Regular Pumping Lemma; Start CFGs

CS154

Chris Pollett

Feb. 27, 2006.

Outline

- Finish Myhill-Nerode
- Language which are not Regular
- Pumping Lemma
- Context Free Grammars

Finish Myhill Nerode

- Last day we defined two equivalence notions on strings: language indistinguishability $\equiv_L$ and machine indistinguishability $\equiv_M$.
- We said the equivalence classes of the latter are contained in the former.
- To complete the proof of the Myhill-Nerode Theorem we prove that if L is regular, there is a finite automata that has exactly the number of states as there are equivalence classes of $\equiv_L$.

Myhill Nerode Machine

- States: [x] - equivalence classes of $\equiv_L$.
- Alphabet - same as L.
- Transition function:
  ([x], a) --> [xa].
- Start state [ε]
- Final States - states [w] such that there is some w' in [w] and w' in L.

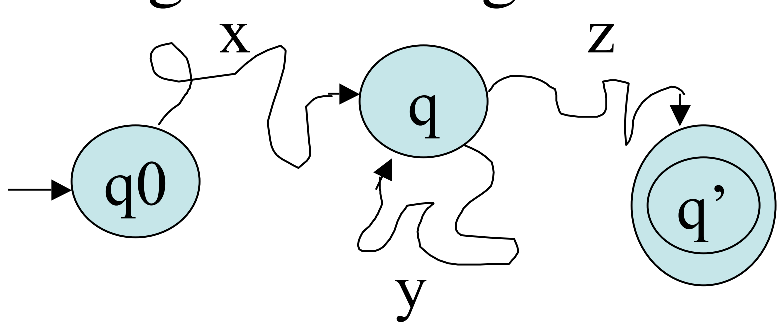
- From the last slide, we know the above machine will have the minimal number of states of any DFA recognizing L. There are actually algorithms which given a machine for L, collapse states by looking at this distinguishability notion until the minimum number of states is achieved (see Hopcraft and Ullman or Sudkamp).

Languages that are not Regular

- It turns out not all languages are regular.
- To see this consider the language L =
  $\{a^n b^n \mid n >= 0\}$
- a^j is not $\equiv_L$ to a^n unless j=n and.
- So L has infinite index so according to the Myhill Nerode theorem it is not regular.
- We will now look at another technique for proving languages not regular: the pumping lemma.

The Pumping Lemma

- Suppose we have a machine M with k states.
- Feed in some input string w of length n>k. At some point in the computation, by the Pigeonhole principle, the machine must repeat a state.
- Suppose M accepts w. Then can imagine M’s computation splitting w into 3 pieces, w=xyz, according to the diagram:



More on the Pumping Lemma

- But this implies that M accepts the strings xz, xyyz, xyyyz, etc.
- This is essentially what the Pumping Lemma says.
- More precisely **the Pumping Lemma** says:

If A is a regular language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided into three pieces s=xyz, such that:

1. for each $i >= 0$, xy^iz is in A
2. $|y| > 0$, and
3. $|xy| <= p$

Using the Pumping Lemma

- We can use the pumping lemma to show language are not regular.
- For example, let C={ w| w has an equal number of 0's and 1's}.
- Suppose DFA M recognizes C. Let p be M's pumping length and consider the string $w = 0^p 1^p$. This string is in the language and has length >p. So w = xyz, where |xy| <=p. That means $x = 0^i$ and $y=0^j$ where i+j <=p and j>0. But then, $xz = 0^i 1^p$ should be in the language. As i is not equal to p this give a contradiction. So C is not regular.

Context Free Languages

- We saw that regular languages were useful for doing things like string matching.
- This might occur in practice as the so-called lexical analysis phase of compiler. That is, the phase in which we recognize tokens like language reserved words, variable names, constants, etc.
- We now turn to ways of specify programming languages or even aspects of natural languages.
- The key to this is to have some way to recognize the underlying structures such as nouns and verbs, or control blocks, etc of the language.
- Context Free Grammars (CFGs) and their languages will provide us with the tools to do this.

Example CFG

- A grammar consists of a collection of **substitution rules** (aka **productions**). For instance:

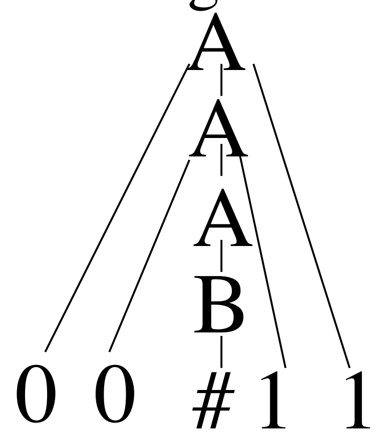
  A --> 0A1

  A --> B

  B --> #
- A rule has a two types of symbols **variables** and **terminals**.
- Usually, we'll write variables using uppercase letters or in brackets like <variable>. Terminals are supposed to be strings over the alphabet of the language we are considering.
- In a CFG, the left hand side of each rule has one variable; the right hand side can be a string of variables and terminals.
- Variables can be substituted for; terminals cannot. One variable usually denoted by S is usually distinguishes as a **start variable**.
- An example sequence of substitutions (aka a **derivation**) in the above grammar might be: A => 0A1 => 00A11 => 00B11 => 00#11

More on CFGs

- Such a derivation might also be drawn as a **parse tree**:



- The set of all strings generated by a grammar is called the **language of the grammmar**.
- A language generated by some context free grammar is called a **context free language**.
- Sometimes we abbreviate multiple rules with same LHS using a `|'. For example, A--> 0A1 | B .