Homework Assignment 1
Due: October 14, 2014 at 11:00pm

Total points: 100
Deliverables: `hw1.pdf` containing typeset solutions to Problems 1-10.
`k-coloring.rkt` containing your implementation for Problem 9.

1 Propositional Logic and Normal Forms (30 points)

1. (2 points) Decide whether each of the following formulas is valid. If the formula is valid, prove its validity using the semantic argument method. Otherwise, provide a falsifying interpretation and, if the formula is satisfiable, a satisfying interpretation.

   (a) $(p \wedge q) \rightarrow (p \rightarrow q)$
   (b) $(p \rightarrow (q \rightarrow r)) \rightarrow (\neg r \rightarrow (\neg q \rightarrow \neg p))$

   (**LaTeX Hint:** use the mathpartir package to typeset proofs.)

2. (3 points) Convert the following formula to equivalent formulas in NNF, CNF, and DNF. Write the final formula as the answer; the intermediate conversion steps need not be shown.

$$\neg(\neg(p \wedge q) \rightarrow \neg r)$$

3. (5 points) Convert the formula from Problem 2 to an equisatisfiable formula in CNF using Tseitin's encoding. Write the final CNF formula as the answer.

4. (10 points) Let ϕ be a propositional formula in NNF, and let I be an interpretation of ϕ. Let the *positive set* of I with respect to ϕ, denoted $pos(I, \phi)$, be the literals of ϕ that are satisfied by I. As an example, for the NNF formula $\phi = (\neg r \wedge p) \vee q$ and the interpretation $I = [r \mapsto \bot, p \mapsto \top, q \mapsto \bot]$, we have $pos(I, \phi) = \{\neg r, p\}$. Prove the following theorem about the monotonicity of NNF:

   Monotonicity of NNF: For every interpretation I and I' such that $pos(I, \phi) \subseteq pos(I', \phi)$, if $I \models \phi$, then $I' \models \phi$.

   (**Hint:** Use structural induction.)

5. (10 points) Let ϕ be an NNF formula. Let $\hat{\phi}$ be a formula derived form ϕ using Tseitin's encoding, modified so that the CNF constraints are derived from implications rather than bi-implications. For example, given a formula

$$a_1 \wedge (a_2 \vee \neg a_3),$$

   the new encoding is the CNF equivalent of the following formula, where x_0, x_1, x_2 are fresh auxiliary variables:

$$\begin{array}{ll} x_0 & \wedge \\ (x_0 \rightarrow a_1 \wedge x_1) & \wedge \\ (x_1 \rightarrow a_2 \vee x_2) & \wedge \\ (x_2 \rightarrow \neg a_3) & \end{array}$$

   Note that Tseitin's encoding to CNF starts with the same formula, except that $\rightarrow$ is replaced with $\leftrightarrow$. As a result, the new encoding has roughly half as many clauses as the Tseitin's encoding.

   Prove that $\hat{\phi}$ is satisfiable if and only if ϕ is satisfiable.

   (**Hint**: Use the theorem from Problem 4.)

2 SAT solving (20 points)

6. (10 points) Consider the following set of clauses:

$$\begin{aligned} c_1 &: \ (x_4 \vee \neg x_1 \vee x_2) \\ c_2 &: \ (\neg x_1 \vee \neg x_4) \\ c_3 &: \ (\neg x_2 \vee \neg x_3) \\ c_4 &: \ (x_1 \vee x_3) \\ c_5 &: \ (x_1 \vee \neg x_3) \\ c_6 &: \ (\neg x_1 \vee x_5) \end{aligned}$$

Complete the table below to show how a modern CDCL SAT solver (Lecture 3) decides the satisfiability of these clauses. For choosing the next assignment in the DECIDE step, use the Dynamic Largest Individual Sum (DLIS) heuristic, and in the case of a tie, favor variable x_i with the lowest index i. If there is a tie between x_i and $\neg x_i$, pick x_i. For deriving conflict clauses, use the first unique implication points, and backtrack to the second highest decision level in the derived conflict clause. Keep all conflict clauses derived in ANALYZECONFLICT. Show the implication graph at each decision level by listing all of its labeled edges. Assume that BCP propagates implications in the increasing order of clause indices.

Level	**Decision**	**Implication Graph**	**Conflict Clause**
j	$literal@j$	$\{\langle literal@k, literal@m, c_i\rangle, \ldots\}$	$c_n : (literal \vee \ldots \vee literal)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

7. (10 points) Each conflict clause is derived from other clauses. SAT solvers can keep track of these derived-from relationships with a *resolution graph*. A resolution graph is a directed acyclic graph in which each node is (labeled with) a clause, each root corresponds to an original clause, and all other nodes correspond to learned (conflict) clauses. Each learned clause has two or more incoming edges from the clauses that were used in its derivation. In the case of an unsatisfiable formula, the resolution graph has a distinguished sink node corresponding to an empty (false) clause.

   (a) Every unsatisfiable CNF formula has one or more *unsatisfiable cores*, which are unsatisfiable subsets of the formula's clauses. Suggest an algorithm that, given a resolution graph, finds an unsatisfiable core of the original formula that is as small as possible. Your algorithm may not take more than $O(N + E)$ time, where N and E are the number of nodes and edges in the resolution graph. It should also not return the trivial core (the entire formula) if it is possible to find a smaller one in $O(N + E)$ time. Describe the algorithm in a few sentences, using high-level pseudocode (think set comprehensions).

   (b) Given an unsatisfiable core, suggest an algorithm that attempts to minimize it further. Your algorithm may invoke a (resolution-graph generating) SAT solver as a subroutine, but each such invocation must be performed on strictly fewer clauses than the previous one. It may also invoke the algorithm from Problem 7a. As above, give a brief description of the algorithm.

3 Graph Coloring with SAT (50 points)

A graph is *k-colorable* if there is an assignment of k colors to its vertices such that no two adjacent vertices have the same color. Deciding if such a coloring exists is a classic NP-complete problem with many practical applications, such as register allocation in compilers. In this problem, you will develop a CNF encoding for graph coloring and apply them to graphs from various application domains, including course scheduling, N-queens puzzles, and register allocation for real code.

A finite graph $G = \langle V, E \rangle$ consists of vertices $V = \{v_1, \ldots, v_n\}$ and edges $E = \{\langle v_{i_1}, w_{i_1} \rangle, \ldots, \langle v_{i_m}, w_{i_m} \rangle\}$. Given a set of k colors $C = \{c_1, \ldots, c_k\}$, the *k-coloring* problem for G is to assign a color $c \in C$ to each vertex $v \in V$ such that for every edge $\langle v, w \rangle \in E$, $\mathsf{color}(v) \neq \mathsf{color}(w)$.

8. (20 points) Show how to encode an instance of a k-coloring problem into a propositional formula F that is satisfiable iff a k-coloring exists.
   (a) Describe a set of propositional constraints asserting that every vertex is colored. Use the notation $\mathsf{color}(v) = c$ to indicate that a vertex v has the color c. Such an assertion is encodable as a single propositional variable p_v^c (since the set of vertices and colors are both finite).
   (b) Describe a set of propositional constraints asserting that every vertex has at most one color.
   (c) Describe a set of propositional constraints asserting that no two adjacent vertices have the same color.
   (d) Identify a significant optimization in this encoding that reduces its size asymptotically. (**Hint:** Can any constraints be dropped? Why?)
   (e) Specify your constraints in CNF. For $|V|$ vertices, $|E|$ edges, and k colors, how many variables and clauses does your encoding require?

9. (20 points) Implement the above encoding in Racket, using the provided solution skeleton. See the `README` file for instructions on obtaining solvers and the database of graph coloring problems. Your program should generate the encoding for a given graph (see `graph.rkt`), call a SAT solver on it (`solver.rkt`), and then decode the result into an assignment of colors to vertices (see `examples.rkt` and `k-coloring.rkt`). What is the minimum, maximum, and average solving time for easy and medium instances in the provided database of problems (see `problems.rkt`)? Can you solve any of the hard instances in 10 minutes or less?

10. (10 points) Describe a CNF encoding for k-coloring that uses $O(|V| \log k + |E| \log k)$ variables and clauses.

11. (**Optional: Symmetry breaking**) The encoding from Problems 8-9 can be strengthened with *symmetry breaking predicates* (SBPs). An SBP is added to a formula to make it easier to solve, without changing its satisfiability. Given the set of all interpretations of a formula that are related by a given symmetry, the SBP for that symmetry is true of only one (or some) interpretations in that set. If a formula's solution space exibits (exponentially) many symmetries, breaking just a few is often enough to make it significantly easier to solve.

    In the case of k-coloring, any permutation of the colors $c_1, \ldots, c_k$ is a symmetry of the problem: a valid assignment of colors to vertices remains valid if the colors are permuted in any way. Similarly, an invalid assignment of colors to vertices remains invalid under every permutation of the colors.

    Design and implement a (partial) symmetry breaking predicate for the encoding from Problems 8-9 using the lex-leader method described in [1]. Can you now solve any of the hard k-coloring instances?

12. (**Optional: Finding the chromatic number of a graph**) Most modern SAT solvers support *incremental solving*—that is, obtaining a solution to a CNF, adding more constraints, obtaining another solution, and so on. Because the solver keeps (some) learned clauses between invocations, incremental solving is generally the fastest way to solve a series of related CNFs. How would you apply incremental solving to your encoding from Problem 9 to find the smallest number of colors needed to color a graph (i.e., its chromatic number)?

References

[1] J. Crawford, M. Ginsberg, E. Luks, A. Roy. Symmetry-breaking predicates for search problems. In Fifth International Conference on Principles of Knowledge Representation and Reasoning, 1996.