Project INF432

1 Outline

This project consists in using a SAT-solver to automatically solve some problems formalized in clauses. A SAT-solver is a generic tool for finding a model of a set of formulas : thus, your task is not to find specific resolution strategies for your problem, but rather to translate that problem into the formalism the SAT-solver expects, and to let it find a solution by itself.

You will have to choose a problem (a logic grid game such as Sudoku) and study its rules. You will then modelize a grid of your problem using boolean variables only, and you will translate the rules of the game into logical formulas, more specifically conjunctive normal forms.

Once the problem has been modelled and validated by your TD lecturer, you will program a software **using the programming language of your choice**, which takes an instance of your problem as an input, and creates a DIMACS file encoding that instance as a formula in conjunctive normal form.

You will then choose a SAT-solver [1] that you will use to treat these DIMACS files.

Finally, using the execution trace from the SAT-solver, you will display an understandable solution to the instance of your problem.

Optionally, depending on how fast you've progressed in your project, you may transform the model of your problem into equivalent 3-SAT clauses (you can use the algorithm proposed in exercise 51 p. 48 of your exercise handout). If your group has been really fast, you may program your own SAT-solver too, which will be made easier using the 3-SAT transformation.

Note : Your grade for the project will be based on three tasks :

— writing a clear, well-structured and rigorous report ;

— delivering a well-structured, readable source code together with relevant test cases ;

— actively participating to a presentation, which will include a demo of your software.

In particular, any student that does not contribute significantly to the presentation faces the risk of receiving the grade 0 for their project.

1. For instance, MiniSat, Z3, Sattime, SAT4j, SATzilla, precosat, MXC, clasp, SApperloT, TNM, gNovelty2+, Rsat, Picosat, Minisat, Zchaff, Jerusat, Satzoo, Limmat, Berkmin, OKSolver, ManySAT 1.1, ...(`http://www.satcompetition.org/`).

2 Your tasks

For this project, students must work in groups (of 3 people ideally, or 4 if need be), select one of the suggested topics or choose one and have it approved by the TD lecturer.

2.1 Deliverables

Your group will have to produce :

1. A report in which :
   — you describe the problem you chose in English (rules, examples, constraints) ;
   — you translate that problem into (propositional or first-order) logic ;
   — you model the problem in conjunctive normal form (product of clauses) ;
   — you explain the important points of the implementation of your programs ;
   — you illustrate the use of your programs with relevant examples.
2. A collection of programs :
   — an interface which takes an instance of the problem as an input, and creates a DIMACS file encoding that instance.

     In order to make testing efficient, your interface should be able to take its inputs at least from a file [2], in a format you will define. Optionally, you can also let the user enter an instance through the keyboard or a graphical interface.

   — if applicable, a program which reads a DIMACS file and returns a set of 3-SAT clauses in DIMACS format ;
   — a program which, given the trace of the SAT-solver, displays an understandable solution to the problem ;
   — ideally, a program which chains these steps together with the call to the SAT-solver.
3. A well-chosen set of instances of your problem, which will allow to test your modelization.
4. A group presentation during which you will present your problem, its modelization, and you will run your programs on your personal computer, or a university computer.

2.2 Planning

— This document is online from the very beginning of the course, a short version is distributed to students.
— Your group and topic have to be chosen before the end of the 2nd week of TD.
— Delivery, by email to the TD lecturer, of the pre-report describing the modelization part of the project (max 5 pages) no later than Sunday before the Midterm exams.
— Delivery, by email to the TD lecturer, of the final report and the source codes, no later than the week of the last lecture.
— Group presentation of your project during the last 2 weeks before the exams (15 minute presentations, question time included).

2. if you are using JAVA, you can mimic the example found at `www-verimag.imag.fr/~devismes/JAVA/Fichier.java`.

3 Technical notes

SAT and n-SAT Knowing if a propositional logic formula in conjunctive normal form (CNF) is satisfiable is called "satisfiability problem" or "SAT problem". When a CNF formula only contains clauses of n literals, the problem is called " n-SAT problem".

DIMACS format : The format used as input to SAT-solvers is an international standard used for encoding formulas in conjunctive normal form. A file in the DIMACS format begins with a line specifying that the formula is in normal form, the number of variables in the formula, and how many clauses it contains. For example, `p cnf 5 3` says that the file contains a formula in conjunctive normal form with 5 variables and 3 clauses. Then, the following lines describe the clauses, one on each line. Each line contains positive or negative integers, and ends with a zero. A positive integer i indicates that the i^{th} variable appears in the clause, whereas a negative integer $-i$ indicates that the negation of the i^{th} variable atppears in the clause. For example, here's a formula, followed by its encoding using the DIMACS format :

$$(x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4)$$

```
c
c start with comments
c
c
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

4 Examples of Possible Topics

Any logical game that consists in filling a grid with all the useful information available can be modeled. For instance, you could tackle :

- Easy :
  - Easy as ABC
  - N queens
  - Mosaic
  - Squaro
  - Coloration de graphes
- Medium :
  - Fill-a-pix
  - Sudoku
  - Norinori
  - Futoshiki
  - Makaro
- Hard :
  - Juosan

    - Tents
    - Light Up
    - Usowan
    - Hitori
    - Kurotto
    - Tetravex
- Any other topic validated by your TD lecturer

Other topics will be listed on page
`https://wackb.gricad-pages.univ-grenoble-alpes.fr/inf402/`
together with links to demo applets.

5 Example with the Pigeon Problem

We illustrate our minimum expectations for this project with a simple problem.

5.1 Problem

A pigeon-fancier owns n nests and p pigeons. He wishes that :
- each pigeon lives in a nest,
- there be at most one pigeon per nest

How can we help him with logic ?

5.2 Modeling the Problem Using First Order Logic

Predicate $P(i, j)$ is true if and only if pigeon i is in nest j. The constraints of the probmes are modelled as follows :
- Each pigeon lives in a nest : $\forall i, \exists j, P(i, j)$.
- There is at most one pigeon in each nest : $\forall i, \forall k, i \neq k \Longrightarrow \forall j, \overline{P(i, j)} \vee \overline{P(k, j)}$.

5.3 Modelization in Conjunctive Normal Form

The boolean variable $x_{i,j}$ is true if and only if pigeon i lives in nest j. For example, for a set of pigeons $\{a, b, c\}$ and a set of nests $\{1, 2, 3, 4\}$, the following constraints must be given to the SAT-solver :

$$\begin{array}{ll}
 & (x_{a,1} \vee x_{a,2} \vee x_{a,3} \vee x_{a,4}) \\
\wedge & (x_{b,1} \vee x_{b,2} \vee x_{b,3} \vee x_{b,4}) \\
\wedge & (x_{c,1} \vee x_{c,2} \vee x_{c,3} \vee x_{c,4}) \\
\wedge & (\overline{x_{a,1}} \vee \overline{x_{b,1}}) \wedge (\overline{x_{a,1}} \vee \overline{x_{c,1}}) \wedge (\overline{x_{b,1}} \vee \overline{x_{c,1}}) \\
\wedge & (\overline{x_{a,2}} \vee \overline{x_{b,2}}) \wedge (\overline{x_{a,2}} \vee \overline{x_{c,2}}) \wedge (\overline{x_{b,2}} \vee \overline{x_{c,2}}) \\
\wedge & (\overline{x_{a,3}} \vee \overline{x_{b,3}}) \wedge (\overline{x_{a,3}} \vee \overline{x_{c,3}}) \wedge (\overline{x_{b,3}} \vee \overline{x_{c,3}}) \\
\wedge & (\overline{x_{a,4}} \vee \overline{x_{b,4}}) \wedge (\overline{x_{a,4}} \vee \overline{x_{c,4}}) \wedge (\overline{x_{b,4}} \vee \overline{x_{c,4}})
\end{array}$$

5.4 Resolution by a SAT-solver

After translating the cnf formula into the DIMACS format, a SAT-solver will give an assignment to each variable satisfying the system of contraints. Using this, we can deduce how to place each pigeon. We note that while this problem is very easy to solve manually, the computer will have to solve an exponential number of constraints to solve it using logic.

6 Optional part : SAT solver

For the most advanced groups, you have the possibility of **programming** your own SAT solver (in your favourite programming language) with several resolution strategies.

Your solver will be a *WalkSat* type solver (see paragraph below). It will take as input the set of clauses to resolve in the form of a DIMACS file. The aim is then to return an assignement satisfying all clauses in that set, whenever possible. Notice that *WalkSat* solvers are *incomplete* : if the set of clauses given as input is contradictory the solver will not be able to determine it.

The pseudo-code of the *WalkSat* algorithm is given below as a reference.

```
 1: Draw an assignment v at random according to a uniform distribution
 2: i = 0
 3: WHILE (v is not a model) and i < N DO
 4:      Pick a clause C amongst clauses C' such that v(C') = false
 5:      Draw a real number q in [0,1] according to a uniform distribution
 6:      IF q < P THEN
 7:           Pick a variable x in C according to a uniform distribution
 8:      ELSE
 9:           Pick a variable x in C deterministically
10:      END IF
11:      Flip the value of v(x) in the assignement v
12:      i++
13: END WHILE
14: IF v is a model THEN
15:      RETURN v
16: ELSE
17:      RETURN "undecided"
18: ENDIF
```

The code above has gaps, and that is on purpose... Indeed it does not specify the values of N, P, or the procedure to deterministically choose variable x at line 9.
It is up to you to determine a reasonable value for such constants, and to choose variable x so that it minimizes the number of unsatisfied clauses.

Optionally, you may experiment so as to find the best possible values of N and P, and program other heuristics for the choice of x, for instance :

- Choose the least modified variable of C ;
- Assign to each variable y a score with computed as the difference between the number of positive occurences, and negative occurences across clauses. Then choose the variable with the best score ;
- Heuristics inspired from MOMS and JW heuristics given in the course notes ;
- A mix some of these strategies ;

— Etc.

In that case it would be interesting to compare the different heuristics that you implemented.

Note that to simplify your implementation, you can assume that the set of clauses received has been preprocessed to get an equivalent 3-SAT problem (see end of Section 1 of this document).