CS241 – Week 5 Tutorial

Regular Languages Part 2 & Context-Free Grammars

Spring 2018

1 Scanning

Up until this point we've only used DFAs for *recognition* – answering a yes/no question about whether a word is in the language specified by that DFA. However, DFAs can also be used for *scanning* – breaking up a string into a sequence of words in the language. Unlike recognition, scanning cannot be done perfectly in linear time, so we usually have to settle either for an imperfect algorithm or superlinear time. Also, scanning is sometimes ambiguous. For example, if we want to break the string "aa" into tokens of type "a" or "aa", we have two choices: breaking it into two "a" tokens or a single "aa" token.

1.1 The Simplified Maximal Munch Algorithm

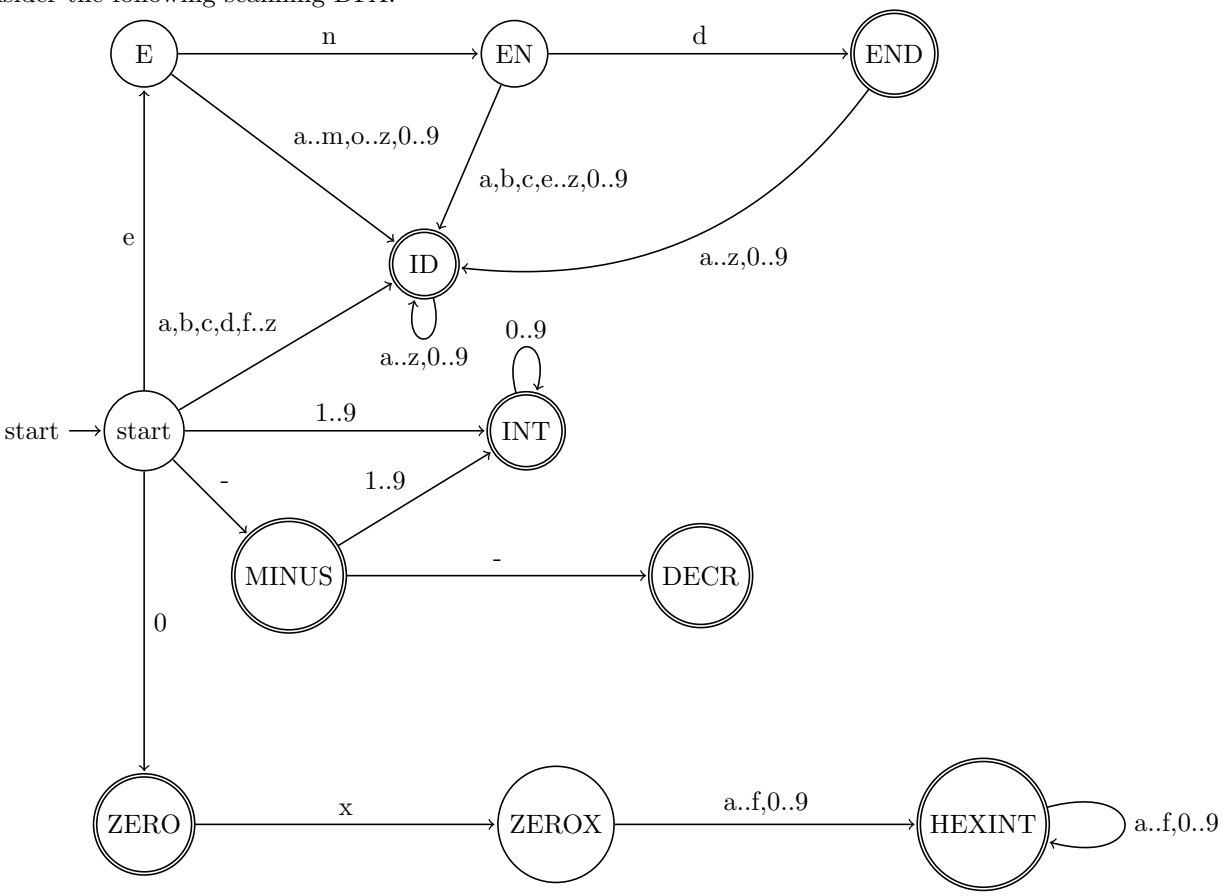
While there are a number of scanning strategies, in this course we focus on *simplified maximal munch*, a left-to-right linear time scanning algorithm which produces the largest tokens it can at each step. The simplified maximal munch algorithm works as follows:

1. Start at the start state of the scanning DFA.
2. Read characters until an error is reached or input is exhausted, keeping track of the current state and previous state. In addition, keep track of the characters read.
3. If input is exhausted and the current state is an accepting state, emit the characters read as a token. Otherwise, signal an error in tokenizing.
4. If an error is reached and the previous state is an accepting state, emit the characters read (except for the character that caused the error) as a token. Then resume from step 1 using the remaining input (starting with the character that caused the error) with both the current and previous states set to the start state of the scanning DFA.

Usually we describe tokens by their *kind* (usually the name of the accepting state which they were emitted from) and their *lexeme* (the characters that make up the token).

1.2 Exercises

Consider the following scanning DFA:



Give the sequence of tokens produced for each input below. If there is an error, put "ERROR" followed by the substring of tokens read since the last token emitted.

1. 0xa0xb0xcd
2. 0xend---
3. 1234-120xb
4. abcend--en-3
5. 01end-end10

2 Context-Free Grammars

A Context-Free Grammar $G = (N, \Sigma, P, S)$ consists of:

- N: the set of nonterminal symbols (or "nonterminals") (sometimes also called V).
- Σ: the alphabet, or the set of terminal symbols (or "terminals") (sometimes also called T).
- P: the set of production rules (or "productions") (sometimes also called R).
- S: the start nonterminal (or "start symbol").

We usually denote individual terminals by lowercase letters near the start of the alphabet $(a, b, c, \dots)$, nonterminals by uppercase letters $(A, B, C, \dots)$, strings of terminals by lowercase letters near the end of the alphabet (u, v, w, x, y, z), and (possibly empty) strings of terminals and nonterminals by Greek letters $(\alpha, \beta, \gamma, \dots)$. We usually simply represent a grammar by its set of production rules, with N and Σ implicitly defined by which terminals and nonterminals appear in the production rules. S is usually named S in the grammar, and is usually given the first production in the grammar as a reminder that it's the start nonterminal.

A production is a statement of the form $A \to \gamma$, where A is a nonterminal and γ is a string of terminals and nonterminals as stated above. A production means "anywhere in our string where we have an A, we can replace it with γ."

A language is context-free if and only if it is defined by some context-free grammar G. A word is in $L(G)$, the language specified by G, if we can take the string "S" and successively apply production rules until we arrive at w. For example, suppose G is the following grammar:

$$S \to 0S1$$
$$S \to 01$$

The word 000111 is in G since we can obtain 000111 by the following sequence of replacements:

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$$

Where $\Rightarrow$ means "the right hand side is obtained by applying a single production rule to the left hand side." This is called a *derivation* of 000111. Sometimes it's convenient to say that something can be derived after arbitrarily many (0 or more) production rules are applied, in which case we can use $\overset{*}{\Rightarrow}$, such as:

$$S \overset{*}{\Rightarrow} 000111$$

In general, $\Rightarrow$ is more convenient to show people why a word is in the language (it's the equivalent of "showing your work"), whereas $\overset{*}{\Rightarrow}$ is more convenient for stating facts or requirements (such as "this only holds for grammars where $S \overset{*}{\Rightarrow} \epsilon$").

While we can replace any nonterminal when using $\Rightarrow$, usually we prefer to consistently replace either the leftmost or rightmost nonterminal first at each step. These are called *left derivations* and *right derivations* (or leftmost and rightmost derivations) respectively. The individual steps (such as $00S11$ above) are called *left-sentential forms* (or *right-sentential forms*).

2.1 Exercises

2.1.1 Derivations

Let G be the following grammar:

$$S \to aAb$$
$$S \to bAa$$
$$A \to aSa$$
$$A \to bSb$$
$$A \to c$$

1. Show that the word $abacbbb$ is in $L(G)$ by giving a derivation.
2. Which of the following are valid derivations of words in $L(G)$?
   (a) $S \Rightarrow aAb \Rightarrow aaAab \Rightarrow aacab$
   (b) $S \Rightarrow aAb \Rightarrow aaSbb \Rightarrow aaaAbbb \Rightarrow aaacbbb$
   (c) $S \Rightarrow bAa \Rightarrow bca$
   (d) $A \Rightarrow aSa \Rightarrow aaAba \Rightarrow aacba$
   (e) $S \Rightarrow aaSab \Rightarrow aaaAbab \Rightarrow aaacbab$

2.1.2 Designing Grammars

1. Write context-free grammars for the following languages:
   (a) The language $L = \{\text{my}, \text{name}, \text{is}, \text{inigo}, \text{montoya}\}$.
   (b) The language of all words over $\Sigma = \{a, b, c\}$ not beginning with b.
   (c) The language defined by the regular expression $(0|1)^*((00)^*1)^*010$.
   (d) The language $L = \{a^n b^n c^m d^m : n, m \in \mathbb{N}\}$, where a^n means "n copies of a in a row."
   (e) The language of palindromes over $\Sigma = \{a, b\}$. A *palindrome* is a word or phrase which is spelled the same forwards and backwards, for example "ABBA," "racecar," or "able was I ere I saw elba"
2. Argue that every regular language is context-free by describing how to convert any regular expression into a context-free grammar.