M1 Info Project: SAT Solver 2018

Due date: Sunday, Decembre 2^{nd} 2018

I Reminder on Propositional Logic

- An **atom** is a Boolean variable, *i.e.* a variable that can have two values : True and False (equivalently True = 1 and False = 0).
- A formula is a **literal** iff it is an atom or the negation of an atom.
  x_1 and $\neg x_1$ are literals ; $x_1 \vee x_2$ is not a literal.
- A formula is a **clause** iff it is a disjunction of literals.
  $x_1 \vee x_2 \vee \neg x_3$ is a clause ; $x_1 \vee (x_2 \wedge x_3)$ is not a clause.

Definition : Conjunctive Normal Form
A formula is in **Conjunctive Normal Form** (CNF) iff it is a conjunction of clauses. Any propositional formula can be transformed into an equivalent CNF formula.

- An **interpretation** is a function that maps every atom to a truth value (0 or 1).
- A literal x is satisfied by an interpretation ω iff $\omega(x) = 1$.
- A literal $\neg x$ is satisfied by an interpretation ω iff $\omega(x) = 0$.
- A clause is satisfied by an interpretation if at least one of its literals is satisfied.
- A CNF formula is satisfied by an interpretation if every clause is satisfied.

Definition : Model of a formula
An interpretation that satisfies a formula is called a **model** of the formula.

II SAT Solvers

1. Basic Notions

The problem of **propositional satisfiability (SAT)** is the decision problem : *Given a CNF formula φ, is φ satisfiable ?*
A SAT solver is a software dedicated to give solutions for the problem SAT. Such a solver reads a formula given as a CNF, and determines whether it is satisfiable or not. Moreover, if it is satisfiable, a model must be given.

2. Solver Input

Definition : Dimacs format
A SAT solver reads a CNF as a Dimacs file. The format is :

```
p cnf nbVar nbClauses
first clause 0
second clause 0
etc 0
```

The first line starts with `p cnf`, followed by the number of variables and the number of clauses. Then, there is one line for each clause. In each line, the literals are represented as integers ; the positive integer value i represents the literal x_i, and the negative integer value $-i$ represents the literal $\neg x_i$. Then, the clause ends with a 0. For instance, the following dimacs file represents the CNF formula $(x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4)$.

```
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

We notice that it is possible that some variables do not appear in the formula (here, there is no occurrence of x_2).

3. Solver Output

The output must also respect some constraints :

- comments (any information that authors want to emphasize), beginning with the two chars `c`
- solution (satisfiable or not). Only one line of this type is allowed. This line must be one of the following ones :
  - `s SATISFIABLE`
  - `s UNSATISFIABLE`
- values of a solution (if any), beginning with the two chars `v` .

For instance, the formula φ from the previous example is satisfied by the interpretation ω defined as : $\omega(x_1) = 1$, $\omega(x_2) = 1$, $\omega(x_3) = 0$, $\omega(x_4) = 1$, $\omega(x_5) = 0$. This corresponds to the output :

```
s SATISFIABLE
v 1 2 -3 4 -5
```

4. SAT Solving Algorithm

Now we present a (very) simple algorithm for solving SAT instances. In the following, the set of clauses of φ is C, and the set variables is V. First, an interpretation ω is initialized : $\omega \leftarrow \emptyset$. Then we apply the algorithm `backtracking`.

Procedure 1 backtracking

Input: φ : CNF formula
Output: ω is a model of φ if φ is SAT

1: Pick some variable $v \in V$, $V = V \setminus \{v\}$
2: $\varphi' \leftarrow$ `simplify`(φ, v)
3: **if** $\varphi' = \emptyset$ **then**
4:     $\omega \leftarrow \omega \cup \{v\}$
5:     **return** ω
6: **else**
7:     **if** φ' contains an empty clause **then**
8:         $\varphi'' \leftarrow$ `simplify`$(\varphi, \neg v)$
9:         **if** $\varphi'' = \emptyset$ **then**
10:             $\omega \leftarrow \omega \cup \{\neg v\}$
11:             **return** ω
12:         **else**
13:             **if** φ'' contains an empty clause **then**
14:                 **return** null
15:             **else**
16:                 **return** `backtracking`(φ'')
17:             **end if**
18:         **end if**
19:     **else**
20:         **return** `backtracking`(φ')
21:     **end if**
22: **end if**

The idea is simple : we choose a variable v, and we try to assign it the value 1. We obtain a simplified formula (line 3). If the simplified formula is empty, it means that φ is satisfiable. If there is an empty clause in the simplified formula, it means that φ' is not satisfiable, so we must change the value of v from 1 to 0, and compute a new simplified formula (line 9). If we do not obtain satisfiability or unsatisfiability at this step, we compute recursively the backtracking algorithm on the simplified formula. Now, let us present the `simplifiy` algorithm. If the literal l belongs to a clause c, then c is satisfied by l, whatever the values of the other variables. So this clause is useless for the rest of the search, and it can be removed. If the negation

---

Procedure 2 simplify

Input: φ : CNF formula, l : literal
Output: φ' is a simplification of φ by l

$\varphi' \leftarrow \varphi$
for all c a clause in φ **do**
    if $l \in c$ **then**
        remove c from φ'
    else
        if $\neg l \in c$ **then**
            remove $\neg l$ from c in φ'
        end if
    end if
    return φ'
end for

---

of l belongs the c, then the literal can be removed from the clause : if the clause is satisfied, it comes from another variable.

III Instructions

- You must implement a SAT solver, *i.e.* a program that reads Dimacs files, and prints the output accordingly to Section II.2 and II.3.
- The solver must be implemented in Java, C or C++ ; the command line only takes one parameter : the Dimacs file.
- You must deliver before Sunday, Decembre 2^{nd}, 23 :59 (Paris time) the whole source code, and a script `build.sh` that allows to build an executable file (or an executable jar archive). The solver must work on Ubuntu (18.04 Bionic Beaver) or MacOS (10.14 Mojave).
- A short report of the project is expected. The report must describe every programming choice, and every improvement of the algorithm. If some of your improvements comes from some external source, this source must be mentionned. The report must be delivered as a pdf file.
- The solver must be jointly implemented by two or three students.
- The source code, the script `build.sh` and the pdf file must be contained in directory corresponding to the students' names (for instance `JohnDoe_JaneDoe`). This directory must be uploaded on Moodle as an archive (zip or tar.gz).
- There will be penalties for any violation of these instructions.