COS 487
Fall 2003

Problem Set 5 Solutions

Problem 1:

(a.) $q_1 11$ -> $\_q_3 1$ -> $\_1q_3\_$ -> $\_1\_q_r\_$

(b.) $q_1 1\#1$ -> $\_q_3\#1$ -> $\_\#q_5 1$ -> $\_\#1q_5\_$ -> $\_\#q_7 1$ -> $\_q_7\#1$
-> $q_7\_\#1$ -> $\_q_9\#1$ -> $\_\#q_{11}1$ -> $\_q_{12}\#x$ -> $q_{12}\_\#x$
-> $\_q_{13}\#x$ -> $\_\#q_{14}x$ -> $\_\#xq_{14}$ -> $\_\#x\_q_a$

(c.) $q_1 1\#\#1$ -> $\_q_3\#\#1$ -> $\_\#q_5\#1$ -> $\_\#\#q_r 1$

(d.) $q_1 10\#11$ -> $\_q_3 0\#11$ -> $\_0q_3\#11$ -> $\_0\#q_5 11$ -> $\_0\#1q_5 1$ -> $\_0\#11q_5$
-> $\_0\#1q_7 1$ -> $\_0\# q_7 11$ -> $\_0q_7\#11$ -> $\_q_7 0\#11$ -> $q_7\_0\#11$
-> $\_q_9 0\#11$ -> $\_0q_9\#11$ -> $\_0\#q_{11}11$ -> $\_0q_{12}\#x1$ -> $\_q_{12}0\#x1$
-> $q_{12}\_0\#x1$ -> $\_q_{13}0\#x1$ -> $\_x\ q_8\#x1$ -> $\_x\#q_{10}x1$ -> $\_x\#xq_{10}1$
-> $\_x\#x1q_r\_$

(e.) $q_1 10\#10$ -> $\_q_3 0\#10$ -> $\_0q_3\#10$ -> $\_0\#q_5 10$ -> $\_0\#1q_5 0$ -> $\_0\#10q_5$
-> $\_0\#1q_7 0$ -> $\_0\#q_7 10$ -> $\_0q_7\#10$ -> $\_q_7 0\#10$ -> $q_7\_0\#10$
-> $\_\ q_9 0\#10$ -> $\_0q_9\#10$ -> $\_0\#q_{11}10$ -> $\_0q_{12}\#x0$ -> $\_q_{12}0\#x0$
-> $q_{12}\_0\#x0$ -> $\_q_{13}0\#x0$ -> $\_xq_8\#x0$ -> $\_x\#q_{10}x0$ -> $\_x\#xq_{10}0$
-> $\_x\#q_{12}xx$ -> $\_xq_{12}\#xx$ -> $\_q_{12}x\#xx$ -> $q_{12}\_x\#xx$ -> $\_q_{13}x\#xx$
-> $\_xq_{13}\#xx$ -> $\_x\#q_{14}xx$ -> $\_x\#xq_{14}x$ -> $\_x\#xxq_{14}$ -> $\_x\#xx\_q_a\_$

Problem 2:

First, note that every ordinary TM has an equivalent write-twice TM. Whenever the ordinary TM performs a write, the write-twice TM marks the current head position, inserts a separator symbol after the last used input square, and makes a copy of the last tape segment (crossing off symbols as they are copied). When the marked head is reached, the value to be written is placed on the new segment of tape and the copy continues.

A small modification to the above construction allows for the conversion of an ordinary TM to a write-once TM. Two tape squares are used for each symbol instead of one - the second square is used to mark head positions and cross out symbols.

Problem 3:

(a.) Yes - *M* accepts 0100.
(b.) No - *M* rejects 011.
(c.) Yes, if the encoding <*M*> is equivalent to the encoding <*M*,>. Otherwise the encoding is invalid and the answer is no.

(d.) No, unless the encoding *M* is also a valid encoding for a regular expression that accepts 0100.
(e.) No - *M* accepts a string (0100, for example).
(f.) Yes - the language recognized by *M* is equivalent to the language recognized by *M*.

Problem 4:

The language of strings containing an odd number of 1s is regular. Therefore, there exists a DFA *M''* which recognizes this language. We include the description of *M''* in the following Turing machine, which recognizes *A*.

M' = "On input <*M*>:
1. Construct *M'''*, a DFA which recognizes the intersection of L(*M*) and L(*M''*).
2. Check if *M'''* recognizes the empty language. If it does, accept."

Problem 5:

The language of palindromes is context-free. Therefore, there exists a PDA *P* which recognizes this language. We include the description of *P* in the following Turing machine, which recognizes *E*.

M' = "On input <*M*>:
1. Construct *P'*, a PDA which recognizes the intersection of L(*M*) and L(*P*). (See problem 2.17).
2. Convert *P'* to a CFG *C*.
3. Check if *C* recognizes the empty language. If it doesn't, accept."

Problem 6:

USELESS = {<*P*, q> | *P* is a PDA in which state q is useless}.

M = "On input <*P*, q>:
1. Construct *P'* from *P* by making q the only accepting state, and making all transitions from q lead to q.
2. Convert *P'* to a CFG *C*.
3. Check if *C* recognizes the empty language. If it does, then accept."

Problem 7:

First, a single tape Turing machine which cannot write on the input tape can simulate a DFA and can thus recognize regular languages.

For the opposite direction, suppose *M* is a Turing machine of the above type with *m* states. Consider the behavior of *M* on an input *w* of length *n*. *M* may only enter the input area

from the first square (at the beginning of the computation) and from the last square (at the end of the computation). *M* may enter the input area in any of *m* states from the the last square for a total of *m+1* ways. For each of these, *M* either accepts, rejects, loops, or leaves the input area in one of *m* states (*m+3* possibilities).

Each input string can be classified into one of these (*m+1*)^(*m+3*) categories based on *M*'s behavior on the input. *M*'s behavior on an input depends only on which category the input is in. A table mapping each category to accept or reject requires only a finite amount of memory.

Each input may be mapped to the correct category in one pass using a DFA. The DFA updates a vector of *n+1* elements. After the *i*th input symbol is read, the *j*th element of the vector indicates the state in which *M* would enter position *i+1* from the left, if started at position *i* in state *j* (or accept, reject, or loop if it fails to reach position *i+1*). The last element indicates the state in which *M* would enter position *i+1* from the left if started in its start state.

Finally, the DFA consults the table to determine if *M* accepts, rejects, or loops.