Context-Free Languages

This material is described in Chapter 10 of the textbook.

- Context-free grammars are the most widely used specification tool for the syntactic structure of programming languages.
- Context-free grammars are more powerful than state diagrams, that is, they define a larger class of languages.

Backus-Naur formalism (BNF):

- the nonterminals are indicated with angle brackets
- the left and right sides of rules (productions) are separated by `::=`
- `|` indicates alternative definitions

Example. `<expr> ::= <expr> + <expr> | <expr>` $\times$ `<expr> | (<expr>) | a`

Note that the string `a + a` $\times$ `a` has two different derivations.

Convention: In the following we will use $\rightarrow$ instead of `::=` in the productions.

Example. Design a context-free grammar for the language

$$\{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$$

Formally a context-free grammar (CFG) is a 4-tuple (V, Σ, P, S) where

1. V is a finite set of nonterminals (variables)
2. Σ is a finite set of terminals; Σ and V are disjoint

3. P is a finite set of productions, the left side of each production is a nonterminal and the right side is a string of nonterminals and terminals

4. $S \in V$ is the start nonterminal

Let w_1 and w_2 be strings over $\Sigma \cup V$ (terminals and nonterminals). We say that w_2 is immediately derivable from w_1 if w_2 can be obtained from w_1 by replacing one occurrence of a nonterminal by a string that appears on the right side of a production for that nonterminal. In this case we write $w_1 \Rightarrow w_2$.

The language generated by the grammar is the language of terminal strings that can be derived from the start nonterminal, that is,

$$\{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

We write $u \Rightarrow^* v$ if $u = v$ or there exists a sequence of strings $u_1, \ldots, u_k$, $k \geq 0$, such that

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$$

Example. Consider the grammar

$$G = (\{S\}, \{a, b\}, P, S)$$

where the productions of P are

$$S \to aSb \mid SS \mid \varepsilon$$

What is the language generated by this grammar?

A context-free grammar where all productions are of the forms

$B \to bC$

$B \to b \mid \varepsilon$

where B, C are nonterminals and b is a terminal, is called a <u>regular grammar</u>. It is not difficult to show that regular grammars generate exactly the regular languages. This means that every regular language is generated by some context-free grammar. We have already seen that there exist context-free languages that are not regular.

Ambiguity

Consider our earlier example

`<expr>` $\to$ `<expr> + <expr> | <expr>` $\times$ `<expr> | (<expr>) | a`

For example, `a + a` $\times$ `a` has two different parse trees. A grammar that allows two different parse trees for the same terminal string is said to be <u>ambiguous</u>.

In the above example, if the expression is evaluated according to the parse tree, the two parse trees may produce different values!

We can redesign the above grammar by introducing new nonterminals:

`<expr>` $\to$ `<term> | <expr> + <term>`

`<term>` $\to$ `<factor> | <term>` $\times$ `<factor>`

`<factor>` $\to$ `(<expr>) | a`

This grammar generates the same strings (expressions) as the original one and it is <u>unambiguous</u>.

Pushdown automata (Section 10.5)

A pushdown automaton (PDA) can write symbols on a stack and read them back later. Writing a symbol “pushes down” all the other symbols on the stack. The symbol on the top of the stack can be read and removed ("pop" operation), see Figure 1. The definition of pushdown automata and examples are given on pages 216–219 in the textbook.

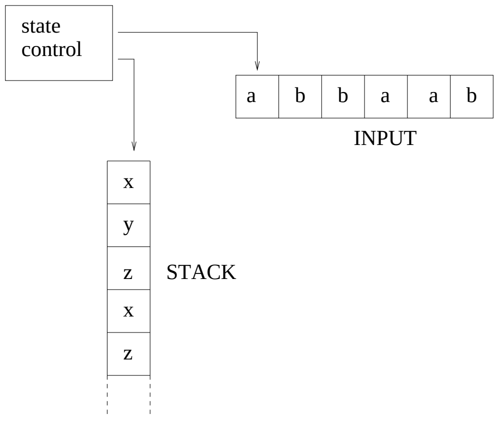

Figure 1: A pushdown automaton

Example. Construct a pushdown automaton for the language

$$\{a^i b^i \mid i \geq 0\} \cup \{a^{2i} b^i \mid i \geq 0\}$$

(To be done in class.)

Pumping lemma for context-free languages (Section 10.6)

For every context-free language L there is a constant p such that every string $s \in L$ of length at least p can be written as

$$s = uvwxy$$

where

(i) $v \neq \varepsilon$ or $x \neq \varepsilon$

(ii) $|vwx| \leq p$

(iii) for each $i \geq 0$, $uv^i wx^i y \in L$

Examples. Show that the following languages are not context-free:

a) $\{a^i b^i c^i \mid i \geq 0\}$

b) $\{ww \mid w \in \{a,b\}^*\}$

Applications: RNA secondary structure prediction using context-free grammars

Regular languages and finite state machines are often used in DNA and protein sequence matching problems (see notes for week 1). The use of regular languages is based on the assumption that the mutations that caused variations in the sequence occurred independently of each other.

The situation is different when we consider the secondary (or three dimensional) structure of an RNA molecule. RNA molecules with same secondary structure usually have the same function. Because of the way RNA molecules fold, in order to preserve the secondary structure, variations to nucleotides in one part of the sequence must be matched in the corresponding bonded subsequence (the bonded subsequences are called a *stem.*) This type of nested dependencies can be described using context-free grammars.

We consider a simple example. The RNA nucleotides contain four different bases: adenine (A), guanine (G), cytosine (C), uracil (U). C and G, and A and U, respectively, are complementary. In the below grammar we use lower case letters (a, g, c, u) since they are the terminals of the grammar.

To generate RNA structures we use a grammar (V, Σ, P, S) where $V = \{S\}$, $\Sigma = \{a, g, c, u\}$, and P consists of the productions:

(i) $S \longrightarrow aSu \mid uSa \mid cSg \mid gSc$

(ii) $S \longrightarrow aS \mid cS \mid gS \mid uS$

(iii) $S \longrightarrow Sa \mid Sc \mid Sg \mid Su$

(iv) $S \longrightarrow SS \mid \varepsilon$

A parse tree for the terminal string

$$ccugagaggcaaccuagaaggu \tag{1}$$

is given in Figure 2. The parse tree corresponds to the RNA secondary structure with two stems (or bonded subsequences) that is illustrated in Figure 3.

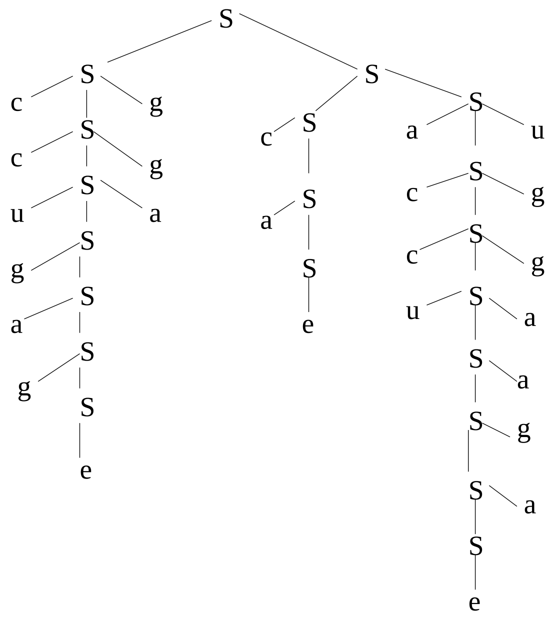

Figure 2: Parse tree for an example RNA structure. In the figure, “e” denotes the empty string ε.

From the parse tree we can read the secondary structure: the application of the grammar rules (i) produces a stem of bonded base pairs. Note that the grammar is *ambiguous*: the same RNA sequence (1) has many other parse trees. The ambiguity is, for the most part, caused by the possibility of applying the rules (ii), (iii) in different order. Derivations that differ only in the order of application of the rules (ii) and (iii) correspond to the same secondary structure.

Above we have outlined how context-free grammars can be used to model RNA folding. A more detailed study tells us that complementary pairs (C–G and A–U) are the likeliest candidates to form base pairs but, sometimes, also other pairs are formed. For more accurate

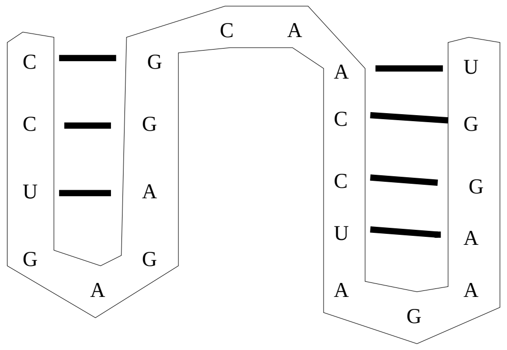

Figure 3: An RNA secondary structure. The thick lines indicate bonds between complementary bases in a stem.

secondary structure prediction the grammar rules need to be augmented with probabilities. The computerized methods used for RNA secondary structure prediction use probabilistic models called *stochastic context-free grammars.* An introductory survey can be found e.g. in [1].

References

[1] R. Durbin, S. Eddy, A. Krogh, G. Mitchison, *Biological sequence analysis: Probabilistic models of proteins and nucleic acid.* Cambridge University Press, 1998.