UNIVERSITÀ
DI TRENTO
Department of
Information Engineering and Computer Science

Automated Reasoning and Formal Verification

Laboratory 1

Gabriele Masina
gabriele.masina@unitn.it

Università degli studi di Trento

March 05, 2025

Course overview

- Schedule: **Wednesday, 11:30-13:30**
- Laboratories will focus on **modelling and resolution** of SAT and SMT problems
- Examples in class + homework (not mandatory, but highly recommended).
- The material will be uploaded here: `https://github.com/masinag/arfv2025`

What to expect from the labs

What labs will be about

- ✓ **Hands-on** experience with SAT/SMT solvers
- ✓ **Modelling** of problems in SAT and SMT and usage of advanced features

. . . and in the second module

- ✓ **Model checking** of discrete, timed and hybrid systems

What to expect from the labs

What labs will be about

- ✓ **Hands-on** experience with SAT/SMT solvers
- ✓ **Modelling** of problems in SAT and SMT and usage of advanced features

. . . and in the second module

- ✓ **Model checking** of discrete, timed and hybrid systems

What labs will not be about

- ✗ **Algorithms** for SAT/SMT solving or model checking

What to expect from the exams

The exam will contain exercises similar to the ones shown during lectures

- If you choose only Automated Reasoning: one SAT exercise and one SMT exercise.
- If you take the full exam: one SAT/SMT exercise and one Model Checking exercise.

To copy at exams very dangerous is!

Outline

MathSAT 5

- 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 **MathSAT 5** (`https://mathsat.fbk.eu/download.html`) to efficiently obtain the answer.

MathSAT 5 (cont.d)

- MathSAT 5 is an efficient Satisfiability Modulo Theories (spoiler...) solver jointly developed by FBK and University of Trento.
- MathSAT supports a wide range of theories (including e.g. equality and uninterpreted functions, linear arithmetic, bit-vectors, and arrays).
- It also support basic SAT operators and algorithms.
- More information can be found here: `https://mathsat.fbk.eu/`

Other SAT solvers

Several open-source (simple) SAT solvers:

- Minisat `https://github.com/niklasso/minisat`
- Glucose `https://github.com/audemard/glucose`
- CaDiCaL `https://github.com/arminbiere/cadical`

Annual "Hack track SAT competition" is held to improve SAT solvers (CaDiCaL) through minimal changes:
`https://satcompetition.github.io/2024/tracks.html#hack`

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?level=5&lang=eng`

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?level=5&lang=eng`

Solving SAT is hard, but SAT solvers are very efficient in many cases

Modelling the problem in the right way can make a huge difference!

The DIMACS format

- If we want to use SAT solvers, we need to know the **input** format and the **output** provided by the tool.
- The original input format accepted by SAT solvers 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 7 8
 1   2 0
-2  -4 0
 3   4 0
-4  -5 0
 5  -6 0
 6  -7 0
 6   7 0
 7  -3 0
```

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

The DIMACS format: an example (cont.d)

```
c quinn.cnf
c
p cnf 7 8
 1   2 0
-2  -4 0
 3   4 0
-4  -5 0
 5  -6 0
 6  -7 0
 6   7 0
 7  -3 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 7 8
 1   2 0
-2  -4 0
 3   4 0
-4  -5 0
 5  -6 0
 6  -7 0
 6   7 0
 7  -3 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**

Notice

The formula must be in CNF:

$$(l_{11} \vee l_{12} \vee \dots) \wedge (l_{21} \vee l_{22} \vee \dots) \wedge \dots$$

Any formula can be converted in CNF in linear time.

The DIMACS format: an example (cont.d)

```
c quinn.cnf
c
p cnf 7 8
 1   2 0
-2  -4 0
 3   4 0
-4  -5 0
 5  -6 0
 6  -7 0
 6   7 0
 7  -3 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 7 8
 1   2 0
-2  -4 0
 3   4 0
-4  -5 0
 5  -6 0
 6  -7 0
 6   7 0
 7  -3 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:

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

MathSAT 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**. One (of possibly many) assignment satisfying the problem can be printed if requested.

MathSAT 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**. One (of possibly many) assignment satisfying the problem can be printed if requested.

```
./mathsat -input=dimacs -model quinn.cnf
s SATISFIABLE
v -1 2 3 -4 5 6 7 0
```

Outline

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

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 must 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

Using DIMACS could be problematic...

In Exercise 1.1, we had to rewrite a non-CNF formula to feed it to the SAT solver.

What happens if the problem to encode is drastically bigger?

- Solution 1: use scripts to map variables into their index and apply CNF-ization...
  $\Longrightarrow$ boring and error-prone :(

Using DIMACS could be problematic...

In Exercise 1.1, we had to rewrite a non-CNF formula to feed it to the SAT solver.

What happens if the problem to encode is drastically bigger?

- Solution 1: use scripts to map variables into their index and apply CNF-ization...
  $\Longrightarrow$ boring and error-prone :(
- Solution 2: use SMT-LIB, another format most SMT solvers accept!

MathSAT input format: SMT-LIB

- SMT-LIB is an international initiative aimed at facilitating research and development in SMT, the language is only one of several goals achieved.
- All details about SMT-LIB can be found here: `https://smtlib.cs.uiowa.edu/`
- SMT-LIB is more powerful than you could expect. We will discover its potential during the course.

General structure of a SMT-LIB file

A typical SMT-LIB file is characterized by the following sections:

- The *Option* section
- The *Declaration* section
- The *Assertion* section
- The *Action* section

SMT-LIB file: option

- The header of the file can contain some commands to enable some additional functionalities, such as:
  - Generation of models $\Longrightarrow$ `(set-option :produce-models true)`
  - Extraction of UNSAT cores
  - Extraction of interpolants
  - Set background logic for more efficient computations
- While solving the exercises we will highlight the most popular options and their effects.

Warning

Each command must be written inside brackets!

SMT-LIB file: declaration

- In this section we must declare each variable/function necessary to describe the problem.
- The declaration of variables can be done in the following way:

```
(declare-const <name> <type>)
```

- Types supported by SMT-LIB are:
  - `Bool`
  - `Int`
  - `Real`
  - `(_ BitVec <size>)`
  - `(Array <type> <type>)`

  For the moment, we will only use `Bool`
- The declaration of functions can be done in the following way:

```
(declare-fun <name> ([input types]) <type>)
```

SMT-LIB file: assertion

- Once the variables have been defined, we need to encode the constraints in the form of assertions:

```
(assert <condition>)
```

- Conditions can be basic (i.e. (x or y)) or nested ((x and y) or ((not x) and y)).

Warning

In SMT-LIB operators always use a prefix notation!

SMT-LIB assertion: propositional logic

Boolean operators are available to use:

- NEGATION is represented as `(not <formula>)`
- OR is represented as `(or <formula1> <formula2>)`
- AND is represented as `(and <formula1> <formula2>)`
- IMPLIES is represented as `(=> <formula1> <formula2>)`.
- XOR can be represented as `(xor <formula1> <formula2>)`
- IFF is represented as `(= <formula1> <formula2>)`

Warning

The `and` and `or` operators are not only binary operators and can be used with multiple arguments.

SMT-LIB file: action

- The bottom part of the file should describe the task the solver has to manage.
- First you should check satisfiability of the actual problem:

```
(check-sat)
```

- We can then ask for a satisfying assignment:

```
(get-model)
```

- Lastly we end the file using:

```
(exit)
```

Then we can run MathSAT as:

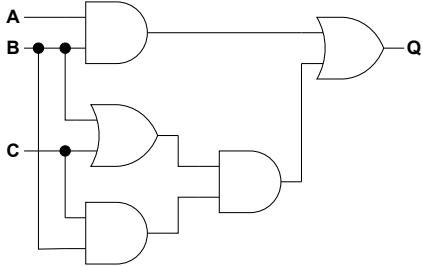
```
./mathsat quinn.smt2
```

Testing Boolean circuits

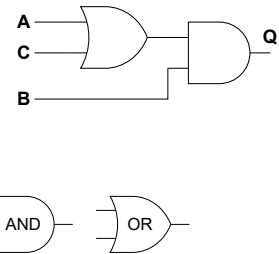
Exercise 1.2

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?

Original circuit:

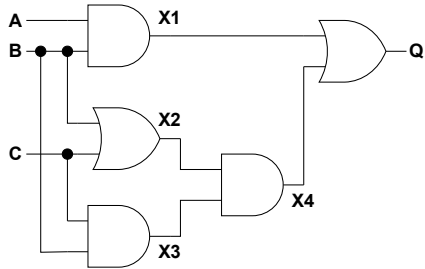


Simplified circuit:



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:

$$X1 \leftrightarrow A \wedge B$$

If we use DIMACS, this formula must be converted into its equivalent CNF form. In this case:

$$\begin{aligned} & (X1 \rightarrow (A \wedge B)) \wedge (X1 \leftarrow (A \wedge B)) \\ \equiv & (\neg X1 \vee (A \wedge B)) \wedge (\neg (A \wedge B) \vee X1) \\ \equiv & (\neg X1 \vee A) \wedge (\neg X1 \vee B) \wedge (\neg A \vee \neg B \vee X1) \end{aligned}$$

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:

$$\begin{aligned}
&(X2 \rightarrow (B \vee C)) \wedge (X2 \leftarrow (B \vee C)) \\
\equiv&(\neg X2 \vee (B \vee C)) \wedge (\neg(B \vee C) \vee X2) \\
\equiv&(\neg X2 \vee B \vee C) \wedge ((\neg B \wedge \neg C) \vee X2) \\
\equiv&(\neg X2 \vee B \vee C) \wedge (\neg B \vee X2) \wedge (\neg C \vee X2)
\end{aligned}$$

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:

$$
\begin{aligned}
&\neg((O1 \rightarrow O2) \wedge (O2 \rightarrow O1)) \\
\equiv &\neg((\neg O1 \vee O2) \wedge (\neg O2 \vee O1)) \\
\equiv &((O1 \wedge \neg O2) \vee (O2 \wedge \neg O1)) \\
\equiv &(O1 \vee O2) \wedge (O1 \vee \neg O1) \wedge (\neg O2 \vee O2) \wedge (\neg O2 \vee \neg O1)
\end{aligned}
$$

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

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

Testing Boolean circuits (cont.d)

- We can now pass the encoding to the solver... after we map each variable into indexed integers!
- Yes, SAT solving is starting to get spicy using DIMACS... What if we use SMT-LIB?

Testing Boolean circuits (cont.d)

- We can now pass the encoding to the solver.
- 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/SMT-LIB 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}) sits in chair j ($j \in$ 1,2,3)
- In this case $3 \cdot 3 = 9$ variables are needed

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

$$\neg(x_{A1} \wedge x_{C2}) \wedge \neg(x_{C1} \wedge x_{A2}) \wedge \neg(x_{A2} \wedge x_{C3}) \wedge \neg(x_{C2} \wedge x_{A3})$$

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

$$\neg x_{A1}$$

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

$$\neg(x_{B2} \wedge x_{C1}) \wedge \neg(x_{B3} \wedge x_{C2})$$

Sorting people: properties (2)

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

Sorting people: properties (3)

- Guest A must sit in at least one chair:

$$x_{A1} \vee x_{A2} \vee x_{A3}$$

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

$$x_{B1} \vee x_{B2} \vee x_{B3}$$

$$x_{C1} \vee x_{C2} \vee x_{C3}$$

Sorting people: properties (4)

- Guest A must sit in at most one chair:

$$x_{A1} \rightarrow (\neg x_{A2} \wedge \neg x_{A3})$$

$$x_{A2} \rightarrow (\neg x_{A1} \wedge \neg x_{A3})$$

$$x_{A3} \rightarrow (\neg x_{A1} \wedge \neg x_{A2})$$

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

The last two conditions can be encoded similarly to the first one, thus we encode them again simply changing the involved variables.

Sorting people: properties (5)

- Only one person can sit on the first position:

$$x_{A1} \rightarrow (\neg x_{B1} \wedge \neg x_{C1})$$

$$x_{B1} \rightarrow (\neg x_{A1} \wedge \neg x_{C1})$$

$$x_{C1} \rightarrow (\neg x_{A1} \wedge \neg x_{B1})$$

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

Cracking codes

Exercise 1.4

You have to guess a two digits code, with digits from 1 to 4. You have three hints:

- In 12, one number is correct and well placed.
- In 14, nothing is correct.
- In 43, one number is correct but wrongly placed.

Does a solution exist? Is it unique?

Cracking codes: variables

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

- x_{ij} states if number i is placed in position j, with $i \in \{1, 2, 3, 4\}, j \in \{1, 2\}$.
- In this case $4 \cdot 2 = 8$ 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.

$$(x_{11} \wedge \neg x_{21} \wedge \neg x_{22}) \vee (x_{22} \wedge \neg x_{11} \wedge \neg x_{12})$$

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

$$\neg x_{11} \wedge \neg x_{12} \wedge \neg x_{41} \wedge \neg x_{42}$$

Cracking codes: properties (3)

Let's encode 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.

$$(x_{31} \wedge \neg x_{41} \wedge \neg x_{42}) \vee (x_{42} \wedge \neg x_{31} \wedge \neg x_{32})$$

Cracking codes: properties (4)

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.

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

  This condition 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 the SAT solver 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 the SAT solver 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.

Cracking codes: results

Once we map the variables into the usual indexed integers we can feed it to the SAT solver 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:

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

Cracking codes: results

Once we map the variables into the usual indexed integers we can feed it to the SAT solver 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:

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

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

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 a SAT solver.

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?

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.