CSE 355 HOMEWORK SIX SKETCHY SOLUTIONS

NOT TO BE HANDED IN

Here are some sample questions on the material at the end of the course.

(1) Sipser 3.10. *Answered in Sipser text.*

(2) Sipser 3.14. To convert a TM to a queue automaton, the idea is that we add a new symbol # that is at the end of the nonblank portion of the tape. Then when the tape contains $w_1 \cdots w_{i-1} w_i \cdots w_n \#$ and the tape head is pointing at w_i, we represent this as queue $\dot{w_i} \cdots w_n \# w_1 \cdots w_{i-1}$. We extend the queue alphabet so that for every symbol there is a dotted and a double dotted version; the position of the TM tape head is also indicated by single dotting the symbol under it.

We remember the state of the TM (along with other things) in the state of the QA.

To simulate the TM computation, it is enough to simulate a single transition of the TM. By pulling and pushing the same symbol until we pull a singly dotted symbol, we can locate the entry under the TM tape head. If it is a # with a dot, we push a blank with a dot followed by a # without a dot and locate the dotted symbol again. Then we know what transition the TM does. We can remember the new state of the TM in the QA state. We can push the symbol written on the TM tape, but double dot it. Now we just have to move the tape head either left or right one position. The "trick" is to pull twice, remembering the last two pulls (a, b) in the state. When the next pull, say for c, is done, we push a and remember (b, c). Keep going until the second symbol remembered, b, is double dotted. If we want to move left, push a single dotted and push b undotted, and push c. If instead we want to move right, we push a then b undotted and then c dotted. Now we are ready again to find the dot and carry out the next transition.

To convert a QA to a TM, store the queue contents on the tape in reverse order. Mark the first and last nonblank symbols so that you can find them. To push a symbol, it will add a symbol just after the marker at the end of the tape (and move the marker). To pull a symbol, it gets the symbol at the left hand marker and marks the next symbol to the right as the new marker of the beginning ot the tape. (This will cause the queue contents to shift right along the tape as you push and pull, but do not worry – we have lots of tape).

(3) Sipser 3.15. For (a), solved in Sipser.
For (b), use the fact that there is a nondeterministic TM decider if and only if there is a deterministic TM decider. To make an NTM decider, nondeterministically split the string in two parts, run the decider for the first language on the first part and the second decider on the second part. If both accept, accept. Otherwise reject.
For (c) use nondeterminism to split the string into an arbitrary number of nonempty parts (for example, by placing a mark on the first character of each part). Run the decider for L on each part and if all accept, accept. Otherwise reject.
For (d) swap q_{accept} and q_{reject}.
For (e), run the deciders for both languages on the input, and accept if both do. Otherwise reject.

(4) Sipser 3.16. For (a), solved in Sipser.
For (b), use the fact that there is a nondeterministic TM recognizer if and only if there is a deterministic TM recognizer. To make an NTM recognizer, nondeterministically split

the string in two parts, run the recognizer for the first language on the first part and if it accepts then run the second recognizer on the second part. If the second accepts, accept.
For (c) use nondeterminism to split the string into an arbitrary number of nonempty parts (for example, by placing a mark on the first character of each part). Run the recognizer for L on each part and if all accept, accept.
For (d), run the recognizers for both languages on the input, and accept if both do.
For (e), we did not discuss this.

(5) Sipser 4.11. Convert the PDA to a CFG in Chomsky normal form (we have algorithms to do this). Next determine which variables have the property that they derive at least one string that contains only terminals (we saw this in class). Delete all variables that do not make strings of terminals. Now determine which variables can be produced in a derivation from the start variable S (the idea is the same as we just used). Delete all variables than cannot be in a derivation starting from S. Now if any variable A has the property that it derives, in one or more steps, a string of variables and terminals that contains A on the RHS, answer "yes, the language is infinite." If no such variable exists, answer "no, the language is not infinite."

(6) Sipser 4.14. *Answered in Sipser text.*

(7) Sipser 4.21. One way to do this is to construct from the DFA M another DFA R so that $w \in L(M)$ if and only if $w^{\mathcal{R}} \in L(R)$. This is closure of the regular languages under reversal, which can be shown (for example) by reversing a regular expression for the language in a straightforward way. Now we use the product construction to make a DFA T whose language is $L(M) \cup L(R)$. And now we present $\langle M, T \rangle$ to EQ_{DFA}, which we know is decidable. If the answer is yes, then the reverse of every string in $L(M)$ must also be in $L(M)$. If the answer is no, the reversal of some string in $L(M)$ is not in $L(M)$. So the answer from EQ_{DFA} is the answer to the question.

(8) Sipser 4.24. Let $U_{PDA} = \{\langle P \rangle : P \text{ is a PDA that has useless states}\}$. By discussions in class, we know that the emptiness problem E_{PDA} is decidable. Let D be a decider for it. To decide U_{PDA}, use the following:

```
$T$ = "On input $\langle P \rangle$ where $P$ is a PDA:
  1. For each $q \in Q$:
       a. Modify $P$ such that $F = \{q\}$.
       b. Run $D$ (decider for $E_{PDA}$) on input $\langle P \rangle$. If $D$ accepts, *accept* $\langle P \rangle$.
  2. *Reject* $\langle P \rangle$."
```

Because E_{PDA} is decidable, U_{PDA} is decidable.

(9) Sipser 5.9. This is a nontrivial property of the language of a TM, because some TMs have languages with this property (e.g., $\Sigma^\star$) and some do not (e.g. $\{ab\}$). So Rice's theorem tells us that this is undecidable.

(10) Sipser 5.12. Let M be a TM. Modify M so that whenever it writes a blank, we write a newblank instead. Transitions on newblank are copies of those reading blanks. Now change all occurrences of q_{accept} to q_{weird}, a new state. Add transitions from q_{weird} that write blanks (not newblanks) and move left. Call the new machine M'. Then M accepts its input if and only if M' writes a blank symbol on top of a nonblank one. This reduces A_{TM} to the problem asked.

(11) Sipser 5.28. *Answered in Sipser text.*