EXAM 4 SOLUTIONS - CSCI 341

Friday, November 30

George Voutsadakis

Read each problem **very carefully** before starting to solve it. Each problem is worth 10 points. It is necessary to show **all** your work. Correct answers without explanations are worth 0 points. GOOD LUCK!!

1. (a) Give the definition of a **context-free language**.

   A language is **context-free** if it can be generated by a context-free grammar.

   (b) True or false?

   (i) $L = \{a^n b^n c^n : n \in \mathbb{N}\}$ is context free. False

   (ii) $M = \{a^n b^n c^k : n, k \in \mathbb{N}\}$ is context-free. True
   Proof of (ii):

   The following context-free grammar generates M:

   $$S \to Sc|B$$
   $$B \to aBb|\Lambda$$

   (c) The union of two context free languages is context-free. True
   Proof:

   Suppose L_1 and L_2 are context-free languages. Then L_1 is generated by a context free grammar G_1, with start symbol S_1, and L_2 is generated by a context-free grammar G_2, with start symbol S_2 (G_1 and G_2 can be taken to have disjoint sets of non-terminal symbols). Therefore, $L_1 \cup L_2$ is generated by the context-free grammar G with start symbol S and productions $S \to S_1|S_2$ together with all those productions in G_1 and those in G_2. We conclude that $L_1 \cup L_2$ is also context-free.

   (d) The intersection of two context-free languages is context-free. False
   Proof:

   $L_1 = \{a^n b^n c^k : n, k \in \mathbb{N}\}$ is context-free, as is $L_2 = \{a^k b^n c^n : n, k \in \mathbb{N}\}$. But their intersection $L_1 \cap L_2 = \{a^n b^n c^n : n \in \mathbb{N}\}$ is not context-free.

   (e) The complement of a context-free language is context-free. False
   Proof:

   If it was then, since the union of two context-free languages is context-free, then the intersection would also be context-free:

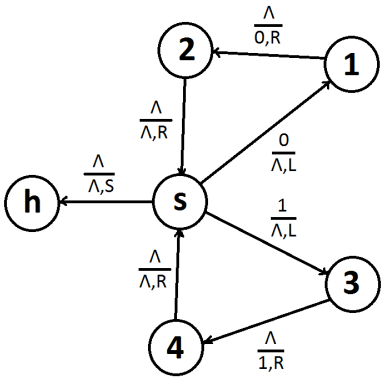
   $$L_1 \cap L_2 = (L_1' \cup L_2')'.$$

2. (a) Give a list of the six components in the formal definition of a **Turing Machine** and a short description of each.
   - A is the input alphabet;
   - Γ is the tape alphabet;
   - Q is a finite nonempty set of states;
   - q_0 is the initial state;
   - h is the final or halt state;
   - δ is the transition relation.

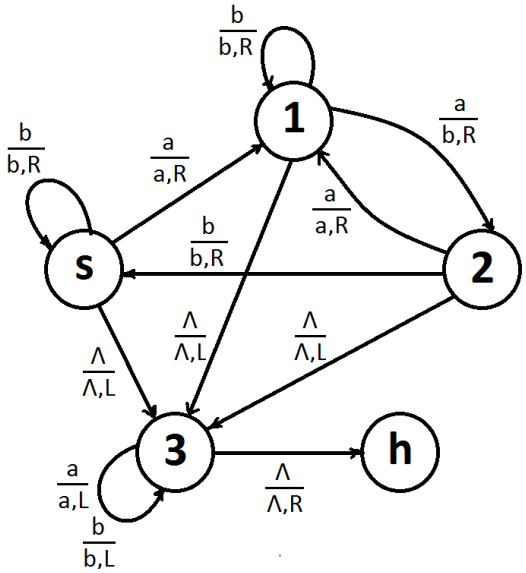
   (b) Give the "official form" of a **Turing Machine Instruction** with a short explanation of its meaning.

   An instruction is a tuple in $(Q-\{h\})\times\Gamma\times\Gamma\times\{L,S,R\}\times Q$, which directs the machine at a specific state with its tape head reading a specific symbol to replace that symbol by another symbol, move its head (left, right or stay) and transition to another state.

   (c) Design a Turing Machine that on input $w\in\{0,1\}^*$, it shifts the contents of its tape one cell to the left and stops (there is no prescription on where the head must be positioned).



   (d) Design a Turing Machine that on input $w\in\{a,b\}^*$ changes every second "a" in the input to a "b" and halts with the head positioned on the first symbol of the output.



3. (a) Describe what is an **effective enumeration** of all Turing Machines.

An effective enumeration of all Turing Machines is a listing (possibly with repetitions) of all Turing Machines by an algorithm.

(b) (i) Describe what is means for a decision problem to be **decidable**.

A decision problem is **decidable** if there is an algorithm that, given any arbitrary instance of the problem, halts with the correct answer.

(ii) Describe what it means for a decision problem to be **partially decidable**.

A decision problem is **partially decidable** if there is an algorithm that, given any arbitrary instance of the problem:

* Answers YES for those instances of the problem that have YES answers;
* May run forever for those instances of the problem whose answers are NO.

(c) Define formally the Halting Problem describing carefully the input instances and the question posed.

Is there an algorithm that can decide, given an arbitrary program and an arbitrary input, whether the execution of the program halts on the given input?

(d) Define formally the Total Problem describing carefully the input instances and the question posed.

Is there an algorithm to tell whether an arbitrary computable function is total?

(e) True or False?

(i) The Halting Problem is partially decidable. True

(ii) The Halting Problem is decidable. False

(iii) There is an effective enumeration of all total computable functions. False

(iv) The Total Problem is decidable. False

(v) The complement of the Halting Problem is partially decidable. False

4. (a) Define a **nondeterministic algorithm**.

A **nondeterministic algorithm** for an instance I of a decision problem has two distinct stages:

Guessing: Guess a possible solution S for instance I.

Checking: A deterministic algorithm starts up to check whether the guess S from the guessing stage is a solution to instance I.
This checking algorithm will halt with the answer YES if and only if S is a solution of I, but it may or may not halt if S is not a solution of I.

(b) Define the class P.

P is the class of all decision problems that are solvable by deterministic algorithms that have worst-case running time polynomial in the size of the input.

(c) Define the class NP.

The class NP consists of those decision problems that can be solved by nondeterministic algorithms in polynomial time.

(d) Define formally the decision version of the Traveling Salesman Problem describing carefully the input instances and the question posed.

Given a set of n cities, a set of distances between them and a bound $B > 0$, does there exist a tour of the n cities that starts and ends at the same city, such that the total distance traveled is less than or equal to B?

(e) Define formally the Clique Problem describing carefully the input instances and the question posed.

Given graph G with n vertices and a natural number $k \leq n$, decide whether G has a clique with k vertices.

(f) The statement "The Clique Problem is in NP" is <u>True</u>
Proof:

Consider a graph G with n vertices and a natural number $k \leq n$.
A nondeterministic algorithm works as follows:
The Guessing Stage guesses a set of vertices $\{v_1, \ldots, v_k\}$ in G that forms potentially a clique of k vertices.
Then the Checking Stage uses at most $\frac{k(k-1)}{2}$ comparisons to check whether the k vertices are connected to each other by edges.
Since $k \leq n$, the comparisons for any instance can be checked in time $O(n^2)$.

5. (a) Define formally a **quantified Boolean formula**.

A **quantified Boolean formula** is a logical expression of the form

$$Q_1 x_1 Q_2 x_2 \cdots Q_n x_n E, \quad n \geq 1,$$

where:

- each Q_i is either $\forall$ or $\exists$;
- each x_i is distinct;
- E is a formula of the propositional calculus that is restricted to using the variables $x_1, \ldots, x_n$; the operations $\neg$, $\wedge$ and $\vee$ and parentheses.

(b) Define recursively the function val from the set of quantified Boolean formulae to the set $\{T, F\}$.

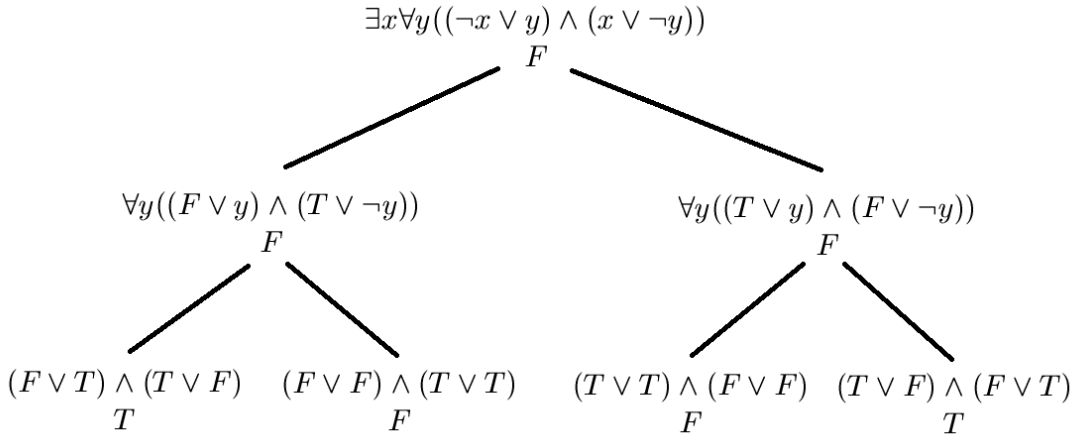
Basis:

$\text{val}(T) = T$ and $\text{val}(F) = F$;

Recursion Cases:

- $\text{val}(\neg A) = \neg\text{val}(A)$;
- $\text{val}(A \wedge B) = \text{val}(A) \wedge \text{val}(B)$;
- $\text{val}(A \vee B) = \text{val}(A) \vee \text{val}(B)$;
- $\text{val}(\forall x E) = \text{val}(E(x/T)) \wedge \text{val}(E(x/F))$;
- $\text{val}(\exists x E) = \text{val}(E(x/T)) \vee \text{val}(E(x/F))$.

(c) Construct the Recursion Tree to decide the truth value of the quantified Boolean formula $\exists x \forall y((\neg x \vee y) \wedge (x \vee \neg y))$



(d) Define formally the Quantified Boolean Formula Problem describing carefully the input instances and the question posed.

Given a quantified Boolean formula, is it true?