JFLAP Project – Due: Friday, April 8, 2005

Prof. Nadeem Abdul Hamid
CSC 320 Spring 2005

This project involves building finite automata and Turing machines using a Java-based program, called JFLAP, written by Professor Susan Rodger of Duke University. You can download the program for free from the following web page (the download link is near the very bottom of the page):

`http://www.cs.duke.edu/~rodger/tools/jflap/`

Documentation for the program is available at this web page:

`http://www.cs.duke.edu/csed/jflap/new/DOCS/`

This software allows you to construct DFAs, NFAs, CFGs, PDAs (non-deterministic), and Turing machines, among other things. There are a number of features which we have not covered in our course but you don't need to worry about them. Once you have constructed your machine or grammar, you can simulate it on input in order to test it. Be sure to test your designs well before submitting them. To submit your work, save each machine or grammar that you design (using the "Save" command in the "File" menu), put all your files into a ZIP archive and email it to me. For each problem, save your solution in a file named as follows: "<your-last-name>-<section-num>-<part-num>.jff". For example, my solution to part b of problem 1 below would be saved in a file called "hamid-1-b.jff".

Note: Instead of the symbol ϵ as we have using in class for the empty string, JFLAP uses the symbol λ.

1 Regular Languages

a. Use JFLAP to construct a DFA that accepts all strings that have exactly one `a` and two or more `b`'s.

b. Use JFLAP to construct a six-state DFA which accepts a string if it has a number of `a`'s that is a multiple of 2 or a multiple of 3 (or both).

c. Construct a DFA which accepts a string of 0's and 1's if, when interpreted as a binary number, is a multiple of 5. (This is not easy; think about it carefully before you start building the DFA.)

d. Construct an NFA that accepts the *complement* of the language from part (a). That is, your NFA should *reject* any strings that have exactly one `a` and two or more `b`'s.

2 Context-Free Languages

Construct pushdown automata (PDAs) for the following languages using JFLAP:

a. $\Sigma = \{a, b\}, L = \{a^{3n}b^{2n} \mid n > 0\}$. For example, `aaabb` and `aaaaaabbbb` are in L.

b. $\Sigma = \{a, b\}, L = \{w \in \Sigma^* \mid w = w^R$ and w is of odd length $\}$. This is the language of palindromes of odd length.

c. $\Sigma = \{a, b\}, L = \{w \in \Sigma^* \mid$ the number of a's in w is greater than the number of b's $\}$.

Write a CFG (grammar) using JFLAP for each of the following languages.

a. $\Sigma = \{a, b, c\}, L = \{a^n b^p c^m \mid n = p + m, p > 0, m \geq 0\}$.

b. $\Sigma = \{a, b, c\}, L = \{a^n b^p c^m \mid p = n + m, n > 0, m \geq 0\}$.

3 Turing Machines

Using JFLAP to construct TM's for the following languages:

a. **Powers of two:** The input alphabet has one letter, `a`, and the language is the set of all strings w whose length is a power of two. That is,

$$\Sigma = \{a\}, L = \{w \in \Sigma^* \mid |w| = 2^n, n \geq 0\}$$

Remember, although the input alphabet only contains the single character a, you may add as many other characters to the tape alphabet as you need. Try not to get carried away, though - keep your TM as simple as possible.

b. **Adder:** Develop a TM to "perform addition." In other words, given a tape containing `111+1111=`, it should stop with a tape containing `111+1111=1111111`. Basically it recognizes the language $\{x + y = z\}$ such that $x, y, z \in 1^k$ and if $x = 1^a, y = 1^b$ then $z = 1^{a+b}$.

c. **Binary increment:** Develop a TM that increments a binary number. Given a binary number such as `10111` as input, the machine should stop with a tape containing `11000`.

   The algorithm for adding one can be described as follows: If the digit on the tape is a zero, then simply change that digit to a one. If the digit is a one, change it to a zero, move left, and apply the same procedure at that position. (This is like "carrying" a one to the next column.) Finally, to add one to a blank space, simply change that blank to a one. (This can occur when a one is carried beyond the leftmost digit of the number; the blank should be treated just like a zero.) For example, 110+1=111, 1011+1=1100, and 11+1=100.