CS206 End-Semester Examination

Max marks: 50 **Time: 3 hours**

- *Please attach all your rough sheets.*
- *Be brief, complete and stick to what has been asked.*
- *Unless asked for explicitly, you may cite results/proofs covered in class without reproducing them.*
- ***If you need to make any assumptions, state them clearly.***
- ***Do not copy solutions from others. Penalty for offenders: FR grade.***

1. *[7.5 + 7.5 marks]* Give proofs in natural deduction (within the number of steps specified) for each of the following sequents. Each step should be either a premise or an application of a natural deduction rule. Also indicate which rule you are applying at each step.

   (a) $\exists y \forall x \neg(f(x) = y) \vdash \exists x \exists y \neg(x = y)$ *(within 10 steps)*

   (b) $\forall x \neg(f(x) = x),\ \forall x \forall y \forall z\, (x = y) \vee (y = z) \vee (x = z) \vdash \exists x \exists y\, (f(x) = y) \wedge (f(y) = x)$ *(within 15 steps)*

2. *[7.5 + 7.5 marks]* In this question, you will be given a pair of formulae, one in LTL and the other in CTL. You are required to give two Kripke structures K_1, K_2, each containing no more than 5 states, for every pair of formulae (ϕ_{LTL}, ϕ_{CTL}). Your Kripke structures should be such if s_1 represents the initial state of K_1 and s_2 represents the initial state of K_2, then the following holds:

   - $K_1, s_1 \models \phi_{LTL}$ but $K_1, s_1 \not\models \phi_{CTL}$.
   - $K_2, s_2 \models \phi_{LTL}$ and $K_2, s_2 \models \phi_{CTL}$.

   For each Kripke structure, you may draw the state transition diagram and label each state with the propositions that are true in each state. You must also indicate the starting state of each Kripke structure. The set of atomic propositions for all subquestions is $AP = \{p, q\}$. The universal path quantifier **A** is assumed to be present before all LTL formulae.

   (a) ϕ_{LTL}: $\mathbf{G}(p \rightarrow q)$, ϕ_{CTL}: $\mathbf{AF}(p \wedge q)$.

   (b) ϕ_{LTL}: $\mathbf{F}(p \wedge \mathbf{G}q)$, ϕ_{CTL}: $\mathbf{AF}(p \wedge \mathbf{AG}q)$.

3. *[5 + 5 marks]* If a first order logic sentence (i.e. formula without free variables) ϕ has a binary predicate $E(x, y)$ and no other predicates or functions, every model M of ϕ can be interpretted as a directed graph G_M. In particular, the domain (or universe) of M represents the set of nodes in G_M, and the edge relation in G_M is given by the interpretation of predicate E in model M, i.e. if $E(x, y)$ is True in M, a directed edge exists in G_M from the node corresponding to x to that corresponding to y.

   Given a directed graph G, a *clique* is a subgraph C such that there is a directed edge in G between every ordered pair of nodes in C. A clique is said to be *maximal* if there is no vertex outside the clique that has an edge to and from *all* vertices in the clique. Two cliques are said to be *disconnected* if there is no edge from any vertex in one clique to any vertex in the other.

   We wish to know if there exists a first order logic sentence ϕ that uses only the binary predicate E in addition to the equality predicate, such that:

(a) $M \models \phi$ *if and only if* G_M has exactly two maximal disconnected cliques.
If you think such a formula exists, write the formula and provide some (informal) justifcation of why you think your answer is correct. Otherwise, prove that such a first-order logic formula doesn't exist.

(b) If $M \models \phi$, then G_M has exactly two maximal cliques, not necessarily disconnected.
If you think such a formula exists, write the formula and provide some (informal) justifcation of why you think your answer is correct. Otherwise, prove that such a first-order logic formula doesn't exist.

4. *[10 marks]* In class, we discussed how to convert non-CNF propositional logic formulae to *equisatisfiable, but not semantically equivalent* CNF formulae in linear time. In this question, we wish to convert non-CNF propositional logic formulae to *semantically equivalent* CNF formulae using a propositional SAT solver. Towards this goal, we are allowed to use the following functions.

- For any propositional logic formula ϕ (not necessarily in CNF), function `EquiSAT`(ϕ) computes a CNF formula ξ *equisatisfiable* to ϕ in time linear in the size of ϕ, as discussed in class. `EquiSAT`(ϕ) returns the set of clauses of the resulting formula ξ. *Recall from our discussion in class that clauses in* `EquiSAT`(ϕ) *may have additional propositions beyond those used in* ϕ.
As an example, if $\phi = \neg(q \wedge r)$, `EquiSAT`(ϕ) returns the set $\{\neg t_1 \vee q, \neg t_1 \vee r, \neg q \vee \neg r \vee t_1, \neg t_2 \vee \neg t_1, t_1 \vee t_2\}$, where t_1 and t_2 are the new propositions.
- For a given set of clauses $\mathcal{C}$, function `PropSAT`($\mathcal{C}$) return a set μ of literals such that: (i) if μ is non-empty, then setting all literals in μ to True causes every clause in $\mathcal{C}$ to evaluate to True, and (ii) if μ is empty, then there is no assignment of truth values to literals appearing in $\mathcal{C}$ that causes every clause in $\mathcal{C}$ to evaluate to True.
As an example, `PropSAT`($\{p \vee q, \neg p \vee r\}$) can return $\mu = \{p, r\}$ or $\mu = \{\neg p, q\}$, etc.
- For a given set of literals S and a propositional logic formula ϕ, function `Filter`(S, ϕ) returns the (possibly empty) set of literals obtained by removes all literals in S that do not correspond to propositions in ϕ.
As an example, if $S = \{t_2, \neg t_1, \neg q, r\}$ and $\phi = \neg(q \wedge r)$, then `Filter`(S, ϕ) returns the singleton set $\{\neg q, r\}$.
- For a given set of literals S, function `NegateSet`(S) returns the set of literals obtained by negating each literal in S.
As an example, if $S = \{t_2, \neg t_1, \neg q, r\}$, `NegateSet`($S$) returns $\{\neg t_2, t_1, q, \neg r\}$.
- For a given set of literals S, function `Clausify`(S) returns a singleton set containing a clause obtained by disjuncting all literals in S.
As an example, if $S = \{t_2, \neg t_1, \neg q, r\}$, `Clausify`($S$) returns the singleton set $\{t_2 \vee \neg t_1 \vee \neg q \vee r\}$.

You are required to use the above functions to fill in the blank and at most 7 lines of pseudocode in the template below to convert a formula ϕ to a set of clauses `result`, such that conjuncting all clauses in `result` gives a CNF formula *semantically equivalent* to ϕ.

- The blank must be filled with a propositional logic formula.
- In each line of pseudo-code, you are allowed to invoke atmost one of the above functions, or perform some set operation like $\cup$, $\cap$, set complementation, set difference, etc.
- You may introduce at most two new program variables (like `result, curr_formula` etc) in the part of the pseudocode that you write.

Violating these restrictions will lead to deduction of marks.

```
function CNFize(phi) {
 result := empty_set;
```

```
 curr_formula := ___________;
 curr_clause_set := EquiSAT(curr_formula);
 do {
   mu := PropSAT(curr_clause_set);

   /* Fill in at most 7 lines of pseudocode here */

 } while (mu !=  empty_set);
 return(result);
}
```