Assignment 4

Due Friday April 7th, 2016

1. [10 marks] Short answers about Turing machines

Answer each of the following questions in *one or two* sentences.

(a) [2 marks] Explain the difference between a Turing recognizable and a Turing decidable languages.
Answer: A language L is Turing recognizable if some TM *recognizes* it, *i.e.,* accepts a string s if and only if $s \in L$. A language is Turing decidable if some TM *decides* it, *i.e.,* accepts a string s if and only if $s \in L$, and rejects s if and only if $s \notin L$.

(b) [2 marks] Explain the Halting problem and its significance.

Answer: The Halting problem is the problem of deciding whether a given Turing machine M will halt on input string s. Turing proved that the Halting problem is undecidable, meaning there exists no algorithmic way to decide if a program will run forever, or not.

(c) [2 marks] Consider the set of languages recognizable by a determnistic Turing Machine. Now consider a multi-tape Turing machine, that is, a Turing machine with multiple taples. Could a multi-tape Turing machine possibly recognize more languages than a single-tape Turing Machine? Why or why not?

Answer: No, according to the Church-Turing thesis. It states that a function is algorithmically computable if and only if it is computable by a ('normal', i.e., single-tape) Turing machine. Further it can be shown that a single-tape machine can simulate any multi-tape machine.

(d) [2 marks] Under what circumstance does a non-deterministic Turing machine output `accept`?
Answer: A NDTM outputs accept when *some* branch of the computation accepts.

(e) [2 marks] In class we mentioned that the set of all possible Turing Machines is countably infinite, and that the set of all languages is uncountably infinite. Use these facts to reason why there *must* be languages that cannot be recognized by a Turing machine.
Answer: In simple terms it means there are more languages than Turing machine i.e., there are more languages than there are Turing machines to recognize them, therefore some languages must not be recognizable.

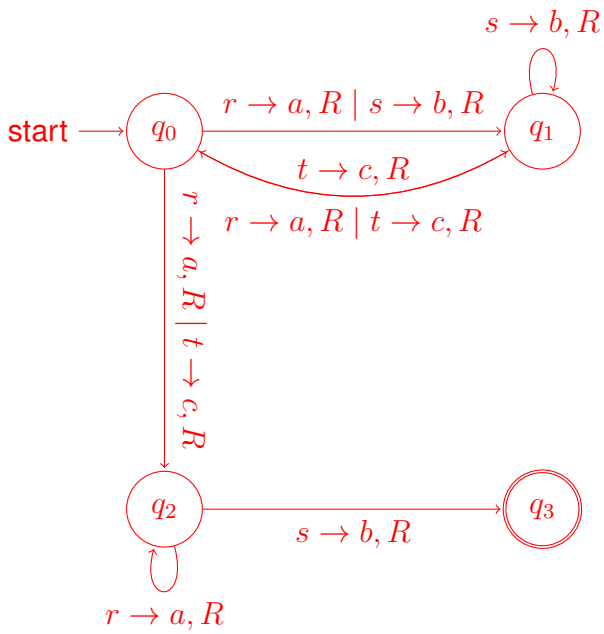
2. [10 marks] Computation tree of a non-deterministic Turing Machine

The following transitions implicitly define a non-deterministic Turing machine (with the alphabet $\Sigma = \{r, s, t\}$):

$\delta(q_0, r) = (q_1, a, R)$
$\delta(q_0, r) = (q_2, a, R)$
$\delta(q_0, s) = (q_1, b, R)$
$\delta(q_0, t) = (q_2, c, R)$
$\delta(q_1, r) = (q_0, a, R)$
$\delta(q_1, s) = (q_1, b, R)$
$\delta(q_1, t) = (q_0, c, R)$
$\delta(q_2, r) = (q_2, a, R)$
$\delta(q_2, s) = (q_3, b, R)$
$\delta(q_2, t) = (q_1, c, R)$

where q_0 is the start state and q_3 is the accepting state.

(a) [5 marks] Create a state diagram to represent the Turing machine.
Answer: See the state diagram below:
Answer:



(b) [5 marks] Given input `rstrsr` , list the final tape contents for each computational path, and indicate which paths accept.
Answer: The Turing machine will halt for three possibilities. The ending tape contents

(excluding blanks) will be 'abtrsr', 'abcabr' and 'abcaba', where the paths 'abtrsr' and 'abcabr' correspond to accepting paths, so the initial input string is accepted.

3. [6 marks] Complexity Classes

For each of the complexity classes covered in the lectures, i.e., **P**, **NP**, **NP**-complete, **NP**-hard, **BPP** and **BQP**, state whether the problems below are known to be in that class or not. For each positive answer (i.e., *yes, problem* ____ *is in class* ____.), give the name of an algorithm from that class that solves it:

(a) [3 marks] **PRIMES**, the problem of determining if a given integer is a prime number.
Answer: PRIMES: In P as per AKS test. In BPP as per Miller-Rabin. Also in NP and BQP since in P.

(b) [3 marks] **FACTOR**, the problem of determining if an integer contains a factor less than some bound b.
Answer: FACTOR: In NP because you can use multiplication to verify positive answers. and in BQP as per Shor's algorithm.

4. [4 marks] Boolean Satisfiability

(a) [2 marks] **Boolean Satisfiability (SAT)**. Consider the following boolean equation:

$$((x_1 \wedge x_2) \vee (x_3 \wedge \neg x_4)) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$$

Is this equation satisfiable? If so, assign the variables to satisfy the equation. If not, explain why not.
Answer: There is an answer which would be X1 false, X2 true, X3 true, X4 false

(b) [2 marks] Recall **SAT** was the first problem proven to be **NP**-complete (by Cook and Levin). What would the consequence be of finding an efficient algorithm to solve **SAT**?
Answer: You will have proven P=NP.

5. [10 marks] NP-Completeness

Let S be a set of integers. Recall SUBSETSUM is the problem of deciding whether there is some subset of S which sums to zero. Now consider the related problem PARTITION which asks whether S can be partitioned (i.e., split) into two subsets such that they have the *same* sum. For example, $\{1, 2, 3, 4\}$ contains such a partition: $\{1, 4\}$ and $\{2, 3\}$.

Given that SUBSETSUM is **NP**-complete, prove PARTITION is **NP**-complete. **Hint**: Suppose the elements of a set S sum to some integer s. Define a new set $S' = S \cup \{-s\}$ and use

it in your proof.

SOLUTION. To show PARTITION is NP-complete we must show (a) it is in NP and (b) all problems in NP have a polynomial time reduction to it.

First, to show PARTITION is in NP we observe two facts: (1) it is a decision problem, i.e., yes or no: can the set can be partitioned into two subsets of equal sum? and (2) it is efficiently verifiable: a certificate can be checked by (i) summing each of the two subsets and checking the sums are equal and (ii) checking the two subsets are a valid partition of the original set. Summation and comparing sets both have polynomial running time in the size of the input set.

Second, to show all languages are poly-time reducible to PARTITION, it would be sufficient to show that SUBSETSUM (a known NP-complete problem) is poly-time reducible to PARTITION. That is, we can solve SUBSETSUM problems in polynomial time, *plus* whatever time it takes to solve a PARTITION problem.

Suppose there's a function PART-solver(s) which accepts a list and returns true if it contains a valid partition, and false otherwise. Using the hint above we can create a subsetsum solver as follows:

```
SUB-solver(S):
s = Sum(S)
$S' = S \cup {-s}$
return PART-solver(s')
```

Case 1: Suppose a valid partition U, V of S' exists. Let $\Sigma U = a$. Then by our definition of a valid partition being two subsets of equal sum, we have $\Sigma V = a$. Next notice the sum of S' is $s + (-s) = 0$. Thus the sum of U and V is 0. Therefore $a + a = 0$. Therefore $a = 0$. Without loss of generality assume the $-s$ element is contained in U. That means V only contains elements of S. And we have $\Sigma V = a = 0$. Therefore S contains a subset that sums to 0 and SUB-solver would return True.

Case 2: There were no valid partitions, meaning for all disjoint subsets U, V of S', we have $\Sigma U = a$ and $\Sigma V = b$ for $a \neq b$. (This has to be the case, because if $a = b$, U, V would be a valid partition, and this is the case where it's *not* a valid partition). Recall however the sum of S' is 0. So the only way to satisfy $a + b = 0$ is for $a \neq b$ is for a, b both to be non-zero. Therefore S does not contain a subset that sums to 0, therefore SUB-solver would return false.

SUB-solver returns true only if S has a subset that sums to 0 and false otherwise. SUB-solver clearly runs in polynomial time excluding the time PART-solver takes: SUB-solver would need to compute the sum of S, and create a new set S' consisting of S and one additional element (i.e., $-s$).

6. [10 marks] Undecidability

Recall the Halting problem:

$$\text{HALT} = \{\langle M, w\rangle : M \text{ halts on string } w\}.$$

We already know this problem is undecidable, but maybe it's still possible to build a decider for a simpler problem, like deciding if a machine M will accept a *specific* input, e.g., `'a'`. As we will find out, this language is undecidable *as well*. Let:

$$\text{L} = \{\langle M\rangle : M \text{ accepts the string 'a'}\}.$$

Prove L is undecidable.

Hint: Toward a contradiction assume there exists a Turing machine C that decides L. It accepts the description of a Turing machine M as input and returns `True` if the machine accepts `a`, and returns `False` otherwise. Now consider the following machine:

```
A(<M,w>):
    Construct a new Turing machine B with the following behavior:
        B(x):
            Simulate M on w
            Halt and output accept

    return <B>  //Return a string describing Turing machine B.
```

Now write an algorithm `Halt-solver` that accepts a string `<M,w>` and outputs accepts if `M` halts on input `w` and outputs rejects otherwise. By making calls to A and C, show how `Halt-solver` can decide the halting problem.

SOLUTION: Goal: We need to show L is undecidable. Strategy: Show that if a decider for L existed, it could be leveraged to decide the halting problem. Since the halting problem is known to be undecidable, a contradiction would arise, meaning L must not have been decidable to begin with.

Towards the contradiction assume there exists a Turing machine C that decides L. It accepts the description of a Turing machine as input and returns True if the machine accepts `'a'`, and returns False otherwise.

Let us define an algorithm `Halt-solver` to decide the halting problem:

```
Halt-solver(<M,w>):
   <B> = A(<M,w>)
   return C(<B>)
```

As a decider for the halting problem, Halt-solver should have the following two possible outcomes: If M halts on w then Halt-solver should halt and accept. If M loops on w then Halt-solver should halt and reject.

Let's consider the first case: If M halts on w, then B will halt and accept because B simulates M on w and halts only if M halts. Now recall C is a decider for L, so if you pass B into C we're asking C to decide if B will accept the string `'a'` as input. In this case M halts on w, so B will halt. Notice B didn't make use of the input `'a'`. But it halted and accepted, so by definition it accepts `'a'`. Since B accepted `'a'`, C will also halt and accept. Halt-solver returns the outcome of C run on B as input, meaning Halt-solver will also halt and accept. Now observe Halt-solver accepted in the case where M halts on w.

Now let's consider the other case: Similar as above. If M loops on w, then B will loop because B simulates M on w and if M loops, B will never make it to the line where it halts. Since B loops, passing it to C will cause C to conclude B does not accept `'a'` as input, and thus C will halt and reject. Therefore Halt-solver will halt and reject. Again observe Halt-solver rejected in the case where if M loops on w.

Thus Halt-solver accepts if M halts on w, and rejects if M loops on w. Therefore Halt-solver decides the halting problem. But the Halting problem is undecidable. This is a contradiction. Since the contradiction is reached by assuming C exists, we conclude that C cannot exist and thus L is undecidable.