Université de Bordeaux
M2 Informatique

Software Verification
SAT and SMT

1 SAT

1.1 CNF and DNF

Exercise 1.1
The goal of this exercise is to understand why, while tempting, SAT solvers do not put formulæ in disjunctive normal form.

1. Give an algorithm deciding whether a propositional formula in Disjunctive Normal Form (DNF) is satisfiable. What is its time complexity?
2. Transform the following formula in DNF: $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge (x_3 \vee y_3)$ (*i.e.*, give a formula in DNF satisfied with exactly the same valuations. What is the size of this formula? What size would be the DNF of a similar formula with n clauses?
3. Do you think it is possible to transform any propositional formula in CNF in polynomial time? Why?

Exercise 1.2
Most SAT solvers convert their formulæ in conjunctive normal form (CNF). We show here that, while counter-intuitive viewing the result of last exercise, it is reasonable.

1. Transform the following formula in CNF: $x \Leftrightarrow (y \wedge z)$. What is the size of this formula?
2. Same question for $x \Leftrightarrow (y \vee z)$.

   We now consider the formula $\phi = (\neg(x_1 \wedge \neg x_2) \vee (\neg x_2 \vee x_3)) \wedge (\neg x_3 \wedge (x_1 \vee x_2))$.
3. Represent ϕ as a tree.
4. For each binary node of the tree, introduce a fresh variable associated to it, and write a formula (with $\Leftrightarrow$) that describes its relation with its children (taking into account the negation).
5. Using these facts, construct a formula ψ in CNF containing these new variables that is satisfiable if and only if ϕ is satisfiable. Furthermore, the formula should be such that the valuations satisfying ψ satisfy ϕ, and every valuation satisfying ϕ can be extended into a valuation satisfying ψ. For readability, you can keep the subformula of the form $x \Leftrightarrow (y \vee z)$ and $x \Leftrightarrow (y \wedge z)$ in this form. What is the size of the CNF formula?
6. Can you apply this algorithm to any formula? What is the size of the CNF formula in function of the number of literals of the original formula?

Exercise 1.3
This exercise is here to save you some time: you didn't just earn 1 million dollars.

1. Give a polynomial algorithm deciding whether a propositional formula in CNF is valid.
2. Why can't you use the previous exercise to decide the validity of any propositional formula in polynomial time?

1.2 Davis, Putnam, Logemann, Loveland (DPLL) algorithm

The DPLL algorithm is the base of most modern SAT solvers. They use it with good heuristics allowing to predict which branch have more chance to yield a result, and keep information from the already explored branches.

In the following rules, $\mathcal{C}$ stands for a set of clauses, C for a clause, and ℓ for a literal.

We depict here the rules of transformation used in the algorithm.

$$\frac{\mathcal{C} \cup \{C, \ell\}}{\mathcal{C} \cup \{C \vee \neg\ell, \ell\}} \text{ Unit Propagation} \qquad \frac{\mathcal{C} \cup \{\ell\}}{\mathcal{C} \cup \{C \vee \ell, \ell\}} \text{ Pure Litteral Elimination}$$

$$\frac{\mathcal{C} \cup \{\ell\}}{\mathcal{C}} \text{ Split True} \qquad \frac{\mathcal{C} \cup \{\neg\ell\}}{\mathcal{C}} \text{ Split False}$$

The algorithm DPLL is the following:

```
DPLL algorithm (C){

        While(UP is applicable){
                C := UP(C)
        }
        While(PLE is applicable){
                C := PLE(C)
        }
        If(False in C) Then return False;
        If(C is empty) Then return True;
        If(DPLL(Split True(C))) return True
        Else Return (Split False(C));
}
```

In the previous algorithm and rules (namely for split), we did not specify how to choose the literal to split. That is the main point on which SAT solvers apply their clever heuristics. We do not ask you to be as clever as a SAT solver here, and these heuristics are out of the scope of this course.

Exercise 1.4

1. Apply the algorithm on the formula ϕ of Exercise 1.2. Represent the application as a tree (it is easier to do).

2. Prove the correction of the algorithm, i.e., prove that the rules do not change the satisfiability of the formula.

3. How can you argue the algorithm always terminate? You may need to do a (simple) assumption not written above.

2 Modelling in FO (without quantifier)

Exercise 1.5
We consider the 3-colouring problem over graphs. We consider we are equipped with a binary predicate Edge(x,y) indicating the presence of an edge between x and y, and 3 unary predicates for colors (Red(x), Green(x) and Blue(x)).

1. Write a formula stating that a node x has only one color.

2. Write a formula stating that two nodes x and y have the same color SameColor(x,y). You may use it as a predicate in the rest of the exercise.

3. Write a formula stating that if there is an edge between x and y, they don't share the same colour.

   We now consider the vertices to be integers.

4. Write a formula stating that if two different nodes x and y have the same colour, their sum have a different colour, and they both have a edge towards it.

5. Write a formula stating that if two different nodes have the same colour, their product have the same colour.

6. Write a formula stating that if a node is even, it has an edge towards its successor.