AUTO
SOLUTIONS

Comprehensive Exam: Autumn 2000-01

Automata and Formal Languages (60 points)
Sample Solutions

Problem 1. [10 points]
Consider the language L defined by the regular expression 00^*10. Provide a PDA M for this language using *as few states as possible.* Note that there is a PDA with only 1 state and that the number of points you get will depend on the number of states used in your solution.

Solution:
The following PDA with only 1 state accepts the language $L(00^*10)$ by empty stack. The PDA has the following components: $Q = \{q\}$, $\Sigma = \{0,1\}$, $\Gamma = \{Z_0, X, Y\}$, $q_0 = q$, and $F = \{\}$. The transition function is as follows:

$$\begin{aligned}\delta(q,0,Z_0) &= \{(q,X)\}\\ \delta(q,0,X) &= \{(q,X)\}\\ \delta(q,1,X) &= \{(q,Y)\}\\ \delta(q,0,Y) &= \{(q,\epsilon)\}\end{aligned}$$

Problem 2. [18 points]
Decide whether the following statements are TRUE or FALSE. *You will receive 3 points for each correct answer and -2 points for each incorrect answer.*

1. If L_1 and L_2 are both non-regular, then $L_1 \cap L_2$ must be non-regular.
2. $L = \{w \in \{a,b,c\}^* \mid w$ does not contain an equal number of occurrences of a, b, and $c\}$ is context-free.
3. Let L represent the language of a non-deterministic finite-state automaton N; then, swapping the final and non-final states of N gives a machine N' whose language is the complement of L.
4. Assume that P $\neq$ NP. If L_1 is in P and L_2 is in NP, then $L_1 \cap L_2$ must be in P.
5. If L_1 and L_2 are both in NP, then $L_1.L_2$ must be in NP.
6. If L_1 is context-free and L_2 is NP-complete, then $L_1 \cup L_2$ must be NP-complete.

Solution:

1. False

3. False

4. False

5. True

6. False

Problem 3. [12 points]
Classify each of the following languages as being in one of the following classes of languages: *empty*, *finite*, *regular*, *context-free*, *recursive*, *recursively enumerable*. You must give the *smallest* class that contains *every possible language* fitting the following definitions. For example, the language of a DFA M could be *empty* or *finite*, and must always be *context-free*, but the smallest class that is appropriate is *regular*.

1. The language $L = \{a^i b^j c^k d^l \mid i = k \text{ and } j = l\}$.
2. The set of strings from $\{0,1\}^*$ which, when viewed as integers written in binary, are divisible by 3.
3. The language of a non-deterministic finite state automaton (NFA) with only two states.
4. The language of a non-deterministic push-down automaton (NPDA) with only one state.
5. The complement of a language L that belongs to P (polynomial time) but is not context-free.
6. A language L to which we can give a polynomial-time reduction from an undecidable language.

Solution:

1. Recursive
2. Regular
3. Regular
4. Context-free
5. Recursive
6. Recursively-enumerable

Problem 4. [10 points]
Using a reduction from a known undecidable problem, prove that the following problem is undecidable: Determine whether a Turing machine M halts on all inputs from $\{0,1\}^*$ that represent a valid encoding of some Turing machine. (You may assume any standard scheme for encoding a Turing machine into a string of 0's and 1's.)

Solution:

Let $L_M = \{< M >|\ M$ halts on any input w which is a valid encoding of a Turing machine$\}$.

The halting problem is undecidable: Determine whether a Turing machine M halts on a particular input w. We define the language $L_H = \{< M, w >|\ M$ halts on $w\}$ to represent the halting problem.

We will give a reduction from L_H to L_M, thereby establishing the undecidability of L_M. Given an instance of the halting problem, say M and w, the reduction constructs a new Turing machine M'. The machine M' ignores its input and simulates M on input w. Clearly, if M halts on w, then M' halts on all w' which are valid encodings of Turing machines (in fact, it halts on all inputs w'). Furthermore, if M does not halt on w, then M' does not halt on any w'. It follows that $< M, w > \in L_H$ if and only if $M' \in L_M$, establishing the validity of the reduction. Since the reduction is easy to compute, it follows that L_M is undecidable.

Problem 5. [10 points]
Recall the decision problems called 2-SAT and 3-SAT. These are the versions of the satisfiability problems for 2-CNF and 3-CNF boolean formulas, respectively.

a). Prove that 2-SAT is polynomial-time reducible to 3-SAT. (Describe a reduction and justify its correctness.)

b). Given that 3-SAT is NP-complete, is the result in part (a) sufficient to prove the NP-completeness of 2-SAT? Explain.

Solution:

a). We describe a polynomial-time reduction from 2-SAT to 3-SAT. Given a 2-SAT formula $F(X_1, \ldots, X_n)$, the reductions a 3-SAT formula $G(X_1, \ldots, X_n, Z, A, B)$ as follows.

We create 3 new variables Z, A, and B. For each clause $X_i \cup X_j$ in F, we create a clause $X_i \cup X_j \cup \overline{Z}$ in G. Also, we add to G four additional clauses: $Z \cup A \cup B$, $Z \cup \overline{A} \cup B$, $Z \cup A \cup \overline{B}$, and $Z \cup \overline{A} \cup \overline{B}$. Quite clearly, the reduction can be computed in polynomial time. We now establish the validity of the reduction by showing that F has a satisfying truth assignment if and only if G has a satisfying truth assignment.

If F has a satisfying truth assignment, we can get a satisfying truth assignment for G as follows: use the same truth values for $X_1, \ldots, X_n$ and Z, A, B to TRUE don't care about A and B). It is easy to verify that G is satisified by this truth assignment.

If G has a satisfying truth assignment, then Z must be set to TRUE; otherwise, there is no way to satisfy the four additional clauses. It follows that the same truth assignment, restricted to $X_1, \ldots, X_n$, is a satisfying truth assignment for F.

b). No, this is not sufficient to prove the NP-completeness of 2-SAT. A reduction from 3-SAT to 2-SAT would have implied the NP-hardness of 2-SAT, but this reduction is in the reverse direction. In fact, 2-SAT can be solved in polynomial time and hence is unlikely to be NP-complete.