Gradiance Online Accelerated Learning

Submission number:	83275
Submission certificate:	IF521152
Submission time:	2014-05-08 00:33:30 PST (GMT - 8:00)

Number of questions:	7
Positive points per question:	3.0
Negative points per question:	1.0
Your score:	0

Based on Chapter 9 of HMU.

1. Suppose a problem P_1 reduces to a problem P_2. Which of the following statements can we conclude to be TRUE based on the above?

a) If P_2 is undecidable, then it must be that P_1 is decidable.

b) If P_1 is undecidable, then it must be that P_2 is decidable.

c) If P_1 is RE, then it must be that P_2 is RE.

d) If P_2 is decidable, then it must be that P_1 is decidable.

Answer submitted: **c)**

Your answer is incorrect.

Hint: Let P1 = { } and P2 be the language L_e defined in Section 9.3.2. on p. 394. You should examine Section 9.3.1 (p. 392) for a discussion of what is implied by the existance of a reduction from one problem to another.

Question Explanation:

See Theorem 9.7 on p. 393. If there is a reduction from P_1 to P_2, then P_2 must be at least as hard as P_1 (and may be harder). Moreover a solution to P_2 combined with the reduction from P_1 to P_2, implies a solution to P_1.

If P_1 is decidable, then P_2 need not be decidable. As a counterexample, Let P_1 = { }, denote the empty language. P_1 is decidable. For any string *w*, the correct answer to the question "is *w* in P_1?" is "no". Let M denote the Turing machine whose language is P_1. Let P_2 be the undecidable problem L_{ne} defined in Section 9.3.2. on p. 394. We can construct a reduction from P_1 to P_2 as follows: Given an instance *w* of P_1, we ask if M is in L_{ne}. Since L(M) is empty, the answer is always "no".

Similarly, we can argue that if P_2 is undecidable, then P_1 need not be undecidable.

If P_1 is RE, then P_2 need not be RE. As a counterexample, Let P_1 = { }, denote the empty language. P_1 is decidable. For any string *w*, the correct answer to the question "is *w* in P_1?" is "no". Let M denote some Turing machine whose language is nonempty. Let P_2 be the non-RE problem L_e defined in Section 9.3.2. on p. 394. We can construct a reduction from P_1 to P_2 as follows: Given an instance *w* of P_1, we ask if M is in L_e. Since L (M) is nonempty, the answer is always "no".

Similarly, we can argue that if P_2 non-RE, then P_1 need not be non-RE.

The correct choice is: **d)**

---

2. Here is an instance of the Modified Post's Correspondence Problem:

	List A	List B
1	01	010
2	11	110
3	0	01

If we apply the reduction of MPCP to PCP described in Section 9.4.2 (p. 404), which of the following would be a pair in the resulting PCP instance.

a) (0*1*, *0*1*0)
b) (0*1*, *0*1*0*)
c) (0*1, *0*1*0)
d) (1*1, *1*1*0)

Answer submitted: **b)**

Your answer is incorrect.

Check the proper way to "begin" the simulation of the MPCP instance by the constructed PCP instance. Remember, lists A and B are treated differently. Post's Correspondence Problem is treated in Section 9.4 (p. 401), and in particular you should read Section 9.4.2 (p. 404) on the "modified" PCP.

Question Explanation:

The PCP instance is:

	List C	List D
0	*0*1*	*0*1*0
1	0*1*	*0*1*0
2	1*1*	*1*1*0
3	0*	*0*1
4	$	*$

The correct choice is: **a)**

---

3. We wish to perform the reduction of acceptance by a Turing machine to MPCP, as described in Section 9.4.3 (p. 407). We assume the TM *M* satisfies Theorem 8.12 (p. 346): it never moves left from its initial position and never writes a blank. We know the following:

1. The start state of *M* is *q*.
2. *r* is the accepting state of *M*.
3. The tape symbols of *M* are 0, 1, and B (blank).
4. One of the moves of *M* is δ(q,0) = (p,1,L).

Which of the following is DEFINITELY NOT one of the pairs in the MPCP instance that we construct for the TM *M* and the input 001?

a) (0q0, p01)

b) (r##, #)

c) (p##, #)

d) (0r1, r)

Answer submitted: **c)**

You have answered the question correctly.

Question Explanation:

We know we need the following pairs:

1. (#, #q001#) --- the starting pair (rule 1).
2. (0,0), (1,1), and (#,#) by rule (2).
3. (0q0, p01) and (1q0, p11) by rule (3).
4. All pairs by rule (4) of the form (xry, r), (xr, r), and (ry, r), where *x* and *y* are each either 0 or 1.
5. (r##, #) by rule (5).

4. For the purpose of this question, we assume that all languages are over input alphabet {0,1}. Also, we assume that a Turing machine can have any fixed number of tapes.

Sometimes restricting what a Turing machine can do does not affect the class of languages that can be recognized --- the restricted Turing machines can still be designed to accept any recursively enumerable language. Other restrictions limit what languages the Turing machine can accept. For example, it might limit the languages to some subset of the recursive languages, which we know is smaller than the recursively enumerable languages. Here are some of the possible restrictions:

1. Limit the number of states the TM may have.
2. Limit the number of tape symbols the TM may have.
3. Limit the number of times any tape cell may change.
4. Limit the amount of tape the TM may use.
5. Limit the number of moves the TM may make.
6. Limit the way the tape heads may move.

Consider the effect of limitations of these types, perhaps in pairs. Then, from the list below, identify the combination of restrictions that allows the restricted form of Turing machine to accept all recursively enumerable languages.

a) Allow the TM to run for only 2^n moves when the input is of length *n*.

b) Limit the number of states to 1,000,000 and the number of tape symbols to 1,000,000.

c) Allow tape heads to move only right or remain stationary on their tape.

d) Allow the TM to run for only n^{10} moves when the input is of length *n*.

Answer submitted: **c)**

Your answer is incorrect.

Notice that if no tape head can ever move left, then the TM cannot read anything it has written. Can you compare this sort of TM to a finite automaton, in particular to a deterministic finite automata with epsilon-transitions (Section 2.5, p. 72)?

Question Explanation:

The key observation is that given any TM *M*, we can design a very restricted form of TM that simulates *M* by writing successive ID's on a tape. The simulating TM can run back and fourth on the tape, computing the symbols of the next ID, one at a time. Remember that, unless the symbol being copied is adjacent to the head, the symbol cannot change.

On the other hand, if we limit the amount of tape that the TM may use to any computable function of the input length, then we can accept only recursive languages. The reason is that after a while, the TM must repeat an ID, and if it hasn't accepted by then, we can conclude it never will. Likewise, if we limit the number of moves to any computable function, we can accept only recursive languages.

Finally, if we limit the tape heads to move in only one direction, then nothing it writes can ever affect what it does. Thus, the TM can be simulated by a finite automaton, no matter how many tapes the TM has.

The correct choice is: **b)**

---

5. In this question, L_1, L_2, L_3, L_4 refer to languages and M, M_1, M_2 refer to Turing machines. Let
L_1 = {(M_1,M_2) | L(M_1) is a subset of L(M_2)},
L_2 = {M | There exists an input on which TM M halts within 100 steps},
L_3 = {M | There exists an input *w* of size less than 100, such that M accepts *w*},
L_4 = {M | L(M) contains at least 2 strings}.

Decide whether each of L_1, L_2, L_3 and L_4 are recursive, RE or neither. Then identify the true statement below.

a) The complement of L_3 is recursively enumerable.

b) L_4 is not recursive but cannot be proved so by Rice's Theorem.

c) The complement of L_2 is not recursively enumerable.

d) The complement of L_4 is not recursive.

Answer submitted: **b)**

Your answer is incorrect.

Hint: The property concerned is a nontrivial property of RE languages. The statement and proof of Rice's Theorem is in Section 9.3.3 (p. 397).

Question Explanation:

This question is based on concepts covered in Sections 9.2 and 9.3 (p.

383--401) of the text.

L_1 is not RE. To prove this statement we can reduce the non-RE language L_e (Section 9.3.2., p. 394) to L_1. Let M be the Turing machine for the empty language L(M). Then, given an instance M_1 of L_e, we can reduce it to the instance (M_1,M) of L_1. M_1 is in L_e if and only if L(M_1) is a subset of L(M).

L_2 is recursive. A TM for L_2 runs M on inputs of size no greater than 100 for upto 100 steps. If M halts on any such input, then accept, else reject. Note that it is sufficient to consider inputs of size no greater than 100 since inputs of length greater than 100 can never have more than their first 100 symbols looked at in 100 steps.

L_3 is RE but not recursive. To show that L_3 is RE, consider a Turing machine for L_3 would run M on all input strings of length less than 100 in an interleaved manner. If any such execution halts, M halts. Now, to prove that L_3 is not recursive, consider the set C = {L | L is RE and L has a string of size less than 100}. C is a proper non-empty subset of RE languages. Hence by Rice's theorem, L_3 is not recursive.

L_4 is not recursive. Consider the set S = {L | L is RE and L contains at least two strings}. S is a proper non-empty subset of RE languages. Hence by Rice's theorem, L_4 is not recursive.

The complement of a recursive language is recursive. If a language L and its complement are both RE then L is recursive.

The correct choice is: **d)**

---

6. Which of the following problems about a Turing Machine *M* does Rice's Theorem imply is undecidable?

a) Does the language of *M* contain at least 10 strings?

b) Does *M* ever write the symbol 1 on its tape?

c) Is the language of *M* not equal to itself?

d) Does *M* ever write the symbol 0 on its tape?

Answer submitted: **c)**

Your answer is incorrect.

This property of languages is trivial --- it is true of no language. Thus, Rice's Theorem does not imply it is undecidable, and in fact it is a decidable problem. See the discussion of Rice's Theorem in Section 9.3.3 (p. 397).

Question Explanation:

Rice's theorem applies to problems that are about the language of a Turing machine. It says any question about Turing machines of the form "does the language of this TM have the property X?" is undecidable unless X is *trivial* --- either no language has the property or all recursively enumerable languages have this property.

The correct choice is: **a)**

---

7. We can represent questions about context-free languages and regular languages by choosing a standard encoding for context-free grammars (CFG's) and another for regular expressions (RE's), and phrasing the question as recognition of the codes for grammars and/or regular expressions such that their languages have certain properties. Some sets of codes are decidable, while others are not.

In what follows, you may assume that G and H are context-free grammars with terminal alphabet {0,1}, and R is a regular expression using symbols 0 and 1 only. You may assume that the problem "Is L(G) = (0+1)*?", that is, the problem of recognizing all and only the codes for CFG's G whose language is all strings of 0's and 1's, is undecidable.

There are certain other problems about CFG's and RE's that are decidable, using well-known algorithms. For example, we can test if L(G) is empty by finding the pumping-lemma constant n for G, and checking whether or not there is a string of length n or less in L(G). It is not possible that the shortest string in L(G) is longer than n, because the pumping lemma lets us remove at least one symbol from a string that long and find a shorter string in L(G).

You should try to determine which of the following problems are decidable, and which are undecidable:

- Is Comp(L(G)) equal to (0+1)*? [Comp(L) is the complement of language L with respect to the alphabet {0,1}.]
- Is Comp(L(G)) empty?
- Is L(G) intersect L(H) equal to (0+1)*?
- Is L(G) union L(H) equal to (0+1)*?
- Is L(G) finite?
- Is L(G) contained in L(H)?
- Is L(G) = L(H)?
- Is L(G) = L(R)?
- Is L(G) contained in L(R)?
- Is L(R) contained in L(G)?

Then, identify the true statement from the list below:

a) "Is L(G) = L(H)?" is decidable.

b) "Is L(H) contained in L(G)?" is decidable.

c) "Is L(G) contained in L(R)?" is decidable.

d) "Is L(R) contained in L(G)?" is decidable.

Answer submitted: **d)**

Your answer is incorrect.

Hint: Reduce "L(G) = (0+1)*" to this problem by choosing R to be the regular expression (0+1)*. Section 9.5.3 (p. 415), especially Theorem 9.22 (p. 416), gives a number of undecidable problems about grammars.

Question Explanation:

Here is the explanation for each of the problems:

- Is Comp(L(G)) equal to (0+1)*? This problem is the same as asking if L(G) is empty. It is therefore decidable.

- Is Comp(L(G)) empty? This problem asks if L(G) = (0+1)*; it is undecidable according to the facts given in the problem statement.

- Is L(G) intersect L(H) equal to (0+1)*? Undecidable. Proof: if we could decide this problem, let H be a CFG that generates (0+1)*, specifically S->0S|1S|ε. If L(G) intersect L(H) = (0+1)*, then L(G) = (0+1)*. That is, we could decide if L(G) = (0+1)*. But we are given that we can't decide that question.

- Is L(G) union L(H) equal to (0+1)*? Let H be a CFG that generates an empty language, specifically, S->S. Then L(G) union L(H) = (0+1)* if and only if L(G) = (0+1)*, and the argument proceeds as for the previous problem.
- Is L(G) finite? Let n be the pumping-lemma constant for G. If there is a string of length between n and 2n, then we can pump this string to generate an infinite number of strings. But if there is an infinite number of strings, then there must be one whose length is between n and 2n. If not, let z be the shortest string of length greater than 2n; there must be one if there are an infinite number of strings. The pumping lemma must let us remove between 1 and n characters of z and get another string in L(G). But this string must be between n and 2n in length. It can't be shorter than n, because z is longer than 2n. It can't be longer than 2n, because z is the shortest such string. Note that this test tells us both whether L(G) is finite and whether it is infinite.
- Is L(G) contained in L(H)? Suppose we could decide this question. Choose G so that L(G) = (0+1)*. Then the condition is true if and only if L(H) = (0+1)*, which we know is undecidable.
- Is L(G) = L(H)? The same argument applies if we choose H such that L(H) = (0+1)*.
- Is L(G) = L(R)? Again, the same argument applies if we let R be (0+1)*.
- Is L(G) contained in L(R)? Construct from G a CFG G', whose language is L(G) intersect Comp(L(R)). We can make this construction because there is an algorithm to find a representation (RE, DFA, or NFA) of the complement of a regular language, given a representation for the language itself, and also because there is a construction to find a CFG for the intersection of the languages of a CFG and RE (convert to a PDA and a DFA, respectively, and simulate the two in parallel; then convert the resulting PDA to a CFG). Now, L(G) is contained in L(R) if and only if L(G') is empty. We can test emptiness of a CFG, as described in the statement of the question.
- Is L(R) contained in L(G)? Choose R = (0+1)*. Then L(R) is contained in L(G) if and only if L(G) = (0+1)*, which we know is undecidable.

The correct choice is: **c)**