Grammars

Grammars are used to define the syntax of a language. Grammars are often encountered in the context of natural languages where the "English grammar" means, for example, the rules that say whether a list of words is or is not a correct sentence in English. Syntax is a structural description without regard for meaning. Sentences that are grammatically correct may also be meaningless. For example, the correctly formed English sentence "Red is blue." is not meaningful (at least in the ordinary interpretation of the words).

The syntax of properly formed data can also be expressed in a grammatical form. The grammar defines which patterns of symbols are valid in the language of the data and which patterns are not valid. As noted above, the grammar for some kind of data can tell whether a string of symbols is syntactically valid without saying what the symbols might "mean" (i.e., it does not give us any information about the semantics of the data).

A common way of defining a grammar in computationally-related situations is the Backus-Naur Form (BNF) and its extension (EBNF). BNF can be used to describe how to perform two different, though related tasks:

- to generate expressions that are valid in the grammar
- to recognize whether a given expression is valid in the grammar

BNF will be explained by using it to generate valid expressions.

A BNF grammar is a set of production rules that have the general form:

    term ::= expansion

where the symbols "::=" are special in BNF. The production rule means that wherever a "term" occurs in the sentence being generated it can be replace by "expansion". The expansion by consist of two forms:

- alternative: where alternative elements are separated by a vertical bar; these are alternatives in that the term is replaced by exactly one of the given alternatives; or
- sequence: where consecutive elements are written next to each other in order.

Each element may be:

- a symbol in the target language which cannot be further expanded; such symbols are enclosed in quote (') marks; or
- a term that is named on the left-hand side of a production rule.

Parentheses are used when needed to avoid ambiguity of grouping. Finally, the grammar has a unique start symbol that must appear on the left-hand side of at least one production rule. The start symbol defines where the generation begins. The generation stops when all of the symbols in the sentence are symbols in the target language.

Here is a small grammar for even whole numbers.

EWN ::= (‘-‘ number) | (‘+’ number | number)
number ::= even-digit | (digits even-digit)
digits ::= (digit digits) | digit
digit ::= ‘0’ | “1’ | “2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’
even-digit ::= ‘0’ | “2’ | ‘4’ | ‘6’ | ‘8’

where EWN is the start symbol. The first production rule means that an EWN has one of three alternatives: a number preceded by a ‘-‘ sign, a number preceded by a ‘+’ sign, or a number with no sign. The second production rule means that a number consists of a single even digit or a sequence of digits followed by an even digit. The third production rule means that the term digits is a digit followed by more digits or a single digit. The last two production rules enumerate the 10 possible digits and the five possible even digits. Notice that the parentheses are used for clarity and do not appear in the generated result.

One possible valid sentence that can be generated from this grammar is shown below, where each line represents an expansion of one term from the previous line:

EWN
number
digits even-digit
digit digits even-digit
5 digits even-digit
5 digits 4
5 digit digits 4
5 digit digit digits 4
5 digit 3 digits 4
5 digit 3 digit 4
5 2 3 digit 4
5 2 3 7 4

Thus, the sequence of symbols “52374” is (according to this grammar) a valid even whole number. There are, of course, an infinite number of valid strings that can be generated from this grammar (there should be if the grammar is to capture the sense of positive even whole numbers, of which there are an infinite number). The fact that a finite grammar can generate an infinite set of elements is due to the third production rule:

digits ::= digit digits | digit

that allows a single occurrence of the term “digits” to be replaced by the term “digit” and another occurrence of the term “digits”. This rule may be seen as a “recursive” rule because it defines the term “digits” in term of itself. Notice that the rule also includes an alternative where the term “digits” is not defined in terms of itself (i.e., replacing “digits”

by a single “digit”). The recursive part allows the number to have as many digits as desired. The non-recursive part guarantees that the generation will stop at some point.

Expressing optional elements and repeated elements can sometime be tedious in BNF. Therefore, an extended form of BNF, EBNF, is sometimes used. This extended form is no more powerful in that any grammar in EBNF can be expressed in BNF. But, EBNF can be more succinct. The additional operators introduced in EBNF and their meanings are:

? the preceding term is optional
* the preceding term appears zero or more times
+ the preceding term appears one or more times

The grammar for even whole number can now be expressed in EBNF as:

EWN ::= (‘+’ ? number) | (‘-‘ ? number)
number ::= digit* even-digit
digit ::= ‘0’ | “1’ | “2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’
even-digit ::= ‘0’ | “2’ | ‘4’ | ‘6’ | ‘8’

In this grammar, the first production rule means that an EWN is a number with an optional preceding ‘+’ sign or a number with an optional preceding ‘-‘ sign. The second production rule means that a number is a sequence of 0 or more “digit”s followed by a single “even-digit”. The last two production rules are as before. Notice that the recursive rule has been replaced by a rule allowing zero or more occurrences of a term.

Exercises:

1. Write a BNF grammar for phone number including area codes (do not worry about which area codes are valid and which are not).
2. Write a BNF grammar for US postal addresses. Describe in text what elements of a postal address you are considering in your grammar.
3. Write a BNF grammar for URL’s. Be sure to include the use of http, https, and ftp protocols. Use some representative set of organization types (e.g, .edu, .gov) and some representative document types (e.g., text, HTML, PDF). After you have done this exercise look at RFC 2396 (see: http://www.ietf.org/rfc/rfc2396.txt).
4. Use EBNF notation to solve any of the above problems.