Computational Thinking 2022/23
Logic Coursework

Barnaby Martin

You should submit a single ZIP file containing (i) one PDF document containing your answers to all the theoretical/mathematical questions, and (ii) a single Python file with your code. Please name the Python file according to your username (e.g. mpll19.py).

The coding part of the coursework will be to write a SAT-solver in Python. Note that you will be restricted in some of your choices for data structures and function names. The data structure for a literal will be an integer, where a negative integer indicates the negation of the variable denoted by the corresponding positive integer. The data structure for partial assignment should be a list of literals. The data structure for a clause set should be a list of lists of literals.

1. Answer the following questions about complete sets of logical connectives, in each case justifying your answer. **[12 marks]**

   (i). Show $\{\neg, \vee\}$ is a complete set of connectives. [3 marks.]

   (ii). Show $\{\rightarrow, 0\}$ is a complete set of connectives. [2 marks.]

   (ii). Is $\{\text{NAND}, \vee\}$ a complete set of connectives? [3 marks.]

   (iv). Is $\{\rightarrow, \leftrightarrow\}$ a complete set of connectives? [4 marks.]

2. State with justification if each of the following sentences of predicate logic is logically valid **[8 marks]**

   (i). $(\forall x \exists y \forall z S(x,y,z)) \rightarrow (\neg \exists x \forall y \exists z \neg S(x,y,z))$ [2 marks].

   (ii). $(\forall x \exists y \forall z S(x,y,z)) \rightarrow (\exists y \forall x \forall z S(x,y,z))$ [2 marks].

   (iii). $(\exists y \forall x \forall z S(x,y,z) \rightarrow (\forall x \exists y \forall z S(x,y,z))$ [2 marks].

   (iv). $(\exists y \forall x \forall z \neg S(x,y,z) \rightarrow (\neg \forall x \exists y \forall z S(x,y,z))$ [2 marks].

3. Evaluate the given sentence on the respective relations S over domain $\{0,1\}$ **[8 marks]**

   (i). $\forall x \exists y \forall z S(x,y,z)$ on $\{(0,1,0),(1,0,1),(0,1,1),(1,0,0)\}$ [2 marks].

   (ii). $\forall x \exists y \forall z S(x,y,z)$ on $\{(0,1,1),(1,0,0),(0,1,0),(0,0,1),(0,0,0)\}$ [2 marks].

   (iii). $\exists y \forall x \exists z S(x,y,z)$ on $\{(1,0,0),(0,0,1),(1,1,1)\}$ [2 marks].

   (iv). $\exists y \forall x \exists z S(x,y,z)$ on $\{(1,0,0),(0,1,0),(0,1,1)\}$ [2 marks].

4. Write some Python code that loads a textual file in DIMACS format into an internal representation of a clause set, for which we will use a list of lists. For example, $(v_1 \vee \neg v_2) \wedge (\neg v_1 \vee v_3)$ would become $[[1,-2],[-1,3]]$. **[6 marks]**

5. Write a Python function `simple_sat_solve` in a single argument `clause_set` that solves the satisfiability of the clause set by running through all truth assignments. In case the clause set is satisfiable it should output a satisfying assignment. A full (truth) assignment should be represented by a list of literals. For example $v_1 \wedge \neg v_2 \wedge v_3$ would be $[1,-2,3]$. **[12 marks]**

6. Write a recursive Python function `branching_sat_solve` in the two arguments `clause_set` and `partial_assignment` that solves the satisfiability of the clause set by branching on the two truth assignments for a given variable. In case the clause set is satisfiable under the partial assignment it should output a satisfying assignment. When this is run with an empty partial assignment it should act as a SAT-solver. A partial assignment should be represented by a list of literals, as was a full assignment in the previous question. **[12 marks]**

7. Write a Python function `unit_propagate` in a single argument `clause_set` which outputs a new clause set after iteratively applying unit propagation until it cannot be applied further. **[12 marks]**

8. Write a recursive Python function `dpll_sat_solve` in the two arguments `clause_set` and `partial_assignment` that solves the satisfiability of the clause set by applying unit propagation before branching on the two truth assignments for a given variable (this is the famous DPLL algorithm but without pure literal elimination). In case the clause set is satisfiable under the partial assignment it should output a satisfying assignment. When this is run with an empty partial assignment it should act as a SAT-solver. **[20 marks]**

9. The final 10 marks of the coursework will be allocated according to the speed of your functions `unit_propagate` and `dpll_sat_solve` running on some benchmark instances. If your code is faster than mine, you receive 10 marks; within a factor of 2, 6 marks; within a factor of 3, 4 marks; within a factor of 4, 2 marks. **[10 marks]**

Total marks: 100