UNIVERSITÀ
DI TRENTO

Formal Method Mod. 1 (Automated Reasoning)

Laboratory 5

Giuseppe Spallitta
giuseppe.spallitta@unitn.it

Università degli studi di Trento

April 14, 2021

UNIVERSITÀ DEGLI STUDI
DI TRENTO

Outline

1. Advanced SMT solving
   - Cybersecurity applications
   - Computing interpolants
   - Formal verification of algorithms
2. Homeworks

Black hat hacker

UNIVERSITÀ DEGLI STUDI DI TRENTO

Exercise 5.1: hacking key

You want to access the UniTN database. Sadly the server is protected by a key. From reverse engineering you obtain the following part of code executed by the machine:

```
% Key is the concat of 3 32-bit numbers a,b and c
assert(isMultiple(a,5))
assert(or(a,b) == 2021))
assert(a - b > 1000)
assert(isAverage(c, [a,b]))
assert(c<0x76543210)
login()
```

Given you have one opportunity to log in and that if you fail you will be expelled, can you guess the key?

Black hat hacker: variables

UNIVERSITÀ DEGLI STUDI DI TRENTO

As always, we first define the variables that efficiently describe the problem:

- 3 variables are necessary to store the three sub-parts of the entire key.
- The comment highlights that they are Bit vectors, so the type is also clearly defined.

Black hat hacker: functions

UNIVERSITÀ DEGLI STUDI DI TRENTO

- No function is mandatory for this problem; the two high-level operations can be encoded as functions if desired.
- *isMultiple* can be defined as a 2-arity function **(BitVector, Int)** $\Rightarrow$ **Bool**.
- *isAverage* can be defined as a 3-arity function **(BitVector, BitVector, BitVector)** $\Rightarrow$ **Bool**.

Black hat hacker: properties

- Properties are trivial for most of the part, since they simply require to encode the content of the Python instructions *assert*.
- Be careful: we work with bit vectors, so do not forget to use the correct operators.
- Moreover, be sure that integers used as constants are also treated as bit vector (MathSAT does not provide implicit type conversion :( )

UNIVERSITÀ DEGLI STUDI DI TRENTO

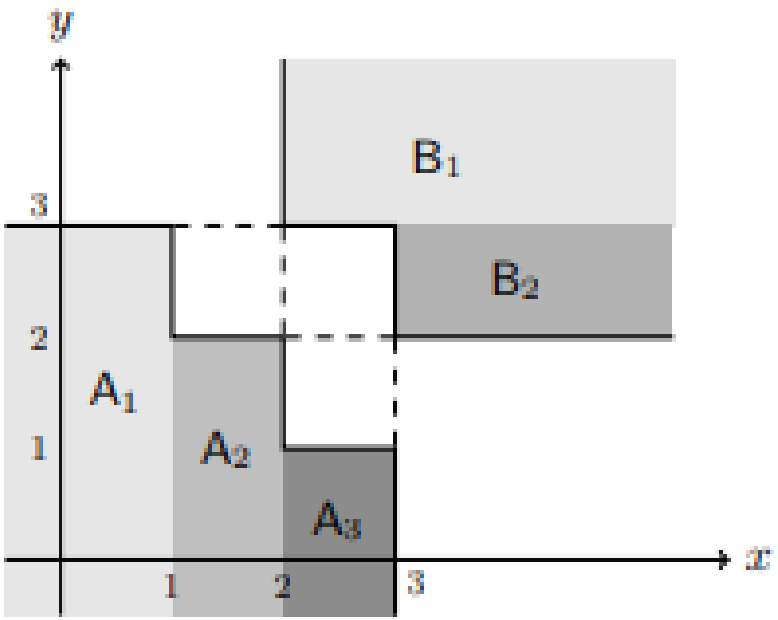
Black hat hacker: constant conversion

- The simplest alternative is directly setting the number using the instruction:
  (_ bv<number> <size>)
- But when we manage negative numbers, `bv-1` does not work, so we require a different instruction, which maps integers to their equivalent BV representation assuming the size chosen is high enough:
  (_ to_bv 3) (- 2)
- You can also write numbers using the hexadecimal or binary representation (convenient when dealing with low numbers of bits) using prefixing respectively *#x* or *#b*.

Interpolants of areas

UNIVERSITÀ DEGLI STUDI DI TRENTO

Exercise 5.2: geometric interpolants



Compute, if existing, the interpolant between the ordered pair (A, B), where the upper case character represent the set of all indexed sub-areas?

Interpolants of areas: variables

UNIVERSITÀ DEGLI STUDI DI TRENTO

As always, we first define the variables that efficiently describe the problem:

- First we must define two `Real` variables to represent the coordinates of the Euclidean system, x and y.
- We must also create several variables to store the area covered by each section shown in the figure (A_1, A_2, A_3, B_1, B_2). Each variable will be of type `Bool`.
- Lastly, we define the two generic variables A and B, both `Bool`, to considering the union of their respective sub-areas.

UNIVERSITÀ DEGLI STUDI DI TRENTO

Interpolants of areas: properties

- Each smaller area can be described using simple constraints in the form *var* $< \alpha$ and *var* $> \alpha$.
- The entire area defined by a single character can be considered as the disjunction of all its smaller areas.
- A quick way to declare a variable and set its value equal to something is the sugar sintax *(define-fun <name> () <type> <condition>)*

Interpolants of areas: is it UNSAT?

- To test the actual satisfiability of the problem, we must activate some of the variables using assertions (in our case A and B, but you can also consider single small areas).
- Running *(chech-sat)*, the problem is UNSAT
  $\Rightarrow$ we can find an interpolant to this problem!

Interpolants of areas: interpolants using MathSAT

- Similarly to UNSAT core, an additional option is required to activate this functionality: *(set-option :produce-interpolants true)*
- Each variable and/or condition that belongs to the first element of the ordered pair should be added to an interpolant group, using the corresponding assertion *(assert (! <name> :interpolation-group <group-name>))*.
- These groups must be called as argument of the *(get-interpolant (<group-name>))* action.

Checking algorithms

UNIVERSITÀ DEGLI STUDI DI TRENTO

Exercise 5.3: pair programming

Given the following function to compute the greatest common divisor can you formally check if, given two random numbers, a solution is obtained under 5 iterations?

```
int GCD(int x, int y){
    while(true) {
        int m = x % y;
        if (m == 0) return y;
        x = y ;
        y = m;
    }
}
```

Checking algorithms: variables

UNIVERSITÀ DEGLI STUDI DI TRENTO

As always, we first define the variables that efficiently describe the problem:

- We need to store the value of the three variables m, x and y and their evolution during several iterations.
- Using simple `Int` variables does NOT work, because in the end we would ask the same variable to assume multiple values at the same time.
- We can instead use `Array` mapping the index of iteration to its value at that moment: *Array Int Int*

Checking algorithms: properties (1)

UNIVERSITÀ DEGLI STUDI DI TRENTO

- We first must initialize the first elements of each array, setting the input values and the first value of m as the remainder.
- The *If* line requires to define two assertions, depending of the value of m. They will simulate the behaviour of the conditional instruction.

Checking algorithms: properties (2)

UNIVERSITÀ DEGLI STUDI DI TRENTO

- If $m = 0$, then we already found a solution and we can stop.
- Otherwise, we must compute the new values for the second iteration and update the array.
- We must iterate this process until the arrays are used for five times. If at that moment m is still not 0, we can return false so that we prove its unsatisfiability.
- To easily retrieve the solution when the solver returns SAT, we can create an additional variable to store the value of y once we reached the end of the loop.

Checking algorithms: properties (3)

UNIVERSITÀ DEGLI STUDI DI TRENTO

- The current idea does not work as expected: some values of the arrays seem to be randomly computed, so I may wonder about correctness.
- Remember that the assertion are NOT executed sequentially, but they must. Working with Int variables, there could be cases where no constraints are actually active on them and so the solver can freely choose a value to assign.
- Can we check step by step
- To easily retrieve the solution when the solver returns SAT, we can create an additional variable to store the value of y once we reached the end of the loop.

UNIVERSITÀ DEGLI STUDI DI TRENTO

SMT Model Checking?

- Thanks to this encoding we were able to prove if a solution is found in n steps: this is known as *Bounded Model Checking*.
- A BMC problem usually requires:
  - An initial state I.
  - A transition relation T to move among different steps, considering n transitions.
  - A final state to reach F.
- In our case the initialization of the arrays x and y is part of I, while T is defined as the various branches depending on the conditional statement.

UNIVERSITÀ DEGLI STUDI DI TRENTO

SMT Model Checking?

- The case we considered is not exactly a BMC problem (there is no clear final state, it depends on the number of assertions computed by the solver), but it can be seen as a generalization of it.
  $\Rightarrow$ Encoding BMC problems is easier, since the final state is usually trivial and the main issue is formally encoding the transition among states.
- If you want to know the general correctness of the algorithm, without upper boundaries, SMT is not your ideal tool...
  $\Rightarrow$ If you are interested, the second part of the Formal Method laboratories will cover it ;)

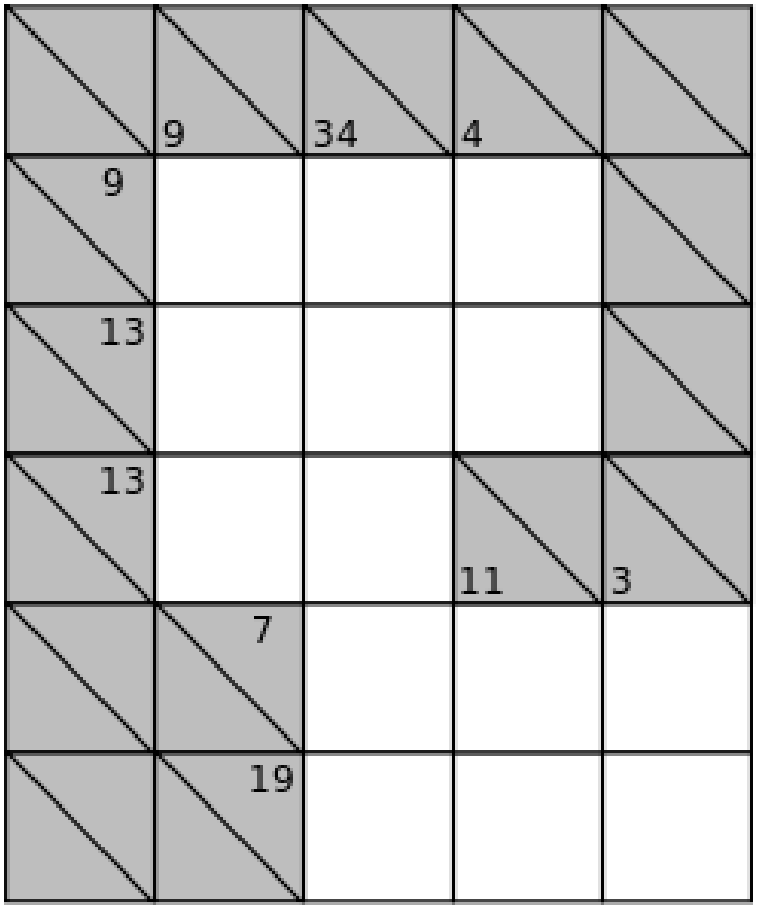
UNIVERSITÀ DEGLI STUDI
DI TRENTO

Outline

1. Advanced SMT solving
2. Homeworks

Solving Kakuro

Homework 5.1: kakuro



Kakuro is a puzzle in which one must put the numbers 1 to 9 in the different cells such that they satisfy certain constraints. If a clue is present in a row or column, the sum of the cell for that row should be equal to the value. Within each sum all the numbers have to be different, so to add up to 4 we can have 1+3 or 3+1. Can we find a solution using SMT solvers?

Homeworks

UNIVERSITÀ DEGLI STUDI DI TRENTO

Homework 5.2: checking professor's interpolants

Apply the extraction of interpolants using MathSAT to see if we can obtain the solution of the exercise shown in slide 111 (handouts) of this presentation: `http://disi.unitn.it/~rseba/DIDATTICA/fm2021/SLIDES/03-smt_handouts.pdf`

Homeworks

Homework 5.3: Collatz conjecture

Given a number x, check if the following algorithm that describe the Collatz conjecture ends in 5 turns (including the first one where it checks the current number):

```
def conjecture(x)
    while(true):
        if x == 1:
            break
        if isOdd(x):
            x = x * 3 + 1
        elif isEven(x):
            x = x / 2
    return SAT
```