the hands of software programmers, hence to choose a general yet not too exotic way of writing macro formats and expansion rules.

Formats are expressed as a set of context-free rules without restrictions. This is not ideal but general enough, well known and there are available parsing methods. Non context-free features may either appear in the expansion rules which have then to use tables or be left to an assembler processing the expanded string.

A parsing algorithm [1], derived from [2], is used to transform calls into syntactic trees. It is less efficient than algorithms which work on restricted classes of CF grammars. For the simplicity of use, it was chosen to put no restrictions on the CF rules. Indeed, it is easy to recognize at a glance that CF rules are indeed CF rules and difficult to see if CF rules constitute for example an operator precedence grammar.

To illustrate expansion, an over used example will serve once again :

1. sum : sum, "+", term.
2. sum : term.
3. term : term, "*", factor.
4. term : factor.
5. factor : "(", sum, ")" .
6. factor : [any identifier]

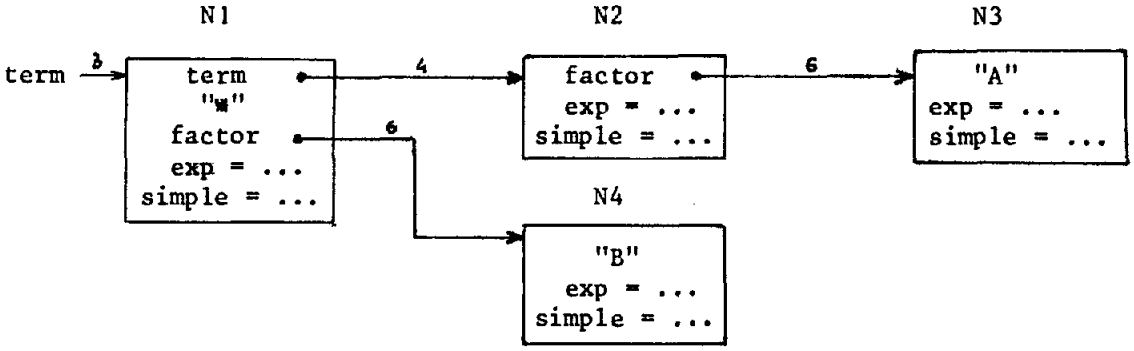
Usually, one expansion routine (i.e. macro body) is associated to each grammar rule (i.e. macro name). This was found to lack flexibility. For example, if the call 'A+B*C' expands into 'LOAD,A ; STACK ; LOAD, B ; ...', it is not easy to have 'A+B' expand into 'LOAD,A ; ADD, B' instead of 'LOAD, A ; STACK ; LOAD, B ; ADD, UNSTACK ;'. Either the grammar has to be changed or a bunch of predefined tree predicates have to be used. It was felt more natural and general to have several routines, characterized by mode and identifier, associated with each CF rule. One of these routines ('exp' in the example) is the expansion in the usual sense. For example, with the symbol '|' denoting concatenation :

```
1. string exp = if simple of sum then "LOAD", "else" "fi | exp of sum |
                if simple of term  then "ADD," | exp of term
                   else "STACK ;" | exp of term |  "ADD, UNSTACK;" fi.
   bool simple = false.
2. string exp = exp of term.
   bool simple = simple of term.
3. string exp = if simple of term then "LOAD, "else" "fi  | exp of  term |
                if simple of factor  then "MUL," | exp of factor
                   else "STACK;" | exp of factor  | "MUL, UNSTACK;" fi.
   bool simple = false.
4. string exp = exp of factor.
   bool simple = simple of factor.
5. string exp = exp of sum.
   bool simple = simple of sum.
6. string exp = id |";".
   bool simple = true.
```

As usual, an example chosen for its shortness is misleading in several respects. First of all, it makes the system look like a sledgehammer used to drive a nail. This cannot be helped. Furthermore, each routine consists here of a single expression but it could be a program with several statements. Also, each grammar rule is associated with the same number of routines with same mode and same

identifier. This is necessary only for grammar rules having the same left-hand side.

With this, the call 'A * B' is first transformed into a tree which can be visualized thus :



This tree is built by a syntactic analysis which stops when the smallest complete tree is obtained for the particular isolated call. This permits piecewise expansion. Each node is then considered as a structure with several fields. Some are pointers to other nodes : they correspond to non-terminals. Others correspond to the expansion routines. Roughly, expansion consists in filling in these fields with values by computing the associated routines : each routine is computed once, when its result is first needed.

Starting with field 'exp' in N1, the value of field 'simple' in N2 is needed (rule 3 : 'simple of term'). This one (rule 4) needs the field 'simple' in N3. Its value is 'true' (rule 6). The value of 'simple' in N2 and N1 become 'true', etc. Eventually, field 'exp' in N1 gets the value 'LOAD, A ; MUL, B;' (whatever the order of evaluation).

Together with tables, this mechanism takes care of various cases. For example, it suffices to add routines for calculating a type to have the same syntactic rules serve for fixed and floating point arithmetic. The association of several, instead of one, expansion routines to a syntactic construct is a source of simplicity of use as well as inefficiency of expansion. The latter is accepted, at least as long as nothing more is known on the actual needs in telephone exchange programming.

This is sufficient to give an idea of an experiment which will be complete only when the system has been tried to actually produce software for telephone exchanges. So far, it can only be said that people who use GPM are certainly able to use a simple syntactic macro processor.

References.

[1] Bouckaert, M.,Pirotte, A.,Snelling, M. "More efficient general context-free top-down parsers", MBLE Report R173 (Nov. 1971).

[2] Earley, Jay. "An efficient context-free parsing algorithm" Comm. ACM 13, 2 (Feb. 1970), 94-102.

[3] Irons, E. T. "The structure and use of the syntax-directed compiler" Annual Review in Automatic Programming, Vol. 3, 1963, pp. 207-227.

[4] Strachey, C. "A general purpose macrogenerator" Comput. J. 8 (1965), 225-241.