CS206 Mid-Semster Examination

Max marks: 45 **Time: 2 hours**

- *Be brief, complete and stick to what has been asked.*
- *If needed, 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 or indulge in unfair means.***

1. *[15 marks]* If ϕ is a propositional logic formula and x is a proposition, Shannon's expansion of ϕ with respect to x is given by: $\phi = x \wedge \phi|_{x=\mathsf{true}} \vee \neg x \wedge \phi|_{x=\mathsf{false}}$.

   Shannon's expansion can be used to compute the ROBDD for $\phi_1 \, op \, \phi_2$ from ROBDDs for ϕ_1 and ϕ_2, where op is a binary operator. This is what lies at the core of the Apply algorithm for ROBDDs. In this question, we want to use Shannon's expansion to develop an algorithm for a special ternary operator SwapProps that is defined as follows.

   SwapProps takes three operands: a propositional logic formula ϕ and two propositions p and q. It returns a propositional logic formula obtained by replacing all occurrences of p in ϕ with q, and by replacing all occurrences of q in ϕ with p. For example, if $\phi = (p \wedge \neg q)$, then $\mathsf{SwapProps}(\phi, p, q) = (q \wedge \neg p)$. Note that $\mathsf{SwapProps}(\phi, p, q)$ can't be obtained in general by first replacing all occurrences of p in ϕ with q, and by then replacing all occurrences of q in the resulting formula with p. For example, if we replace all occurrences of p in $\phi = (p \wedge \neg q)$ with q, we get $(q \wedge \neg q) = \bot$. If we then replace all occurrences of q in $\bot$ with p, we get $\bot$, which is different from $\mathsf{SwapProps}(\phi, p, q) = (q \wedge \neg p)$.

   Given below is a sketch of an algorithm for computing the ROBDD of $\mathsf{SwapProps}(\phi, p, q)$. The following points may be noted about this algorithm.

   - p is assumed to be distinct from q.
   - The ordering of propositions for constructing ROBDDs is assumed to be fixed and known. In this ordering, if proposition x is considered earlier than proposition y for splitting in Shannon's expansion, we say that $x < y$.
   - The algorithm uses a function $\mathsf{ite}(p, \phi_1, \phi_2)$ that takes a proposition p and two propositional logic formulae ϕ_1 and ϕ_2 (as ROBDDs) and returns the ROBDD for the propositional logic formula $(p \wedge \phi_1) \vee (\neg p \wedge \phi_2)$. Function $\mathsf{ite}(p, \phi_1, \phi_2)$ requires that **that neither ϕ_1 nor ϕ_2 contain the proposition p.**
   - You may use a function $\mathsf{restrict}(\psi, p, v)$ that takes an ROBDD for the formula ψ, a proposition p and a value v that is either True or False, and returns an ROBDD for the formula obtained by setting proposition p in ψ to v. This effectively computes the cofactors of ψ.
   - You may use a function $containsProp(\psi, p)$ that takes an ROBDD for the formula ψ and a proposition p and returns True if the ROBDD has a node labeled p. Otherwise it returns False.

(Please turn over for the pseudocode)

```
SwapProps(proposition phi, ROBDD p, ROBDD q)
  // Termination of recursion
  if (p doesn't appear in  phi and q doesn't appear in phi)
     return phi;

  Let t be the proposition labeling the root node of the
  ROBDD for phi.

  // Find the next proposition in order for splitting
  if (p < t) and (p < q)
     x = p;
  else if (q < t) and (q < p)
     x = q;
  else x = t;


  // Compute the answer
  if ((x != p) and (x != q))
      return (ite(v1, psi1, psi2))

  if (x == p) then y = q;
  else y = p;

  if (C1)
     return (ite(v3, psi4, psi5));
  else
     return (ite(v4, psi6, psi7));
```

Indicate what propositions/variables must be used for `v1, v3, v4`, what Boolean condition `C1` must represent, and what formulae/function calls returning ROBDDs of formulae must be used for `psi1, psi2, psi4, psi5, psi6, psi7` so that the above algorithm always returns the correct ROBDD.

Your solution must not use any functions other than those mentioned above. In other words, no textual descriptions of what the propositions/variables, conditions or formulae are acceptable. You must give justification why you think your answer is correct. This should not require more than 2 pages of your answer book, if you got the problem right.

2. We are given a tumbler of volume V and a set of n glasses $\{g_1, \ldots g_n\}$, each filled with water. Each glass g_i has a volume $v_i > 0$. The volumes of different glasses are not necessarily identical. It is known that $\Sigma_{i=1}^{n} v_i > V$. Therefore if all the glasses were emptied into the tumbler, the tumbler would definitely overflow. Our goal is to fill up the tumbler at least upto volume $D\,(D \leq V)$ without overflowing. However, we have the restriction that a glass can either be completely emptied into the tumbler or not a drop of its water can be poured into the tumbler. In other words, you can't pour part of the water in a glass into the tumbler. You may assume that V, D and each v_i are integers.

   We wish to use propositional logic to find out if it is indeed possible to fill up the tumbler at least upto volume D without overflowing, under the constraints mentioned above. In other words, we want to construct a propositional logic formula on a suitably defined set of propositions, such that the formula is satisfiable if and only if the tumbler can be filled at least upto volume D without overflowing.

   (a) *[5 marks]* Indicate what set of propositions you would use to solve this problem. You must indicate what the truth value of each such proposition means in the context of the original problem. The total number of propositions must be polynomial in n and D (recall D is an integer). Any solution with more propositions is not acceptable.

   (b) *[10 marks]* Indicate how you will construct a propositional logic formula to capture the constraints in the above problem. Your formula must be in the form of a conjunction of different subformulae, where each subformula captures some of the constraints in the above problem. You must indicate clearly what the subformulae are, and what constraints are captured by them. The total size of your formula (i.e., no. of propositions + operators) must be a polynomial in n and D as well.

3. *[15 marks]* Consider a propositional logic formula $\phi = \phi_1 \wedge \phi_2$, where ϕ_1 is in CNF but ϕ_2 is not in CNF. The total number of propositions in ϕ is N. In addition, all N propositions appear in both ϕ_1 and ϕ_2. We wish to find out whether ϕ is satisfiable. However, since ϕ_2 is not in CNF, we cannot feed ϕ directly to a DPLL solver.

   The usual strategy of converting a non-CNF formula to an equisatisfiable CNF formula could conceivably be applied here, so that a DPLL solver can be used. However, this technique introduces additional propositions (albeit polynomial in the size of the original formula). Our DPLL solver is such that the maximum number of propositions it can handle in any formula is N, although there are no restrictions on the number of clauses that the solver can handle. Therefore, obtaining an equisatisfiable formula from ϕ with additional propositions is not admissible here.

   We wish to solve the above problem using the pseudocode shown below. In this pseudocode, the function `DPLL` is the usual DPLL procedure. You are required to fill in the pseudocode with a recursive call to `SolveWithDPLL` such that the function `SolveWithDPLL` always terminates and returns the right answer.

```
SolveWithDPLL(phi1, phi2)
 do {
   if (DPLL(phi1, emptyset) returns ‘‘UnSatisfiable’’)
      return ‘‘UnSatisfiable’’;
   else
      let mu be the satisfying assignment returned by DPLL.
      if (phi2 evaluates to true with truth assignments in mu)
         return ‘‘Satisfiable’’
      else
         // Your pseudocode comes here
 }
```

   Fill in the pseudocode above. Your answer should not require more than 5 lines of pseudocode. You must also provide justification why your solution guarantees termination of the recursion, and always produces the correct answer.