Formal Method Mod. 1 (Automated Reasoning)

Laboratory 3

Giuseppe Spallitta
giuseppe.spallitta@unitn.it

Università degli studi di Trento

March 23, 2022

Outline

1. Satisfiability Modulo Theories
   Quick overview on MathSAT

2. Getting used with SMT

3. Simple real-life applications

4. Homework

MathSAT

- MathSAT 5 is an efficient Satisfiability modulo theories (SMT) 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).
- More information can be found here: `https://mathsat.fbk.eu/`
- Some of the next slides will be redundant, but at least you have a single presentation showing the most used operations with the tool.

SMT-LIB file: option

- The header of the file can contain some commands to enable some additional functionalities, such as:
  - Generation of models

    ```
    (set-option :produce-models true)
    ```

  - Extraction of UNSAT cores

    ```
    (set-option :produce-unsat-cores true)
    ```

  - Extraction of interpolants

    ```
    (set-option :produce-interpolants true)
    ```

  - Set background logic for more efficient computations

    ```
    (set-logic <logic>)
    ```

- While solving the exercises we will highlight the most popular options and their effects.

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>)
- The declaration of functions (both **interpreted** and **uninterpreted**) can be done in the following way:

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

SMT-LIB file: assertion

- Once defined the variables, it is necessary to determine the constraints that rules the satisfiability of the problem in the form of assertions.
- The declaration of assertions can be done in the following way:

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

- Conditions can be basic (i.e. x = 5) or nested (x=2 or x=5).

Warning

In SMT-LIB operators always use a prefix notation!

SMT-LIB assertion: propositional logic

Of course Boolean operators are available to use:

- NEGATION is represented as *(not <var>)*
- OR is represented as *(or <var1> <var2>)*
- AND is represented as *(and <var1> <var2>)*
- IF is represented as *(=> <var1> <var2>)*.
- XOR can be represented as *(xor <var1> <var2>)*
- EQUALITY is represented as *(= <var1> <var2>)*

Warning

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

SMT-LIB assertion: arithmetic

The SMT-LIB format standardizes syntax for arithmetic over integers and over reals.

- ADDITION is represented as +
- SUBTRACTION is represented as -
- MULTIPLICATION is represented as *
- DIVISION is represented by / (Real) and`div` (Int)
- REMAINDER (only using Int) is represented as `mod`
- Relations among variables (i.e. greater (or equal) than, lower (or equal) than) are represented respectively by > (>=) and < (<=)

Warning

The $*$ and + operators are not only binary operators and can be used with multiple arguments.

SMT-LIB assertion: Bit Vectors

Numbers can be represented using a bit vector representation and require different operators

- ADDITION is represented as *bvadd <var1> <var2>*
- SUBTRACTION is represented as *bvsub <var1> <var2>*
- MULTIPLICATION is represented as *bvmul <var1> <var2>*
- DIVISION is represented *bvudiv <var1> <var2>*
- REMAINDER is represented as *bvurem <var1> <var2>*
- Relations among variables (i.e. greater (or equal) than, lower (or equal) than) are represented respectively by `bvugt` (`bvuge`) and `bvult` (`bvule`)

Warning

If you change the *u* into a *s* for the last two sets of operators, you obtain equivalent operations using signed vectors (thus changing the range of admitted values).

SMT-LIB assertion: Arrays

- Arrays map an index type to an element type (similarly to Python `dict` type).
- To select the element associated to index *i* in array *a* the command to use is the following:

```
(select a i)
```

- To update the element associated to index *i* in array *a* with value *e* the command is the following:

```
(store a i em)
```

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 the model value of some of the constants (in this case x and z):

```
(get-value (x z))
```

- Lastly we end the file using:

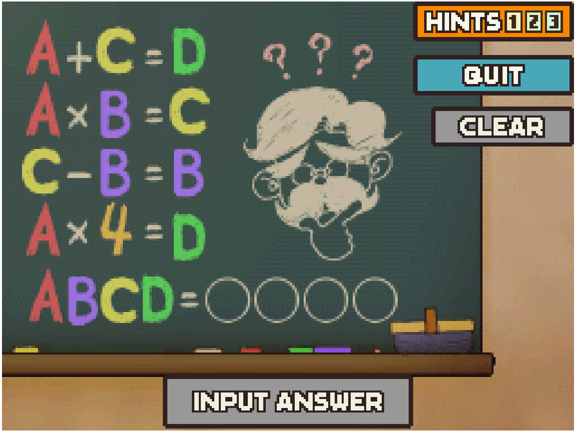
```
(exit)
```

Outline

1. Satisfiability Modulo Theories
2. Getting used with SMT
3. Simple real-life applications
4. Homework

Exercise 3.1: guess the code

A, B, C and D are single-digit numbers. The following equations can all be made with these numbers:



Encoding step-by-step

The procedure to feed a problem into a SMT solver is identical to the one we adopted for SAT problems:

- Identify the variables that can describe the problem.
- Define the assertions to constraints the domains of each variables and check its satisfiability.

The only relevant difference is the expressive power of SMT-LIB with respect to standard SAT.

First encodings: variables

- Reading exercise 3.1, we requires 4 constants: A, B, C and D
- Since they are single digit numbers, we set them as *Int*.
- No additional functions are required for this exercise.

First encodings: assertions

- We must encode the 4 equations that are written on the blackboard, using the basic arithmetical operators.
- Moreover we must ensure that all the digits are different: we can use the command `distinct` to easily encode it. If you don't remember it during the exam don't worry, you can encode it by hand...

First encodings: output

- Once we add the final action, we can feed it to the SMT solver.
  $\Rightarrow$ The solver returns SAT
- If we want to know the values of the variables, we have to add some options and some additional actions.

Additional task

Can you write a simple function to evaluate the maximum among 10 values?

- Maybe creating an arity 10 function is not that easy...
- Try to decompose the problem: **modularity** is the key to win!

Outline

1. Satisfiability Modulo Theories
2. Getting used with SMT
3. Simple real-life applications
   - Geometric exercises
   - SAT/SMT functionality: ALLSAT/ALLSMT
4. Homework

Solving geometric problems

Exercise 3.2: intersecting lines

Given two points in the Euclidean space (i.e. A(1,3) and B(2,7)), let's define an encoding to determine the lines passing from both points and the value x where the line intersect the x-axis.

Solving geometric problems: variables

- We can set 4 variables to store the coordinates of each point (xa, ya, xb, yb).
- We need also to define a function variable (we will call it f) with arity 1, so that we can have an analytical representation of the line.
- A line is represented by the formula:

$$f(x) = mx + q$$

Thus we need other two variables. In addition, f is an interpreted function (we know its behaviour).

Solving geometric problems: assertions (1)

- We start defining 4 assertions to set the value of the coordinates and one assertion to define the line equation:

  ```
  (define-fun f ([(<var> <type>)) <out-type> <func>)
  ```

- Then we can encode two assertions to calculate the values of m and q using the analytic formulae:

$$m = \frac{yb - ya}{xb - xa}$$

$$q = ya - m * xa$$

Solving geometric problems: assertions (2)

- Now an assertion to update the analytic function f using the calculated parameters is necessary.
- Lastly we intersect the generic line with the equation of the x-axis, which is:

  $$y(x) = 0$$

  We need to store the value of the horizontal intersection, so we can add a novel variable that will store the solution of this last assertion.

Solving geometric problems: results

- Now we can feed the encoding into MathSAT, obtaining a valid solution.
- The problem can be easily adapted to different sets of points: if we change the coordinates, we will obtain a different line.
- You can also extend this code to generalize this exercises in the case you want to determine the intersection of two lines.

Unlocking phones

UNIVERSITÀ DEGLI STUDI DI TRENTO

Exercise 3.3: unlocking phones

You wants to unlock the mobile phone of your friend to see if they are dating someone. Sadly, there is a 2*2 grid pattern lock that stops you. You remember that the password requires all 4 pins to be connected; moreover there are no diagonal lines in the pattern. How many combinations you have to try in the worst case to unlock the phone?

Unlocking phones: variables

- This exercise can be modeled as a SAT problem, so we can reason in the same way as the first laboratories.
- In particular we need 16 variables, labeled x_{ij}, where i is the cell in the grid and j is the order in the sequence.

Unlocking phones: assertions

- For each cell in the grid, exactly one temporal position in the sequence is correct.
- For each temporal position in the sequence, exactly one cell in grid must be chosen.
- If a cell in the grid is chosen, we must ensure that the next one is not the diagonal one.

Unlocking phones: results

- If we simply run the *(check-sat)* command we will see that the problem is SAT (thus at least one password exists), but we are interested in knowing the total number of solutions admitted...
- The *(check-allsat)* command returns all possible solution given a set of Boolean variables (if no set is passed as arguments, all the defined Boolean variables are considered). Thanks to it, we can see how many solutions can be generated.

Outline

Homework

Homework 3.1: math olympics

Find the number of positive integers with three not necessarily distinct digits, abc, with $a \neq 0$ and $c \neq 0$ such that both abc and cba are multiples of 4.

Homework

Homework 3.2: balance puzzle

Solve it using an SMT solver (use some temporary variables to store the possible solutions...)