Context free languages, context free grammars, and BNF

We describe context free languages, context free grammars, and Backus Naur Form (BNF) grammars. Although the set of palindromes is not a regular language, it is a context free language.

Context free languages and BNF

We consider a more powerful method of specifying sets of strings, namely, context free grammars. There are different notations for context free languages; we'll illustrate grammars in both the standard linguistics format and in Backus Naur Form (BNF).

A context free grammar consists of two finite alphabets, a terminal alphabet T and a nonterminal alphabet N, a start symbol (an element of N) and a finite set of rules. Each rule is of the form $X \rightarrow \gamma$, where X is a single nonterminal symbol, and γ is a (possibly empty) string of terminal and nonterminal symbols.

As an example, we consider a grammar for the set of strings over the alphabet $\{a, b\}$ that are palindromes. The terminal alphabet T is $\{a, b\}$ and the nonterminal alphabet N is $\{S\}$. The start symbol is S. The rules are $S \rightarrow \lambda$, $S \rightarrow a$, $S \rightarrow b$, $S \rightarrow aSa$ and $S \rightarrow bSb$.

Given a grammar G, we define the language of G, denoted $L(G)$, as follows. If α, β, and γ are strings of terminals and nonterminals and X is a nonterminal such that $X \rightarrow \gamma$ is a rule of the grammar, then we say that $\alpha X \beta$ derives $\alpha\gamma\beta$ in a single step, denoted $\alpha X \beta \vdash \alpha\gamma\beta$. Then $L(G)$ is the set of all strings of terminals that can be derived in a finite number of steps from the start symbol.

In the example grammar, the start symbol is S. Using the rule $S \rightarrow \lambda$, we have $S \vdash \lambda$. Because λ is a(n empty) string of terminal symbols, we conclude that λ is in $L(G)$. Similarly, we can derive the strings a and b in one step, so these also are in $L(G)$. The following multi-step derivation shows that $abbba$ is in $L(G)$.

$$S \vdash aSa \vdash abSba \vdash abbba.$$

It should be relatively clear that we can derive any palindrome using these rules, and that every string of terminals that we can derive is a palindrome. A language (set of strings) is context free if there is a context free grammar for it.

An equivalent notation for context free languages is Backus Naur Form (BNF). In BNF the set of palindromes over $\{a, b\}$ can be denoted as follows.

```
<palindromes> ::= <empty> | a | b | a<palindromes>a | b<palindromes>b
```

The notation `<empty>` denotes the empty string, λ. This may be viewed as a recursive definition of the set of palindromes. The base cases are λ, a, and b. The recursive cases are that if we have a palidrome s, then s with an a concatenated at each end is a palindrome, and s with a b concatentated at each end is a palindrome.

As for a context free grammar, a BNF grammar has a finite set of terminal symbols, a finite set of nonterminal symbols, a start symbol (one of the nonterminal symbols), and a finite set of rules. Each rule has a lefthand side, which is one of the nonterminal symbols, and a righthand side, which is a finite string of terminal and nonterminal symbols, possibly empty (which we denoted by `<empty>` above.) The lefthand and righthand sides are separated by `::=`.

In terms of the example above, the set of terminal symbols is $\{a, b\}$, the set of nonterminal symbols is $\{< palindromes >\}$, the start symbol is $< palindromes >$, and there are five rules:

```
<palindromes> ::= <empty>
<palindromes> ::= a
<palindromes> ::= b
<palindromes> ::= a<palindromes>a
<palindromes> ::= b<palindromes>b
```

There is a convention to abbreviate several rules with the same lefthand side by separating the different righthand sides with the symbol |, as shown above. (This convention is also used for context free grammars in standard linguistics notation.)

Parse trees

Parsing is the process of determining whether a given string can be derived from a give context free grammar. One way to depict the derivation of a string using a grammar is via a parse tree. For example, for the palindrome *abba*, we construct a parse tree in stages from the start symbol <palindromes>

```
                  <palindromes>
```

The start symbol <palindromes> is rewritten using the rule

```
    <palindromes> ::= a<palindromes>a
```

giving the tree

```
                  <palindromes>
                 /     |       \
                /      |        \
               a  <palindromes>  a
```

Then <palindromes> is rewritten using the rule

```
    <palindromes> ::= b<palindromes>b
```

giving the tree

```
                  <palindromes>
                 /     |       \
                /      |        \
               a  <palindromes>  a
                 /     |       \
                /      |        \
               b  <palindromes>  b
```

Finally, <palindromes> is rewritten using the rule

```
    <palindromes> ::= <empty>
```

giving the tree

```
        <palindromes>
       /     |       \
      /      |        \
     a  <palindromes>  a
       /     |       \
      /      |        \
     b  <palindromes>  b
             |
             |
          <empty>
```

If we concatenate together all the leaves, left to right, we get the string `abba`, as desired.

An interesting challenge is to write a context free grammar for the set of all strings over the alphabet $\{a, b\}$ that contain an equal number of a's and b's.

(Spoiler: One possible solution is the grammar

$$S \rightarrow \lambda | SaSbS | SbSaS.$$

For example, to generate the string $abba$, which has 2 a's and 2 b's, we can proceed as follows.

$$S \vdash SbSaS \vdash SbSa \vdash Sba \vdash SaSbSba \vdash SaSbba \vdash Sabba \vdash abba.$$

It is not difficult to see that the set of terminal strings derivable from S must have an equal number of a's and b's; it is a bit more challenging to see why every such string is derivable from S.)

Also: can we find a context free grammar for the set of all strings over the alphabet of $\{a, b, c\}$ that have an equal number of a's, b's, and c's?