Exercise sheet: Polynomial time formula classes

Exercise 1: 2-CNF-SAT

A 2-CNF Formula is a Formula in CNF, where every clause contains exactly 2 literals. Show that SAT restricted to 2-CNF Formulae is decidable in Polynomial time:

Let F be a 2-CNF formula. Consider the directed graph $G = (V, E)$ where

$$V = \{p, \neg p \mid p \in \text{Vars}(F)\}$$

$$E = \left\{(\overline{L}_1, L_2), (\overline{L}_2, L_1) \mid \{L_1, L_2\} \text{is a clause of} F\right\}$$

Here $\overline{p} \coloneqq \neg p$ and $\overline{\neg p} \coloneqq p$.

Show that F is satisfiable if and only if there is no Literal L such that G has a cycle containing both L and $\overline{L}$.

Solution

First note, that if $(L_i, L_j) \in E$ then $\{\overline{L_i}, L_j\}$ is a clause of F and therefore $F \models L_i \to L_j$. By induction, this also holds if G contains a path from L_i to L_j. If G contains a cycle containing both L_i and $\overline{L_i}$, then $F \models L_i \leftrightarrow \overline{L_i}$ and therefore F is unsatisfiable.

Now assume G contains no such cycle. Then we can obtain a satisfying assignment in the following way: Start with an empty assignment α. While there remain undefined variables, choose a Literal L_i, for which $\alpha(L_i)$ is undefined and such that $(L_i, \overline{L_i}) \notin E^*$. This is always possible as at most one of $(L_i, \overline{L_i})$ and $(\overline{L_i}, L_i)$ can be contained in E^*. Set $\alpha(L_i) \coloneqq 1$; then set $\alpha(L_j) \coloneqq 1$ for all L_j such that $(L_i, L_j) \in E^*$. This will never lead to conflicts for the following reason: If we had previously set $\alpha(\overline{L_j}) \coloneqq 1$ then we also would have set all $\alpha(L_k) \coloneqq 1$ for all L_k with $(L_j, L_k) \in E^*$. However by construction of G if $(L_i, L_j) \in E^*$, then also $(\overline{L_j}, \overline{L_i}) \in E^*$. Since $(L_i, \overline{L_i}) \notin E^*$, it cannot be that $(L_i, \overline{L_j}) \in E^*$. Therefor we must have set both $\alpha(\overline{L_j}) \coloneqq 1$ and therefore also $\alpha(\overline{L_i}) \coloneqq 1$, before choosing L_i. But this is a contradiction to the assumption, that $\alpha(L_i)$ was undefined.

The resulting assignment is satisfying: Assume there is a clause $\{L_i, L_j\}$ such that $\alpha(L_i) = \alpha(L_j) = 0$ this is only possible if there exists a Literal L_k such that $\alpha(L_k) = 1$ and $(L_k, \overline{L_i}) \in E^*$. Since by definition $(\overline{L_i}, L_j) \in E$, it follows that $(L_k, L_j) \in E^*$. But then $\alpha(L_j)$ should have been set to 1 in the same iteration.

Exercise 2: Horn Formulae

1. Show that for any CNF formula F one can compute in polynomial time an equisatifiable formula $G_1 \wedge G_2$, with G_1 a Horn formula and G_2 a 2-CNF formula. Justify your answer.

   (**Hint:** Consider first the case that F consists of a single clause.)

2. A *renamable Horn formula* is a CNF formula that can be turned into a Horn formula by negating (all occurrences of) some of its variables. For example,

$$(p_1 \vee \neg p_2 \vee \neg p_3) \wedge (p_2 \vee p_3) \wedge (\neg p_1)$$

can be turned into a Horn formula by negating p_1 and p_2.

Given a CNF-formula F, show how to derive a 2-CNF formula G such that G is satisfiable if and only if F is a renamable Horn formula. Show moreover that one can derive a renaming that turns F into a Horn formula from a satisfying assignment for G.

3. Show that a Horn-renamable CNF formula in which no unit clauses occur has a satisfying assignment which makes in every clause all literals true except at most one.

Solution

1. Suppose that F mentions propositional variables $p_1, \ldots, p_n$. Introduce new propositional variables $q_1, \ldots, q_n$ and consider the formula

$$G := F^* \wedge (p_1 \vee q_1) \wedge (\neg p_1 \vee \neg q_1) \wedge \ldots \wedge (p_n \vee q_n) \wedge (\neg p_n \vee \neg q_n)\,,$$

where F^* is obtained from F by replacing each positive literal p_i with $\neg q_i$. Note that F^* has only negative literals, so is a Horn formula.

Any assignment $\mathcal{A}$ that satisfies F can be extended to an assignment that satisfies G by defining $\mathcal{A}(q_i) = 1 - \mathcal{A}(p_i)$. Likewise any assignment that satisfies G restricts to an assignment that satisfies F.

2. Let $p_1, \ldots, p_n$ be the propositional variables appearing in F. We define G over the same set of propositional variables, where the intuitive meaning of p_i in G is "negate the variable p_i in F". The definition of G is such that a satisfying assignment $\mathcal{A}$ of G gives a recipe to obtain a Horn formula from F—negate those variables p_i in F for which $\mathcal{A}(p_i) = 1$.

Now the set of clauses of G is just the set of all two-element subsets of clauses of F. For example, if p_1, p_2 both appear in some clause of F then G contains a clause $\{p_1, p_2\}$ meaning that either p_1 or p_2 must be negated in F. A simple case analysis shows that this definition ensures that no pair of literals in some clause of F are both positive after the renaming prescribed by G.

3. We first observe that the assignment $\mathcal{A}_0$, which sets every variable to false, is a satisfying assignment for every Horn formula in which no unit clauses occurs. Moreover, $\mathcal{A}_0$ sets all but at most one literal in every clause to true. Given a Horn-renamable CNF formula F and A such that $F[A]$ is a Horn formula, we have that $\mathcal{A}_0$ is a desired assignment for $F[A]$, and consequently

$$\mathcal{A}(x) := \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise} \end{cases}$$

is a desired assignment for F.

Exercise 3: Solving Sudoku with a SAT-Solver

Write a program which solves sudokus using a SAT-Solver. Start by constructing a CNF Formula F which is satisfiable if and only if the sudoku has a solution. Pass the formula to a SAT Solver (for example picosat) and extract a solution for the sudoku from a satisfying assignment. Your program should read in sudokus from stdin in the following format: Each line is a string of $81 = 9 \cdot 9$ digits representing the concatenation of all rows of the sudoku. Blank squares are represented by the digit 0. It should then print out the solution in the same format and wait for the next input. Try to come to make it as fast as possible. The three students with the fastest implementations will get a bar of chocalate as a reward. You may use any language you want, but need to provide building instructions and a list of dependencies. Upload your code to moodle by **Wednesday, May 14th, 10:00 AM CEST**.