

# Structuring two-level grammar specifications

M.H. WILLIAMS

Department of Computer Science, Heriot-Watt University, 79 Grassmarket, Edinburgh EH1 2HJ

*The two-level grammar notation is a powerful tool for specifying the syntax, static semantics and even dynamic semantics of programming languages. However, it can also be a very difficult notation to follow. For this reason an approach to writing two-level grammars is advocated which explicitly indicates the 'direction of propagation' of metanotions. This suggests the requirement that the direction of propagation of each metanotion within a given hypernotation should be consistent and enables one to test a two-level grammar specification for circularity.*

## 1. INTRODUCTION

The two-level grammar or W-grammar notation<sup>1</sup> is a powerful tool for describing languages. By introducing parameters (metanotions) into nonterminal names and by using a separate set of productions (metaproduction rules) to define these parameters, it is a simple matter to produce grammars with an infinite number of production rules, and hence generate not only context-free languages but also context-sensitive and unrestricted (Type 0) languages<sup>2</sup>. This notation has the advantage that static semantic restrictions (and even dynamic semantics) can be expressed along with the syntax of a language in a single definition – but the disadvantage that for anyone who is not expert in the notation, it is considerably more difficult to follow than a simple notation such as BNF or an extended notation such as attribute grammars.

With a conventional single-level grammar it is very easy to construct a parse tree to assist one to visualize the way in which productions are applied to generate sentences of the language but with a two-level grammar this is much more difficult to accomplish owing to the complexity of the productions. In an attribute grammar the productions describing the syntax can be used to construct a parse tree, after which the passage of the attributes about the parse tree can be traced using the directions laid down in the specification. However, in a two-level grammar specification metanotions cannot be dealt with in isolation nor is there any indication of the 'direction' to be followed by individual metanotions. The Algol 68 report<sup>3,4</sup> is evidence in itself both of the power of the two-level grammar notation and its imperspicuity to all but a few.

The main problem of the two-level grammar notation is not dissimilar to the problem associated with the goto statement in programming languages. If goto statements are avoided and the control structures used in any program are restricted to certain simple types, it is possible to look at parts of a program in isolation and follow the flow of control through these sections. On the other hand by permitting uncontrolled usage of goto statements one may have a more general and flexible language but one forfeits the ability to look at a segment of code in isolation without first obtaining a picture of the overall control structure surrounding the segment in question.

The two-level grammar notation suffers from precisely this defect. In general it is not possible to look at part of a specification in isolation and master it without a reasonable understanding of the interface with the

remainder of the specification. Again the definition of Algol 68 is evidence of this fact.

This problem could be overcome if, where metanotions are used to carry information about the parse tree, a 'direction of propagation' is associated with these metanotions in the same way as directions are associated with attributes in an attribute grammar. This will certainly assist anyone who is not expert in the notation to obtain a clearer understanding of any particular two-level grammar in a shorter period of time. At the same time it will also benefit the language designer in that portions of a specification can be dealt with in isolation.

## 2. BRIEF OVERVIEW OF TWO-LEVEL GRAMMAR APPROACH

Whereas in a conventional one-level grammar one has a fixed and finite set of nonterminal symbols, the power of the two-level grammar approach lies in the idea that nonterminal symbols need not necessarily be fixed but may be treated as variable strings which are in turn controlled by a set of productions at a higher level. Thus a two-level grammar is composed of two finite sets of rules called *metaproduction rules* (or *metarules*) and *hyper-rules*. From these two sets the set of production rules is derived.

For example, in a one-level grammar one might have  
<assignment stm> ::=  
    <numeric variable> = <numeric exp>  
    | <boolean variable> = <boolean exp>

In a two-level grammar this might be written as

Metaproduction rules

TYPE:: boolean; numeric.

Hyper-rules

assignment stm: TYPE variable, equals symbol,  
                  TYPE exp.

In van Wijngaarden's notation for metasyntax, the nonterminals (called metanotions) are written in upper case letters, the terminals (which form part of the nonterminals of the syntax) are written in lower case letters, '::' is used to stand for 'is defined as' (in place

of '::<='), ';;' is used to separate options (in place of vertical bar) and '.' is used to mark the end of each metaproduction rule.

The notation for hyper-rules is similar using ':' instead of '::<=' and ';;' and '.' as before. In the metaproduction rules if a nonterminal or terminal is to be followed by another, they are simply juxtaposed (as in BNF); in the hyper-rules the symbol ',' is used as a separator indicating concatenation.

If the metaproduction rules are substituted into the hyper-rules in the above example, the following productions are obtained:

assignment stm: boolean variable, equals symbol, boolean exp.

assignment stm: numeric variable, equals symbol, numeric exp.

In these three types of rules, three types of symbols are used:

- (i) A *protonotion* is a possibly empty sequence of lower case letters and spaces (eg assignment stm) which is the nonterminal/terminal of the production rules and the terminal of the metaproduction rules. In a production rule a protonotion which ends in 'symbol' is a terminal, one which does not, is a nonterminal.
- (ii) A *metanotion* is a non-empty sequence of upper case letters (eg TYPE) and is the nonterminal of the metaproduction rules.
- (iii) A *hypernotation* is a possibly empty sequence of metanotions and/or protonotions (eg TYPE variable) and is the nonterminal/terminal of the hyper-rules.

These are summarized in Table 1.

The term 'terminal metaproduction' of a metanotion refers to any protonotion which can be derived from that metanotion by applying the metaproduction rules. To change a hyper-rule to a production rule:

- (1) Each metanotion found only once in the hyper-rule is replaced with a terminal metaproduction for that metanotion.
- (2) If a metanotion occurs more than once in the hyper-rule, each occurrence of that metanotion must be replaced with the same terminal metaproduction. This is referred to as the uniform replacement rule or consistent substitution rule.

For more details see Cleaveland and Uzgalis<sup>5</sup> or Marcotty et al<sup>6</sup>.

### 3. DIRECTION OF PROPAGATION

In order to understand a particular grammar and see how it is used to parse a string, it is helpful to have some idea of when the values represented by metanotions are being

passed up the parse tree, when they are being passed across from one node to another at the same level (within a production) and when they are being transmitted down the tree. In other words it is useful to be able to follow the 'direction of propagation' of metanotions in the same way that one can follow the direction of propagation of attributes in an attribute grammar.

For this purpose a metanotion *m* occurring within some hypernotation will be said to be synthesized (or upward propagating) if the protonotion which that metanotion represents is determined in the process of expanding the hypernotation to obtain a terminal string. Such a metanotion will be marked with an upward arrow. Likewise an occurrence of metanotion *m* is said to be inherited (or downward propagating) if the protonotion which the metanotion represents is determined elsewhere and is 'passed down' to the hypernotation concerned from an ancestral node in the parse tree. In this case *m* will be marked with a downward arrow. These arrows can be regarded as an addition to the subscripting mechanism and are ignored in applying the consistent substitution rule.

To illustrate this convention, consider the example given by Cleaveland and Uzgalis<sup>5</sup> of a two-level grammar which generates the set of strings of letters in which no letter is repeated.

#### Metaproduction rules

1. ALPHA::a;b;c;d;e;f;g;h;i;j;k;l;m;n;o;p;q;r;s;t;u;v;w;x;y;z.
2. LETTER:: letter ALPHA.
3. TAG::LETTER; LETTER TAG.
4. EMPTY::.
5. NOTION::ALPHA;NOTION ALPHA.
6. NOTETY::NOTION;EMPTY.

#### Hyper-rules

7. s:TAG unique.
8. LETTER TAG unique: LETTER symbol, TAG unique, where LETTER is not in TAG.
9. LETTER unique: LETTER symbol.
10. where LETTER1 is not in LETTER2 TAG: where LETTER 1 isnt LETTER2, where LETTER1 is not in TAG.
11. where LETTER1 is not in LETTER2: where LETTER1 isnt LETTER2.
12. where letter ALPHA1 isnt letter ALPHA2: where ALPHA1 precedes ALPHA2 in abcdefghijklmnopqrstuvwxyz; where ALPHA2 precedes ALPHA1 in abcdefghijklmnopqrstuvwxyz.
13. where ALPHA1 precedes ALPHA2 in NOTETY1 ALPHA1 NOTETY2 ALPHA2 NOTETY3: EMPTY.

Table 1. Nomenclature used for nonterminal/terminal symbols in the three types of rules

Type of rule	nonterminals	terminals
Metaproduction rules	metanotions	protonotions
hyper-rules	hypernotations	hypernotations
production rules	protonotions not ending in 'symbol'	protonotions ending in 'symbol'

This specification can be made easier to understand if the hyper-rules are rewritten with the direction of propagation of the metanotions indicated as follows:

7.  $s: TAG \uparrow unique$ .
8.  $LETTER \uparrow TAG \uparrow unique: LETTER \uparrow symbol, TAG \uparrow unique,$   
where  $LETTER \downarrow$  is not in  $TAG \downarrow$ .
9.  $LETTER \uparrow unique: LETTER \uparrow symbol$ .
10. where  $LETTER1 \downarrow$  is not in  $LETTER2 \downarrow TAG \downarrow$ :  
where  $LETTER1 \downarrow$  is not in  $LETTER2 \downarrow$ , where  
 $LETTER1 \downarrow$  is not in  $TAG \downarrow$ .
11. where  $LETTER1 \downarrow$  is not in  $LETTER2 \downarrow$ : where  
 $LETTER1 \downarrow$  is not in  $LETTER2 \downarrow$ .
12. where letter  $ALPHA1 \downarrow$  is not letter  $ALPHA2 \downarrow$ :  
where  $ALPHA1 \downarrow$  precedes  $ALPHA2 \downarrow$  in  
abcdefghijklmnopqrstuvwxyz;  
where  $ALPHA2 \downarrow$  precedes  $ALPHA1 \downarrow$  in  
abcdefghijklmnopqrstuvwxyz.
13. where  $ALPHA1 \downarrow$  precedes  $ALPHA2 \downarrow$  in  $NOTETY1 \downarrow$   
 $ALPHA1 \downarrow NOTETY2 \downarrow ALPHA2 \downarrow NOTETY3 \downarrow$ :  
EMPTY.

An example of a parse tree for a sentence of this language is given in Fig. 1. This shows where the protonotions are associated with the metanotions and how they propagate about the parse tree.

It is intended that this tracing of the direction of propagation of a metanotion should be regarded as something fundamental to two-level grammars as it is in the case of attribute grammars, but is introduced as an aid to the user in seeing how productions are selected. As will be seen in the next section some specifications have to be rewritten in order to be able to do this.

Nevertheless, where a direction of propagation can be established, it does make it easier for the reader,

particularly in the case of complex grammars with a large number of hyper-rules.

#### 4. TYPES OF HYPERNOTION

As a first step to determining the direction of propagation of metanotions, consider the different ways in which hypernotions may be used. Suppose that one has a hypernotation  $h$  containing one or more metanotions  $m_i$  ( $1 \leq i \leq n$ ). This hypernotation may be classified according to the way in which it is used, into at least one of the following categories:

- (a) *Predicate with no return value (PNR)*.  $h$  can produce only the empty string or protonotions which cannot be reduced to terminal strings (blind alleys), and all metanotions  $m_i$  are passed down to  $h$  from above (all  $m_i$  are inherited). Examples of this include:

where  $LETTER1 \downarrow$  is not in  $LETTER2 \downarrow TAG \downarrow$ ,  
where letter  $ALPHA1 \downarrow$  is not letter  $ALPHA2 \downarrow$ ,

from the example in the previous section.

- (b) *Predicate with returned value (PR)*. Once again  $h$  can produce only the empty string or blind alleys. However, in this case some of the  $m_i$  are inherited and some are synthesised. The simplest example of such a hypernotation is one which passes down a symbol table and an identifier and returns the mode of the identifier, or passes down an array name and a symbol table and returns the number of dimensions of the array.

- (c) *Terminal with no inherited metanotions*. Here  $h$  can

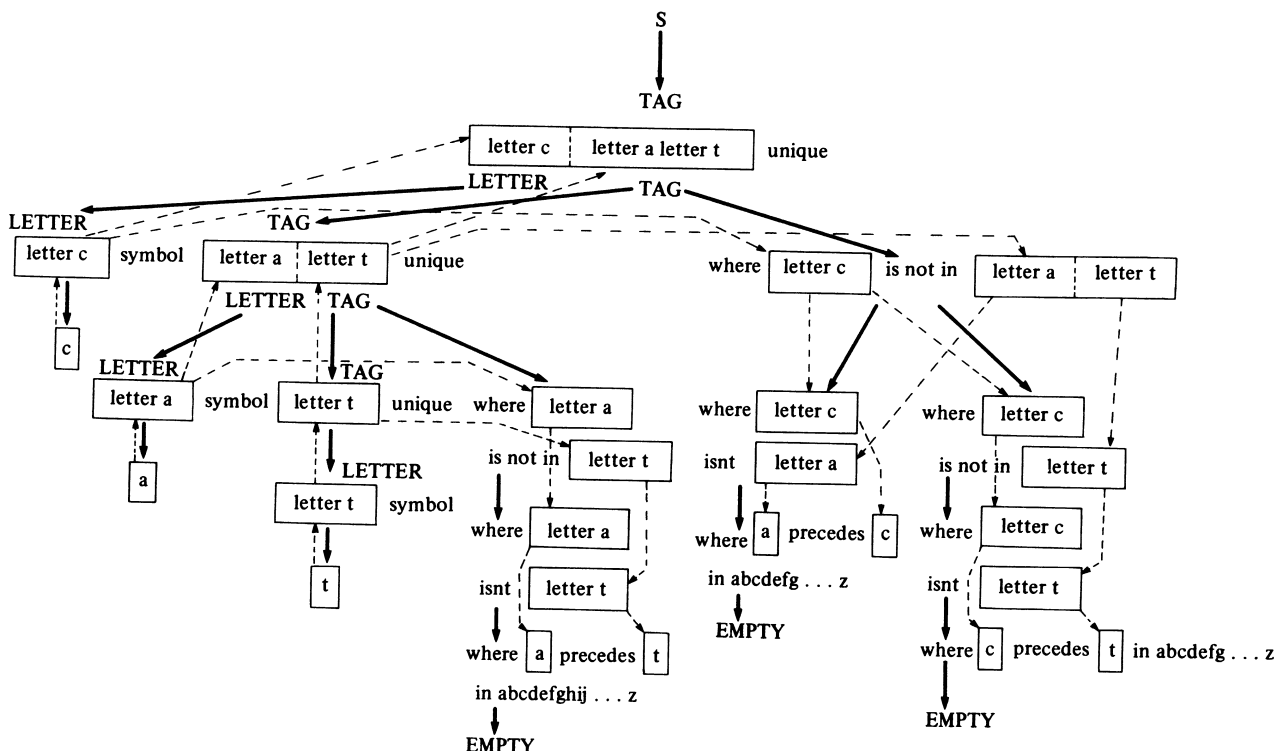


Fig. 1. Parse tree for a sentence of the language given in first example

produce only a terminal string (or the empty string) and none of the  $m_i$  are inherited. An instance of this type of hypernotation taken from the example in the previous section is:

LETTER $\uparrow$  unique

- (d) *Terminal with inherited metanotations.* Again  $h$  can produce only a terminal string (or the empty string) but in this case at least one of the  $m_i$  is inherited. An illustration of this type of hypernotation might occur in Algo168 where either begin-end or ( ) may be used to encompass a construction provided that the pair of delimiters used is of the same type (ie one cannot have begin... ). For example

Metaproduction rules

DELIM::end;close bracket.

Hyper-rules

stmt group: begin symbol, end group;  
open bracket symbol, close bracket group.  
DELIM group: DELIM stmt;  
stmt, separator symbol, DELIM group.  
DELIM stmt: stmt, DELIM symbol.

- (e) *Mixed hypernotation.* This is a hypernotation which is capable of producing both terminal strings (including the empty string) and protonotations which cannot be reduced to terminal symbols (blind alleys). Referring again to the example in the previous section, an instance of this type of hypernotation is

LETTER $\uparrow$  TAG $\uparrow$  unique

For the inexperienced reader it is confusing when different directions of propagation are associated with a particular occurrence of a metanotation within a given hypernotation depending on where the hypernotation is used. This can even result in a hypernotation falling into two different categories.

To illustrate this consider the example (taken again from Cleaveland and Uzgalis) of a grammar which generates all strings of the form  $a^n = b^n + c^n$  where  $a, b, c$  and  $n$  are positive integers,  $n > 2$ .

Note that Fermat's last theorem (which has remained unproven for generations) states that there do not exist any positive integers  $a, b, c, n$  satisfying the equation  $a^n + b^n = c^n$  for  $n > 2$ ; thus any string generated by this grammar will be a counterexample which disproves the theorem.

Metaproduction rules

1. N::one; N one.
2. EMPTY::.
3. NETY::N; EMPTY.
4. RADIX:: one one one one one one one one one one.
5. EXP:: one one N.

Hyper-rules

6. fermat: EXP poweresult N1, equal symbol, EXP poweresult N2, plus symbol, EXP poweresult N3, where N1 is N2 N3.
7. N1 poweresult N2: N3 number, power symbol, N1 number, where N2 is N3 to the N1 power.
8. where N1 is N2 to the NETY one power: where N3 is N2 to the NETY power, where N1 is N3 times N2; where N1 is N2, where NETY is EMPTY.
9. where NETY1 N is N times NETY2 one: where NETY1 is N times NETY2; where NETY1 NETY2 is EMPTY.
10. where NETY is NETY: EMPTY.
11. N1 number: N1 token; N2 number, NETY token, where N3 is N2 times RADIX, where N1 is NETY N3.
12. EMPTY token: zero symbol.
13. one token: one symbol.
14. one one token: two symbol.
15. one one one token: three symbol.
16. one one one one token: four symbol.
17. one one one one one token: five symbol.
18. one one one one one one token: six symbol.
19. one one one one one one one token: seven symbol.
20. one one one one one one one one token: eight symbol.
21. one one one one one one one one one token: nine symbol.

If one attempts to classify hypernotations in this specification one finds that in hyper-rule 6, 'where N1 is N2 N3' is a PNR hypernotation since the values of N1, N2 and N3 are passed down to it and it can produce only the empty string or protonotations which cannot be reduced to terminal strings; however, in hyper-rule 11, 'where N1 is NETY N3' is a PR hypernotation in which the values of NETY and N3 are passed down to it and the value of N1 is returned. A similar conflict occurs in several of the other rules.

It is obvious that if a hypernotation is being used in different ways at different points in the specification, resulting in confusion about the direction of propagation of its component metanotations, this hypernotation could be replaced by two or more different hypernotations each of which performs a single function. Alternatively it may be better to rewrite the hyper-rules in such a way that this confusion is removed. In the case of the above example, rules 6-11 can be rewritten and directions assigned to the metanotations as follows:

Hyper-rules

6. fermat: EXP $\uparrow$  poweresult N1 $\uparrow$ , equal symbol, EXP $\uparrow$  poweresult N2 $\uparrow$ , plus symbol, EXP $\uparrow$  poweresult N3 $\uparrow$ , where N1 $\downarrow$  is N2 $\downarrow$  N3 $\downarrow$ .
7. N1 $\uparrow$  poweresult N2 $\uparrow$ : N3 $\uparrow$  number, power symbol, N1 $\uparrow$  number, where N2 $\uparrow$  is N3 $\downarrow$  to the N1 $\downarrow$  power.
8. where N1 $\uparrow$  is N2 $\downarrow$  to the N4 $\downarrow$  one power: where N3 $\uparrow$  is N2 $\downarrow$  to the N4 $\downarrow$  power, where N1 $\uparrow$  is N3 $\downarrow$  times N2 $\downarrow$ .
- 8A. where N2 $\uparrow$  is N2 $\downarrow$  to the one power: EMPTY.
9. where NETY1 $\uparrow$ N $\uparrow$  is N $\downarrow$  times N2 $\downarrow$  one: where NETY1 $\uparrow$  is N $\downarrow$  times N2 $\downarrow$ .
- 9A. where N $\uparrow$  is N $\downarrow$  times one: EMPTY.
10. where NETY $\downarrow$  is NETY $\downarrow$ :EMPTY.

11. N1↑ number: N1↑ token;  
 N2↑ number, NETY↑ token,  
 where N3↑ is N2↑ times RADIX↓,  
 where NETY↓ N3↓ is rewritten as N1↑.  
 11A. where NETY↓ is rewritten as NETY↑: EMPTY.

## 5. RULES FOR ESTABLISHING DIRECTION OF PROPAGATION

The next step is to formulate a set of rules which will enable one to establish the direction of propagation of metanotions within a given grammar in some instances. Four fairly obvious rules are given below. These rules are straightforward and place no restrictions on the specification. However, they only establish a direction of propagation when certain obvious conditions hold.

- (1) *Absence from right-hand side.* If a hypernotation on the left hand side of a hyper-rule contains a metanotion  $m$  and at least one of the alternatives on the right hand side of the hyper-rule contains neither  $m$  nor any hypernotation derived from  $m$ , then the occurrence of  $m$  is inherited. For example, in a slightly simplified form of one of the hyper-rules from the definition of Algol 68 (Revised Report)

VIRACT declarer: VIRACT declarator;  
 TALLY applied mode indication with TAB.

VIRACT must be inherited since there is no occurrence of VIRACT or a hypernotation derived from VIRACT (virtual, actual or formal) in the hypernotation 'TALLY applied mode indication with TAB'. Similarly in the case of:

formal row NEST row: up to token option.

there is no hypernotation derived from NEST on the right hand side, and hence NEST must be inherited.

- (2) *Absence from left hand side.* If one or more alternatives on the right hand side of a hyper-rule each contains a single occurrence of a meta-notion  $m$  and the hypernotation on the left hand side contains neither  $m$  nor any hypernotation from which  $m$  might be derived, and if  $m$  can generate more than one possible protonotion, then the occurrences of  $m$  are synthesised. For example, in the simplified form of hyper-rule from the definition of Algol 68 considered in the previous rule:

VIRACT declarer: VIRACT declarator;  
 TALLY applied mode indication with TAB.

both TALLY and TAB must be synthesised since no ancestor of either occurs on the left hand side and both generate more than one protonotion. Hence one can write the rule as:

VIRACT↓declarer: VIRACT declarator;  
 TALLY↑ applied mode indication with TAB↑.

- (3) *Consistency within a hypernotation.* If one occurrence of a hypernotation  $h$  contains a metanotion  $m$  marked as synthesised (inherited) then any other occurrence of  $h$  in the grammar containing a corresponding occurrence of  $m$  (or a derivative or ancestor of  $m$ ) must have  $m$  marked as synthesised (inherited). For

instance, in the example of Fermat's last theorem, the hypernotation

EXP↑ powerresult N1↑

occurring in rule 6 will require all other occurrences of this hypernotation such as

EXP↑ powerresult N2↑ in rule 6  
 N1↑ powerresult N2↑ in rule 7

to have synthesised metanotions in the first and last positions.

- (4) *Consistency within a hyper-rule.* If metanotion  $m$  occurs in a hypernotation  $h$  and there are corresponding occurrences of  $m$  in the alternatives on the right hand side of the hyper-rule defining  $h$ , and if each appropriate occurrence of  $m$  on the right hand side is synthesised (inherited) then the corresponding occurrence of  $m$  in  $h$  is synthesised (inherited). Similarly if there is only one occurrence of  $m$  on the right hand side of the hyper-rule defining  $h$  then if either occurrence of  $m$  is synthesised (inherited) then the other must be synthesised (inherited) too. Thus the hyper-rule discussed in rules (1) and (2) would become:

VIRACT↓declarer: VIRACT↓declarator;  
 TALLY↑ applied mode indication with TAB↑.

As mentioned these rules only establish a direction of propagation for certain limited cases. Unfortunately they are not sufficient in themselves to determine the direction of propagation of most of the metanotions in any specification. Indeed it is usually possible to determine the direction of propagation after a careful study of the context, but at this stage no general set of rules has been established which will do this automatically.

## 6. THE PROBLEM OF CIRCULARITY

Just as one of the problems in one-level grammar specifications is that of cycles, and for any practical purposes a grammar must be cycle free, likewise one of the problems which can arise in an attribute grammar specification is that of circular definition of attributes. Knuth<sup>7</sup> drew attention to this problem and gave an algorithm for detecting such circularity.

The introduction of directions of propagation into a two-level grammar specification naturally raises the possibility that circularity might occur in such a specification. To illustrate the problem consider an adaptation of the example considered by Knuth.

Suppose that one wants to extend the definition of a binary integer, viz

<bit> ::= 0|1  
 <string> ::= <bit> | <string> <bit>  
 <integer> ::= <string>

to incorporate the value of the integer. Following the lines of Knuth's solution using an attribute grammar, a

two-level grammar specification (with direction of flow indicated) for this might be:

#### Metaproduction rules

1. VALUE:: NUMBER.
2. SCALE:: NUMBER.
3. LENGTH:: NUMBER.
4. NUMBER:: one NUMBER; EMPTY.
5. EMPTY::.

#### Hyper-rules

6. bit with VALUE $\uparrow$  and SCALE $\downarrow$ :  
zero symbol, where VALUE $\uparrow$  is EMPTY $\downarrow$ ;  
one symbol, where VALUE $\uparrow$  is SCALE $\downarrow$ .
7. list with VALUE $\uparrow$  and LENGTH $\uparrow$  and SCALE $\downarrow$ :  
bit with VALUE $\uparrow$  and SCALE $\downarrow$ , where LENGTH $\uparrow$  is one;  
list with VALUE1 $\uparrow$  and LENGTH1 $\uparrow$  and SCALE1 $\downarrow$ ,  
bit with VALUE2 $\uparrow$  and SCALE $\downarrow$ ,  
where VALUE $\uparrow$  is VALUE1 $\downarrow$  VALUE2 $\downarrow$ ,  
where SCALE1 $\uparrow$  is SCALE $\downarrow$  SCALE $\downarrow$ ,  
where LENGTH $\uparrow$  is one LENGTH1 $\downarrow$ .
8. integer with VALUE $\uparrow$ : list with VALUE $\uparrow$  and LENGTH $\uparrow$  and SCALE $\downarrow$ ,  
where SCALE $\uparrow$  is one.
9. where NUMBER $\uparrow$  is NUMBER $\downarrow$ : EMPTY.

If one had unintentionally written hyper-rule 8 as:

8. integer with VALUE $\uparrow$ : list with VALUE $\uparrow$  and LENGTH $\uparrow$  and SCALE $\downarrow$ ,  
where SCALE $\uparrow$  is VALUE $\downarrow$ .

the definition can easily be shown to be circular. Fig.2 illustrates this with the parse of the string '1'. Clearly the

metanotion VALUE in the hypernotation 'integer with VALUE' is not uniquely determined by the parse. If the above specification with the incorrect version of hyper-rule 8 had been written without any indication of the direction of flow, the circularity (if it can still be referred to as such) would not be easily detectable. Nevertheless, it still exists and the grammar will be ambiguous in this case.

## 7. CONCLUSION

While the two-level grammar notation is both powerful and flexible, it does suffer from the drawback that specifications in this notation can be very difficult to follow. In view of this a case has been presented for indicating the 'direction of propagation' of metanotions wherever possible in a two-level grammar specification.

For the inexperienced reader tracing the direction of flow enables him to view sections of a specification in isolation and to obtain a better understanding of the specification as a whole. However, determining the direction of propagation of a metanotion within a given hypernotation is, in general, not an easy task. Two different approaches have been suggested which may assist the reader in this respect, namely

- (a) Classification of hypernotations. This is based on an understanding of what the hypernotation does.
- (b) General rules about propagation. These are based purely on the form of the hyper-rules.

Language designers who use the two-level grammar notation to specify their languages should determine the direction of propagation of metanotions in their specification in order to ensure that this is consistent. The advantages of doing so are:

- (a) It will make the specification easier for the reader to understand.

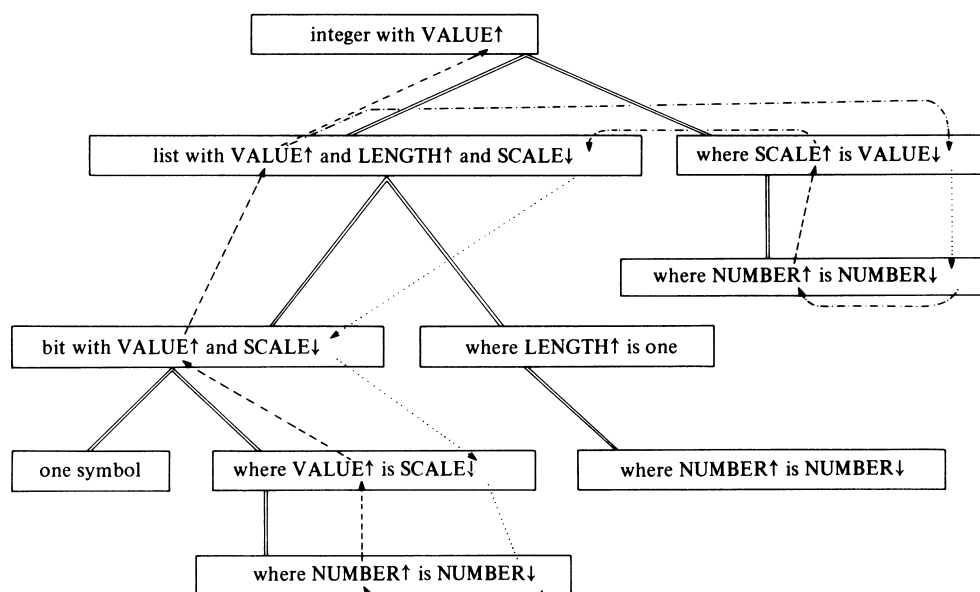


Fig. 2. Parse tree for the string '1' using the incorrect two-level grammar definition. The passage of the values represented by the metanotions VALUE and SCALE about the tree are traced with dotted lines indicating inherited values, dashed lines indicating synthesised values and a sequence of dots and dashes indicating horizontal transfer.

- (b) It enables part of a specification to be considered in isolation from the remainder of the specification thereby making it less prone to error.
- (c) It enables the designer to check for circularity in a specification.

The net effect of a practice such as this is to improve the readability and understandability of a specification and reduce the probability of errors, without any significant sacrifice to the power of the notation. In some ways this approach to writing two-level grammars can be compared with the discipline of structured programming in a conventional programming language. Thus just as indiscriminate use of the goto statement in computer programs is undesirable, a non-unique direction of propagation of a metanotion could be considered 'harmful'.

#### Acknowledgement

The author is indebted to the referee for helpful comments leading to the revision of this paper.

#### REFERENCES

1. C. H. A. Koster, Two-level Grammars, in *Compiler Construction An Advanced Course*, Ed. by G. Goos and J. Hartmanis, pp.146–156. Springer-Verlag, Berlin (1974).
2. M. Sintzoff, Existence of a van Wijngaarden syntax for every recursively enumerable set, *Ann. Soc. Sci. Bruxelles* **81**, 115–118 (1967).
3. A. van Wijngaarden, B. J. Mailloux, J. E. L. Peck, C. H. A. Koster, M. Sintzoff, C. H. Lindsey, L. G. L. T. Meertens, and R. G. Fisker, *Revised report on the algorithmic language ALGOL68*, Springer-Verlag, New York (1976).
4. A. van Wijngaarden, B. J. Mailloux, J. E. L. Peck and C. H. A. Koster, Report on the algorithmic language ALGOL68, *Numer. Math.* **14**, 84–218 (1969).
5. J. C. Cleaveland and R. C. Uzgalis, *Grammars for Programming Languages*, Elsevier, New York (1977).
6. M. Marcotty, H. F. Ledgard and G. V. Bochmann, A sampler of formal definitions, *ACM Computer Surveys* **8**, 191–276 (1976).
7. D. E. Knuth, Semantics of context free languages, *Math. Syst. Theory* **2**, 127–145 (1968).