Eliminating ambiguity in context free grammars

This note shows how to eliminate ambiguities in context free grammars by encoding operator precendence and associativity. ~~The point of the note is that there is a simple algorithm that does this, and so you can do this essentially without thinking. In fact it is easy to write a program does this.~~

~~I use BNF rather than extended BNF (which Coco-R uses) because associativity is not clear in extended BNF (EBNF) grammars, which is the second point of the note (more on this below)~~.

Consider the following ambiguous grammar

$$R = R\&R \mid R \oplus R \mid R \otimes R \mid R^* \mid (R) \mid a \mid b \mid c \tag{1}$$

The grammar consists of one non-terminal (R) and eight productions (divided by vertical bars). You should check that the grammar is ambiguous by constructing three different parsing trees for the string "$a\&b \oplus a^*$".

One can get rid of the ambiguity by encoding operator precedence and associativity. Let us assume that * binds more strongly than $\oplus$ and $\otimes$ and that these bind more strongly than &. We also assume that $\oplus$ and $\otimes$ bind equally tight and that ambiguity between these is resolved by choosing left associativity. This means that $a \oplus b \otimes c$ should be parsed as $(a \oplus b) \otimes c$. Let us also assume that & is right associative.

To construct an unambiguous ~~BNF~~ grammar for the same language do as follows. First encode operator precedence by picking one nonterminal for each level of operator precedence, using the given R for the least binding such level:

$$\begin{aligned} R &= R\&R \mid S \\ S &= S \oplus S \mid S \otimes S \mid T \\ T &= T^* \mid U \\ U &= (R) \mid a \mid b \mid c \end{aligned}$$

The idea here is that the first non-terminal is used to divide the string to be parsed into segments divided by &, then the second non-terminal is used to divide each of these segments into segments divided by $\oplus$ and $\otimes$ symbols and so on. Note that an S cannot contain a & except if it is contained in parentheses, and this is why the grammar encodes operator precedence.

Then encode associativity as follows

$$\begin{aligned} R &= S\&R \mid S \qquad (2) \\ S &= S \oplus T \mid S \otimes T \mid T \\ T &= T^* \mid U \\ U &= (R) \mid a \mid b \mid c \end{aligned}$$

You should check that this encodes operator associativity by checking that $a\&b\&c$ can only be parsed in one way. If an exam question asks for an unambiguous grammar ~~in BNF~~, then (2) is a correct answer.

~~The above grammar is an unambiguous grammar, but it suffers from *left recursion*, which means that it cannot be parsed using LL parsing techniques such as Coco-R uses. It can be parsed using LR parsing, however. The problem is that in the productions $S = S \oplus T$ and $S = S \otimes T$ the same non-terminal (S)~~

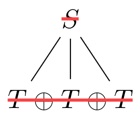
~~appears both on the left hand side of the production and as the first symbol of the right hand side.~~

~~I will show two ways of eliminating left recursion in (2). The first uses EBNF and is similar to what you have all done a few times in the course. The second stays within BNF~~.

~~In EBNF we can write the following~~

$$
\begin{aligned}
&\cancel{R = S\&R \mid S} && (3)\\
&\cancel{S = T\{\oplus T \mid \otimes T\}}\\
&\cancel{T = U\{{}^*\}}\\
&\cancel{U = (R) \mid a \mid b \mid c}
\end{aligned}
$$

~~I would argue that it is unclear from this grammar whether $\oplus$ and $\otimes$ are left associative or right associative. The reason is that the most natural way of drawing the parse tree of $a \oplus b \oplus c$ as an S starts like this~~



~~In a parser generator like Coco R, the parse trees are really constructed in the semantic actions (the commands between '(.' and '.)'). In the course we have seen how to construct semantic actions for a grammar like (3) parsing $\oplus$ and $\otimes$ as left associtive, and so we consider this as encoding left associtivity. But in general associtivity is unclear in EBNF. For example, I do not know how to interpret a production like $S = \{T\oplus\}T$ as left or right associative.~~

~~We can also eliminate left recursion by staying in BNF as follows~~

$$
\begin{aligned}
&\cancel{R = S\&R \mid S} && (4)\\
&\cancel{S = TS'}\\
&\cancel{S' = \oplus TS' \mid \otimes TS' \mid}\\
&\cancel{T = (R)T' \mid aT' \mid bT' \mid cT'}\\
&\cancel{T' = {}^*T' \mid}
\end{aligned}
$$

~~(Note that there are three productions for S' and two for T': in both cases the last production is empty). Again this is considered the standard solution to encoding left associativity without left recursion, but try to construct the parse tree for $S \oplus S \oplus S$ - it actually unfolds to the right! Again this can be saved by semantic actions.~~

~~Summing up: Grammar (1) is ambiguous, (2) is the result of eliminating ambiguity not worrying about left recursion, and (3) and (4) are unambiguous with no left recursion in EBNF and BNF respectively~~.