PRIVE MF

Specifying precedence relations using a two-level grammar

Maarten M. Fokkinga, 27 febr. 1980.

Abstract. Usually the priority or precedence of operators, and of arbitrary composition schemes for expressions, is modelled within the context free grammar of a language by choosing distinct nonterminals for each priority class. This however gives rise to derivations (parse trees, productions trees) in which semantically redundant steps occur. We show, by means of an example, how the priority relations can be modelled in a simple two level (2VW) grammar such that the derivations do not contain semantically redundant steps.

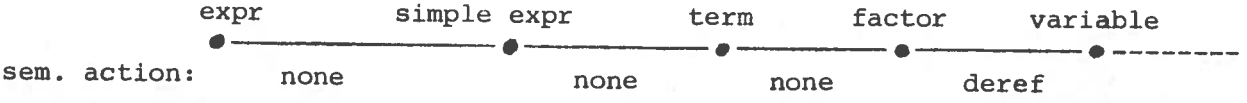
The example grammar. Consider the following grammar G1 for a subset of Pascal.

1. expr : simple expr , relational operator , simple expr ; simple expr .
2. simple expr : simple expr , adding operator , term ; term .
3. term : term , multiplying operator , factor ; factor .
4. factor : open par , expr , close par ; variable.

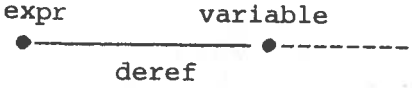
Before proceeding, we first remark that the nonterminals, occurring in the lhs of rules 1 : 4, actually denote the priority classes and that one also might wish to introduce four additional nonterminals in order to name each composition scheme. Consider e.g. grammar G2:

A. expr : relation ; simple expr .
B. simple expr : addition ; term .
C. term : multiplication ; factor .
D. factor : pack ; variable .
a. relation : simple expr , relational operator , simple expr.
b. addition : simple expr , adding operator , term .
c. multiplication : term , multiplying operator , factor .
d. pack : open par , expr , close par .

The following derivation (horizontally written parse tree), contains semantically redundant steps.



We would like amongst others to derive a variable directly from an expression:



This means that we need many more rules which enable (and force!) us to shortcircuit the semantically redundant steps (viz. the choice of the 2nd alternative in rules 1 : 3)
This is not difficult to achieve. Taking the second grammar as a staring point, we select just rules a : d and eliminate by means of rules A : D all occurrences of the nonterminals denoting the priority classes. This yields 39 rules.

A two level grammar, however, provides a convenient way to denote a big set of rules in a compact way. Rather surprisingly, the very above manipulation is implicit in a two level grammar derivation. Thus the following grammar G3 does the job.
Meta level (here characterizing the priority classes and priority relations)
A. EXPR :: relation ; SIMPLE .
B. SIMPLE :: addition ; TERM .
C. TERM :: multiplication ; FACTOR .
D. FACTOR :: pack ; variable .

Hyper rules (corresponding to semantic actions)
a. relation : SIMPLE1 , relational operator , SIMPLE2 .
b. addition : SIMPLE , adding operator , TERM .
c. multiplication : TERM , multiplying operator , FACTOR .
d. pack : open par , EXPR , close par .

The number of production rules is 39 (recall : production rules are obtained from hyper rules by eliminating all metanotions), and each semantically irrelevant derivation according to G1 has been short-circuited by a production rule of G3. The price paid is however very modest, for the actual rules written down correspond uniquely to those in G2 and there is also a one-one correspondence between derivations in G2 and "derivations in G3 when specified at the meta and hyper level". In fact, we have only introduced names for semantically relevant composition schemes, and distinguished these names and their rules from those dealing with syntactic priority.

It remains to be seen whether the elimination of semantically irrelevant production rules has practical advantages.

Note 1. Although parentheses have no non-trivial semantic effect, we consider them "semantically relevant". This is a consequence of the postulation that

"anything explicitly occurring in the program text has (should have) some meaning", one of the possible meanings being "don't change". ("Don't change the state" is denoted by "skip", "don't change the value" by parentheses). It is however possible to eliminate rule d from G3 . To this end, erase the first alternative of rule D and add three alternatives to each of rules a : c, giving "open par, EXPR, close par" instead of respectively the left operand, the right one, or both. (End of note 1.)

Note 2. The converse of the postulation of note 1 reads "every semantic action has (should have) some explicit representation in the program text". This forbids e.g. implicit coercions, and we think both postulations to be a valuable design objective for programming languages. (End of note 2.).

.-.-.-.-.-.-.-.-.-.

Opm. Joost 31-3-do:
G3 breidt enigszins ongebruikelijk: de metanotions zelf zijn al hyper-notions. Dit kan vermeden worden door, bv., overal -expression aan toe te voegen:

(B) SIMPLE-expression :: addition-expression ; TERM.

(6) addition-expression : SIMPLE-expression, add op, TERM-expression.

Opm G3 vormt geen volledig schema van 2 level grammatica's
(i) geen uniforme substitutie nodig
(ii) geen meta-notions in linkerzijde van hypernules
(iii) alleen reguliere regels op meta niveau

Opm in Algol68 heeft niet iedere priority class zijn eigen naam, maar slechts een geparametriseerde naam ± PRIO :: priO TALLY.
Daarmee kan het meta-niveau (en hyperniveau) vereenvoudigd worden.

NB zoz

inherited attribuut 'prec' $\in \{0,1,2,3\}$

productie	semantische regels	semantische condities
$S \to E$	$prec(E) := 0$	
$E_0 \to E_1 \leq E_2$	$prec(E_1) := prec(E_0)+1$ $prec(E_2) := prec(E_0)+1$	$prec(E_0) = 0$
$E_0 \to E_1 + E_2$	$prec(E_1) := prec(E_0)$ $prec(E_2) := prec(E_2)+1$ (0)	$prec(E_0) \leq 1$
$E_0 \to E_1 * E_2$	$prec(E_1) := prec(E_0)$ $prec(E_2) := prec(E_0)+1$	$prec(E_0) \leq 2$
$E_0 \to (E_1)$	$prec(E_1) := 0$	$(prec(E_0) \leq 3)$
$E \to Var$		$(prec(E_0) \leq 3)$

Ik probeerde een minstens even foeie Attributengr. te maken, maar dat gaat niet

Joost Engelfriet