Formal Method Mod. 1 (Automated Reasoning)

Laboratory 1

Giuseppe Spallitta
giuseppe.spallitta@unitn.it

Università degli studi di Trento

March 10, 2021

Course overview

- Schedule: **Wednesday, 11:30-13:30**
- Laboratories will focus on **modelling and resolution** of SAT and SMT problems
- Examples in class + homework (solutions will be provided during the course).
- The exam will contain exercises similar to the ones shown during lectures

Outline

1. First steps on SAT solving
   Quick overview of Glucose
2. Getting used with DIMACS format
3. Simple real-life applications

Glucose 4.0

- Given a SAT problem, we want to determine if there is a satisfiable assignment to each variable s.t. the problem evaluates to true.
- In these classes we will use **Glucose 4.0** (https://www.labri.fr/perso/lsimon/glucose/) to efficiently obtain the answer.
- Instructions to install the tool have been uploaded to the Moodle page of the course.

Glucose 4.0 (cont.d)

- SAT solver developed by Gilles Audemard and Laurent Simon
- Heavily based on Minisat, an older minimal SAT solver
- Easy to setup and to use, it provides a minimal yet efficient interface to deal with satisfiability problems.
- It relies on the CDCL algorithm.
- It has been chosen as the base solver for the "Hack track sAT competition", focusing on improving the solver through minimal modifies: (https://satcompetition.github.io/2020/track_hack.html)

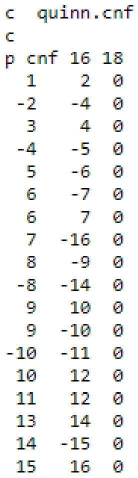
Why using SAT solvers?

Let's try the "SAT game" and we will see the importance of SAT solving...
`http://www.cril.univ-artois.fr/~roussel/satgame/satgame.php?level=5&lang=eng`

The DIMACS format

- If we want to use Glucose, we need to know the **input** format and the **output** provided by the tool.
- The input format accepted by the tool is called **DIMACS format**
  - Widely considered the standard input format for SAT solving
  - Benchmarks are created using this standard

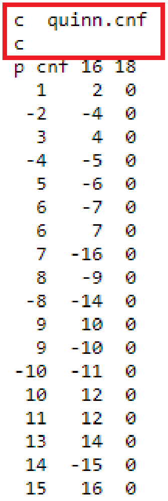
The DIMACS format: an example



```
c  quinn.cnf
c
p cnf 16 18
  1    2  0
 -2   -4  0
  3    4  0
 -4   -5  0
  5   -6  0
  6   -7  0
  6    7  0
  7  -16  0
  8   -9  0
 -8  -14  0
  9   10  0
  9  -10  0
-10  -11  0
 10   12  0
 11   12  0
 13   14  0
 14  -15  0
 15   16  0
```

It seems difficult to read and understand, but actually it's easier than you thought!

The DIMACS format: an example (cont.d)



```
c   quinn.cnf
c
p cnf 16 18
  1    2  0
 -2   -4  0
  3    4  0
 -4   -5  0
  5   -6  0
  6   -7  0
  6    7  0
  7  -16  0
  8   -9  0
 -8  -14  0
  9   10  0
  9  -10  0
-10  -11  0
 10   12  0
 11   12  0
 13   14  0
 14  -15  0
 15   16  0
```

Comments

Each row starting with a lower case *c* is a comment
$\Rightarrow$ use it to explain the SAT problem you encoded and other useful information

The DIMACS format: an example (cont.d)

```
c  quinn.cnf
c
p cnf 16 18
  1    2  0
 -2   -4  0
  3    4  0
 -4   -5  0
  5   -6  0
  6   -7  0
  6    7  0
  7  -16  0
  8   -9  0
 -8  -14  0
  9   10  0
  9  -10  0
-10  -11  0
 10   12  0
 11   12  0
 13   14  0
 14  -15  0
 15   16  0
```

Problem line

- The first non-comment line must be the problem line, starting with a lower case *p*
- The first word is the **problem type** (in our case CNF)
- The first number is the **number of variables**
- The second number is the **number of clauses**

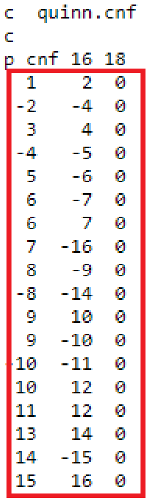
The DIMACS format: an example (cont.d)

```
c  quinn.cnf
c
p cnf 16 18
  1    2  0
 -2   -4  0
  3    4  0
 -4   -5  0
  5   -6  0
  6   -7  0
  6    7  0
  7  -16  0
  8   -9  0
 -8  -14  0
  9   10  0
  9  -10  0
-10  -11  0
 10   12  0
 11   12  0
 13   14  0
 14  -15  0
 15   16  0
```

Clauses

- Each subsequent row contains a single clause
- A clause is defined by listing the index of each positive literal, and the negative index of each negative literal.
- The last number, 0, tells the solver that the previous clause ended and we are starting with a new clause.

The DIMACS format: an example (cont.d)



```
c  quinn.cnf
c
p cnf 16 18
  1    2  0
 -2   -4  0
  3    4  0
 -4   -5  0
  5   -6  0
  6   -7  0
  6    7  0
  7  -16  0
  8   -9  0
 -8  -14  0
  9   10  0
  9  -10  0
-10  -11  0
 10   12  0
 11   12  0
 13   14  0
 14  -15  0
 15   16  0
```

Clauses: example

If 1 is the identifier of x_1, 2 the identifier of x_2 and so on, then the first three clauses are:

- $x_1 \vee x_2$
- $\neg x_2 \vee \neg x_4$
- $x_3 \vee x_4$

Glucose output

Once you create your file using the DIMACS format, you can feed it to the solver:

- If no solution exists to the problem, the solver returns **UNSAT**
- If at least one solution exists, the solver returns **SAT**. If the path of an output file is provided, a valid assignment satisfying the problem is printed into it, using the same notation of DIMACS to indicate positive or negative assignments for each variable.

Outline

First encodings

Exercise 1.1

Encode the following Boolean formulas and check their (un)satisfiability:

- $\varphi_1 := (x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_2)$
- $\varphi_2 := (x_1 \rightarrow x_2) \vee x_3$

First encodings

Exercise 1.1

Encode the following Boolean formulas and check their (un)satisfiability:

- $\varphi_1 := (x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_2)$
- $\varphi_2 := (x_1 \rightarrow x_2) \vee x_3$

- The first formula can be easily encoded using the DIMACS format!

First encodings

Exercise 1.1

Encode the following Boolean formulas and check their (un)satisfiability:

- $\varphi_1 := (x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_2)$
- $\varphi_2 := (x_1 \rightarrow x_2) \vee x_3$

- The first formula can be easily encoded using the DIMACS format!
- The second formula should be written into CNF format before feeding it to the solver!

$$\varphi_2 = (\neg x_1 \vee x_2) \vee x_3$$

First encodings (cont.d)

Homework 1.1

Encode the following Boolean formulas and check their (un)satisfiability:

- $\varphi_3 := \neg x_1 \rightarrow (x_1 \rightarrow x_2)$
- $\varphi_4 := (x_1 \leftrightarrow x_2)$

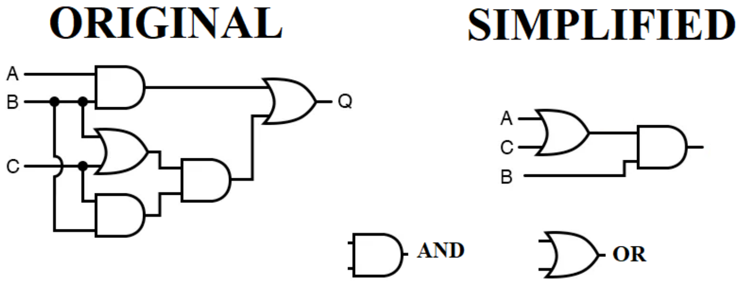
Outline

1. First steps on SAT solving
2. Getting used with DIMACS format
3. Simple real-life applications
   - Testing Boolean circuits
   - Solving simple logic puzzles

Testing Boolean circuits

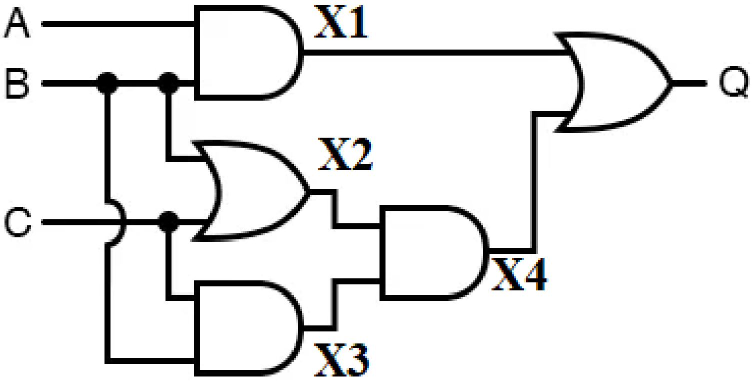
Exercise 1.2

You are asked to build a circuit for a top-secret project. Each gate costs a lot of money, so you suggest an alternative and cheaper circuit. Are the two circuits are equivalent?



Testing Boolean circuits (cont.d)

The first step is encoding the two Boolean circuit. Starting with the first one, let's define intermediate variables to store the result of each inner gate and the final output:



Testing Boolean circuits (cont.d)

Each gate represents a clause of the final encoding. The upper left AND gate can be represented as:

$$X1 \leftrightarrow A \wedge B$$

This formula must be converted into its equivalent CNF form. In this case:

$$(X1 \rightarrow (A \wedge B)) \wedge (X1 \leftarrow (A \wedge B))$$

$$(\neg X1 \vee (A \wedge B)) \wedge (\neg(A \wedge B) \vee X1)$$

$$(\neg X1 \vee A) \wedge (\neg X1 \vee B) \wedge (\neg A \vee \neg B \vee X1)$$

The same pattern can be applied for each AND gate!

Testing Boolean circuits (cont.d)

The leftmost OR gate can be represented as:

$$X2 \leftrightarrow B \vee C$$

This formula must be converted into its equivalent CNF form:

$$(X2 \rightarrow (B \vee C)) \wedge (X2 \leftarrow (B \vee C))$$

$$(\neg X2 \vee (B \vee C)) \wedge (\neg(B \vee C) \vee X2)$$

$$(\neg X2 \vee B \vee C) \wedge ((\neg B \wedge \neg C) \vee X2)$$

$$(\neg X2 \vee B \vee C) \wedge (\neg B \vee X2) \wedge (\neg C \vee X2)$$

The same pattern can be applied for each OR gate!

Testing Boolean circuits (cont.d)

- Using the two retrieved formulas, we can encode both circuits into a CNF equivalent formula.
- We are missing the last step: checking the equivalence of the two systems!
  - If at least one counter-example is found (the two output are not identical for the same input), then the two circuits are not equivalent.
  - As a consequence, we can encode this condition into the sub-formula:

    $$\neg(O1 \leftrightarrow O2)$$

    Satisfiability corresponds to the desired counter-example, thus the non-equivalence.

Testing Boolean circuits (cont.d)

As always we must convert into into CNF form:

$$\neg((O1 \rightarrow O2) \wedge (O2 \rightarrow O1))$$

$$\neg((\neg O1 \vee O2) \wedge (\neg O2 \vee O1))$$

$$((O1 \wedge \neg O2) \vee (O2 \wedge \neg O1))$$

$$(O1 \vee O2) \wedge (O1 \vee \neg O1) \wedge (\neg O2 \vee O2) \wedge (\neg O2 \vee \neg O1)$$

Each clause in the form $A \vee \neg A$ is always true, so they are useless and can be removed, leaving:

$$(O1 \vee O2) \wedge (\neg O2 \vee \neg O1)$$

Testing Boolean circuits (cont.d)

- We can now pass the encoding to the solver... after we map each variable into indexed integers!

Testing Boolean circuits (cont.d)

- We can now pass the encoding to the solver... after we map each variable into indexed integers!
- The solver returns UNSAT
  $\Rightarrow$ no counter-example has been found
  $\Rightarrow$ the two circuits are equivalent

The importance of modeling

- At the moment we simply converted Boolean formulas into an equivalent DIMACS format
  $\Rightarrow$ Real life is not so easy :)
- Determining the variables to describe the problem and correctly write all the necessary conditions will be the hardest task.

Sorting people

Exercise 1.3

Consider three chairs in a row and three guests: A, B e C. We know that:

- A does not want to sit next to C.
- A does not want to sit on the leftmost chair.
- B does not want to sit at the right of C

Is it possible to satisfy the following constraints and find a valid placement?

Sorting people: variables

First let's define the Boolean variables necessary to model the problem:

- x_{ij} states if user i ($i \in${A,B,C}) is sat in chair j ($j \in$1,2,3)
- In this case 3*3 = 9 variables are needed
- Remember to map each variable into an indexed variable to satisfy the DIMACS format!
    - $x_{11} \rightarrow 1$, $x_{12} \rightarrow 2$, $x_{13} \rightarrow 3$, $x_{21} \rightarrow 4$ and so on...

Sorting people:properties (1)

Now let's encode the conditions stated in the text of the problem:

- A does not want to sit next to C

$$\neg(x_{11} \land x_{32}) \land \neg(x_{31} \land x_{12}) \land \neg(x_{12} \land x_{33}) \land \neg(x_{32} \land x_{13})$$

After CNF-ization, we obtain:

$$(\neg x_{11} \lor \neg x_{32}) \land (\neg x_{31} \lor \neg x_{12}) \land (\neg x_{12} \lor \neg x_{33}) \land (\neg x_{32} \lor \neg x_{13})$$

Sorting people: properties (2)

- A does not want to sit on the leftmost chair.

$$\neg x_{11}$$

- B does not want to sit at the right of C

$$\neg(x_{22} \land x_{31}) \land \neg(x_{23} \land x_{32})$$

After CNF-ization, we obtain:

$$(\neg x_{22} \lor \neg x_{31}) \land (\neg x_{23} \lor \neg x_{32})$$

Sorting people: properties (3)

- Is this enough to model the problem?
- There are some "not obvious" conditions that must be provided!

Sorting people: properties (4)

- Guest A must sit in at least one chair:

$$x_{11} \vee x_{12} \vee x_{13}$$

The same applies for guests B and C, so we must add two additional conditions using the same pattern:

$$x_{21} \vee x_{22} \vee x_{23}$$

$$x_{31} \vee x_{32} \vee x_{33}$$

Sorting people: properties (5)

- Guest A must sit in at most one chair:

$$x_{11} \rightarrow (\neg x_{12} \wedge \neg x_{13})$$

$$x_{12} \rightarrow (\neg x_{11} \wedge \neg x_{13})$$

$$x_{13} \rightarrow (\neg x_{11} \wedge \neg x_{12})$$

Each condition must be converted into an equivalent CNF formula. The first one will become:

$$\neg x_{11} \vee (\neg x_{12} \wedge \neg x_{13})$$

$$(\neg x_{11} \vee \neg x_{12}) \wedge (\neg x_{11} \vee \neg x_{13})$$

The other two follows the same pattern.

Sorting people: properties (6)

- Guest B must sit in at most one chair
- Guest C must sit in at most one chair

Both conditions can be encoded similarly to the previous condition and thus we encode them again simply changing the involved variables.

Sorting people: properties (7)

- Only one person can sit on the first position:

$$x_{11} \rightarrow (\neg x_{21} \wedge \neg x_{31})$$

$$x_{21} \rightarrow (\neg x_{11} \wedge \neg x_{31})$$

$$x_{31} \rightarrow (\neg x_{11} \wedge \neg x_{21})$$

The structure is identical to the previous formulas, thus we can repeat the same pattern changing the variables!
We need also to encode the same typology of clauses for the second and the third position.

Sorting people: results

Now we can fed the encoding into Glucose

$\Rightarrow$ The solver returns **UNSAT**

Cracking codes

Exercise 1.4

$\boxed{1}\boxed{2}$ One number is correct and well placed

$\boxed{1}\boxed{4}$ Nothing is correct

$\boxed{4}\boxed{3}$ One number is correct but wrong placed

Assuming we can use only digits from 1 to 4, does a solution exists? Is it unique?

Cracking codes: variables

First let's define the Boolean variables necessary to model the problem:

- x_{ij} states if number i ($i \in \{1,2,3,4\}$) is placed in position j ($j \in 1,2$)
- In this case $4*2 = 8$ variables are needed

Cracking codes: properties (1)

Let's encode the first condition: "In 12, one number is correct and well placed".

- This means that either the digit 1 is correct and 2 is not part of the code or viceversa.

$$(x_{11} \land \neg x_{21} \land \neg x_{22}) \lor (x_{22} \land \neg x_{11} \land \neg x_{12})$$

Its equivalent CNF representation is:

$$(x_{11} \lor x_{22}) \land (x_{11} \lor \neg x_{12}) \land (\neg x_{21} \lor x_{22}) \land (\neg x_{21} \lor \neg x_{11}) \land (\neg x_{21} \lor \neg x_{12})$$

$$\land (\neg x_{22} \lor \neg x_{11}) \land (\neg x_{22} \lor \neg x_{12})$$

Cracking codes: properties (2)

The second condition, "In 14, nothing is correct", is easier to encode:

- 1 and 4 must be excluded from the possible valid values in each position

$$\neg x_{11} \land \neg x_{12} \land \neg x_{41} \land \neg x_{41}$$

Cracking codes: properties (3)

Now we can deal with the third condition, "In 43, one number is correct but wrongly placed".

- Either the digit 3 should be put in position 1 and 4 is not part of the code or the digit 4 should be put in position 2 and 3 is not part of the code.

$$(x_{31} \land \neg x_{41} \land \neg x_{42}) \lor (x_{42} \land \neg x_{31} \land \neg x_{32})$$

This pattern is identical to the one used to encode the first condition, so we can recycle its CNF structure!

Cracking codes: properties (3)

Similarly to the previous exercise, we must add some "hidden" conditions:

- Each position must contain at least a digit

$$(x_{11} \vee x_{21} \vee x_{31} \vee x_{41}) \wedge (x_{12} \vee x_{22} \vee x_{32} \vee x_{42})$$

- Each position must contain at most a digit.

$$x_{11} \rightarrow (\neg x_{21} \wedge \neg x_{31} \wedge \neg x_{41})$$

Whose CNF equivalent representation is:

$$(\neg x_{11} \vee \neg x_{21}) \wedge (\neg x_{11} \vee \neg x_{31}) \wedge (\neg x_{11} \vee \neg x_{41})$$

This formula must be replicated for each digit, for both position 1 and 2. A total of 8 different formulas must be encoded!

Cracking codes: results

Once we map the variables into the usual indexed integers we can feed it to Glucose and see if there is a valid solution.

Cracking codes: results

Once we map the variables into the usual indexed integers we can feed it to Glucose and see if there is a valid solution.
⇒ The solver returns **SAT** and, checking the output, the two true variables generate 32 as solution.

Cracking codes: results

Once we map the variables into the usual indexed integers we can feed it to Glucose and see if there is a valid solution.
$\Rightarrow$ The solver returns **SAT** and, checking the output, the two true variables generate 32 as solution.

To check the uniqueness of this solution, we must ensure the two digits cannot be respectively 3 and 2. This can be easily encoded using the following clauses:

$$\neg x_{31} \lor \neg x_{22}$$

Cracking codes: results

Once we map the variables into the usual indexed integers we can feed it to Glucose and see if there is a valid solution.
$\Rightarrow$ The solver returns **SAT** and, checking the output, the two true variables generate 32 as solution.

To check the uniqueness of this solution, we must ensure the two digits cannot be respectively 3 and 2. This can be easily encoded using the following clauses:

$$\neg x_{31} \lor \neg x_{22}$$

$\Rightarrow$ Now the solver returns **UNSAT**, proving the uniqueness of the solution

Homeworks

Homework 1.3: cheaters

Three students A, B and C are accused of having illegally obtained the questions for the Automated Reasoning exam. During the investigation process the students made the following statements:

- A said: "B is guilty and C is innocent"
- B said: "If A is guilty, then C is also guilty"
- C said: "I'm innocent and one of the others, perhaps even the two, are guilty"

Considering that all the students spoke the truth, which of the students are guilty and which are innocent? Solve it using Glucose.

Homeworks

Homework 1.4: password

Using the digits 1,2,3 and 4 you need to create a 3-length password. There are some rules that must be fulfilled:

- The password should be even
- We cannon use the same digit three times, otherwise it would be easy to guess it.
- It is possible to repeat the same digit twice, just make sure the two digits are not adjacent.

Solve it using a SAT solver and report the solution. Is this unique?

Homeworks

Homework 1.5: coloring graph

You are given the graph shown in the figure on the right. Suppose you want to color the nodes of this graph so that nodes connected by an edge cannot have the same color. Given these assumptions:

- Is it possible to color the graph using only 2 colors?
- Is it possible to color the graph using only 3 colors?

Solve it using a SAT solver.