CS 361 Meeting 13 — 3/9/20

Announcements

1. Homework 5 is online. Due Friday.
2. 3/16 will be midterm week.

Informal Formal Grammars

1. Basic introduction to context-free grammars.

   - A grammar is a way of specifying a language using rules like the following which (informally) says that anything composed of a variable followed by an assignment operator and an expression is a valid statement.

     < stmt > → < var > = < expr >

     In this particular notation for writing grammars (a variant of BNF or Backus Normal Form (or Backus Naur Form), which was originally used to describe the programming language Algol 58 back in 1958!), the symbols in angle brackets denote classes of syntactic phrases and are called *variables* (or non-terminals).

     The symbols not in angle brackets denote components of strings in the language being described. They are called terminals.

     The rules are sometimes called productions.

   - The syntactic phrases in most interesting grammars are frequently defined recursively (either directly or indirectly).

     < stmt > → **while** (< expr >) < stmt >

   - When used as a notation for specifying languages, various notational conveniences are employed (such as using a | to abbreviate a set of rules that would have the same phrase type on the left hand side).

     < stmt > → < var > = < expr >
     | **while** (< expr >) < stmt >

   - Given this we can give a complete set of rules describing a simple language capturing some of the syntax of common control structures and assignment statements:

     < stmt > → < var > = < expr >
     | **if** (<expr>) <stmt>
     | **if** (<expr>) <stmt> **else** <stmt>
     | **while** (<expr>) <stmt>
     <expr> → <var>
     <var> → x | y | z

   - We can view the rules of a grammar as specifications of relationships between sets. For example,

     < stmt > → < var > = < expr >

     implies

     < stmt > ⊂ < var > = < expr >

   - It is more common to view the rules of a grammar as replacement rules. That is, given a string containing terminals and variables, we can replace any of the variables with the right hand side of any rule with the variable's name on the left.

   - For example, the grammar above lets us write:

     < stmt > ⟹ if (<expr>) <stmt>
     ⟹ if (<var>) <stmt>
     ⟹ if (x) <stmt>
     ⟹ if (x) < var > = < expr >
     ⟹ if (x) y = < expr >
     ⟹ if (x) y = < var >
     ⟹ if (x) y = z

2. Grammars of this sort are called context-free grammars.

Slightly More Formal Grammars

1. The notation shown above is typical when context-free grammars are actually used to describe programming languages.

Click here to view the slides for this class

2. When studied as an example of a notation for describing languages from a theoretical standpoint, a slightly different notation is typically used.

   - Instead of words in angle brackets, variables are typically denoted using capital letters.
   - Lower case letters and digits are used for elements of the alphabet of the language being defined.

3. In addition, theoretical studies typically focus on sillier languages. For example, consider the following grammar:

   E $\rightarrow$
   E $\rightarrow$ 0E
   E $\rightarrow$ 1D
   D $\rightarrow$ 1E
   D $\rightarrow$ 0D

   - Can you tell what language this grammar describes?
   - It might help to consider a derivation like:

   $$E \Longrightarrow 0E \Longrightarrow 00E \Longrightarrow 001D \Longrightarrow 0010D \Longrightarrow 00101E \Longrightarrow 00101$$

   - In general, the grammar describes

   $$L_{Parity} = \{w \in \{0,1\}^* \mid \text{ the number of 1s in } w \text{ is even } \}$$

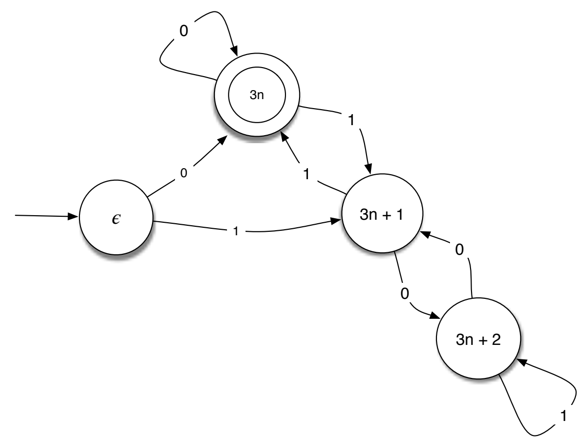
   - We encountered this language weeks ago as an early example of a regular language. Hence, we now know that context-free grammars can describe at least some regular languages.

4. Recall the language $L_{EQ} = \{1^n = 1^n \mid n \geq 0\}$. This was probably the simplest example of a language that is not regular that we discussed. Think about how one could describe this language with a context-free grammar.

   - The string "=" is in the language, so we would include the production $E \rightarrow =$.
   - If we have a string in the language, we can form another string that belongs in the language by adding a one to the front and another 1 to the end. The production $E \rightarrow 1E1$ captures this.
   - Together, these two productions describe all the strings in the language!
   - We can see, therefore that at least in some cases, context-free grammars are more expressive than DFAs or regular expressions.

5. Let's try a few more examples for practice so that you get a good sense how context-free grammars can be used to describe languages. I would like you to work in pairs on grammars for the following three languages and then I will ask for volunteers to present a grammar for each one.

   - The first example is one of my favorites. Binary strings that represent values that are multiples of 3. You may recall that the following FSA recognized this language:

   - The second is possibly the simplest of the three languages since it is quite a bit like the language we just did (i.e., it may seem like the one to go for if you are aiming to be an underachiever):

   $$L_{\text{UnaryAddition}} = \{1^a + 1^b = 1^{a+b} \mid a, b \geq 0\}$$

   - The third is a good "real" example of the use of context-free grammars to formalize the recursive description of a language.

   $$L_{\text{RE}} = \{e \mid e \text{ is a valid regular expression over } \{0,1\}\}$$

It may help to recall that:

Definition: Given some finite alphabet Σ, we define e to be a regular expression if e is

- a for some $a \in \Sigma$
- $\emptyset$
- ε
- $e_0 \cup e_1$, where e_0 and e_1 are regular expressions
- $e_0 e_1$ where e_0 and e_1 are regular expressions
- e_0^* where e_0 is a regular expression.
- (e_0) where e_0 is a regular expression.

6. The key to the second example is to recognize that any string of the form $1^n + w1^n$ where $w \in L_{EQ}$ belongs in $L_{\text{UnaryAddition}}$.

   Therefore, the grammar:

   A $\rightarrow$ 1 A 1
   A $\rightarrow$ + E
   E $\rightarrow$ 1 E 1
   E $\rightarrow$ =

   describes $L_{\text{UnaryAddition}}$.

7. The grammar for L_{RE} must include rules for the base cases of the definition of regular expressions:

   R $\rightarrow$ 0
   R $\rightarrow$ 1
   R $\rightarrow$ $\emptyset$
   R $\rightarrow$ ϵ

   together with rules for the recursive steps:

   $R \rightarrow (R)$
   $R \rightarrow RR$
   $R \rightarrow R \cup R$
   $R \rightarrow R^*$

8. Any language that can be described by a context-free grammar is called a context-free language.

9. The examples we have considered raise an interesting question. We know that there are non-regular languages that are context-free languages. We also know that there are regular languages that are context-free. What we don't know is whether there are regular languages that are not context-free.

10. The big question is whether the set of regular languages is a subset of the set of context-free languages.

11. In fact, all regular languages can be described by context-free grammars. The divisible by three example suggests two ways we might go about proving this using the approach outlined below:

    - **Proof:** Given a regular langauge L, we know that there is some DFA D such that L = L(D). Given D, we can construct a grammar G with...

      $\cdots$

      and clearly the grammar G describes the language L.

    - **Proof:** Given a regular langauge L, we know that there is some regular expression e such that L = L(e). Given e, we can construct a grammar G with ...

      $\cdots$

      and clearly the grammar G describes the language L.

12. To do this correctly, however, we need more precise definitions of what a grammar is and what language it describes.