UNIVERSITÀ
DI TRENTO

Formal Method Mod. 1 (Automated Reasoning)

Laboratory 4

Giuseppe Spallitta
giuseppe.spallitta@unitn.it

Università degli studi di Trento

March 23, 2023

UNIVERSITÀ DEGLI STUDI
DI TRENTO

Outline

1. Advanced SMT solving
   - Cybersecurity applications
   - Private investigations
   - Formal verification of algorithms
2. Homeworks

Black hat hacker

UNIVERSITÀ DEGLI STUDI DI TRENTO

Exercise 4.1: 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) == 2022))
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?

UNIVERSITÀ DEGLI STUDI DI TRENTO

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*.

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**.
- Other than that,the encoding is simply to write using SMT-LIB functions.

Private investigations

UNIVERSITÀ DEGLI STUDI DI TRENTO

Exercise 4.2: who killed Agatha?

Someone who lives in Dreadbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein. A killer always hates his victim, and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Aunt Agatha hates. No one hates everyone. Who killed Agatha?

First approach: without quantifiers

- MathSAT does not natively support quantifiers!
- If we iterate through all variables, we can simulate the behaviour of $\exists$ and $\forall$ thanks to the **Shannon expansion**.
- Can we use functions that maps variables to a truth value, in order to set if a relation holds?

Private investigations: variables

UNIVERSITÀ DEGLI STUDI DI TRENTO

First we must define constants, predicates and functions that efficiently describe the problem:

- **Constants**: agatha, butler, charles (for each person, I use a different `Int`.
- **Predicates/Functions**: hates(X,Y), richer(X,Y)

The killer will be an `Int` variable, constrained by all the conditions stated in the problem and the hidden ones.

UNIVERSITÀ DEGLI STUDI DI TRENTO

Private investigations: SMT formulae (1)

- A killer always hates his victim...
  $hates(killer, agatha))$
- and is never richer than his victim.
  $\neg\ richer(killer, agatha))$
- Charles hates no one that Aunt Agatha hate.
  $\forall x.(hates(agatha, x) \rightarrow \neg\ hates(charles, x)))$

Private investigations: SMT formulae (2)

UNIVERSITÀ DEGLI STUDI DI TRENTO

- Agatha hates everyone except the butler.
  $\forall x.(x \mathrel{!}= butler \rightarrow (hates(agatha, x))$
- The butler hates everyone not richer than Aunt Agatha
  $\forall x.(\neg\ richer(x, agatha) \rightarrow (hates(butler, x))$
- The butler hates everyone Aunt Agatha hates.
  $\forall x.(hates(agatha, x) \rightarrow hates(butler, x)))$
- No one hates everyone.
  $\forall x \exists y.(\neg\ hates(x,y))$

Private investigations: hidden conditions

UNIVERSITÀ DEGLI STUDI DI TRENTO

Richer is a function such as:

- It is non-reflexive:
  $\forall x.(\neg richer(x, x))$
- It is non-symmetric:
  $\forall xy.(richer(x, y) \leftrightarrow \neg richer(y, x))$

UNIVERSITÀ DEGLI STUDI DI TRENTO

Defining functions

- Every time you define a function mapping one or more values to a Boolean or a sorted type, be sure to encode also its hidden properties; this could drastically change the behaviour of the solver!
- A brief list includes:
  - (non-)Simmetry
  - (non-)Reflexivity
  - (non-)Transitivity

UNIVERSITÀ DEGLI STUDI
DI TRENTO

Second approach: with quantifiers

- Z3 supports quantifiers!
- pysmt offers the two shortcuts: **ForAll** e **Exists**, accepting the list of quantified variables and the formula.
- Can we use Z3 to easily encode the same problem?

Checking algorithms

UNIVERSITÀ DEGLI STUDI DI TRENTO

Exercise 4.3: 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;
    }
}
```

SMT Model Checking?

- Every time we must check something happening at a specific iteration, we could think of encoding it as a *Bounded Model Checking* (BMC) problem.
- 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.

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)

- 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.
- We must encode conditions such that the DPLL search is bound to fix some values for each step, partially simulating the sequentiality.
- 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.

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 ;)

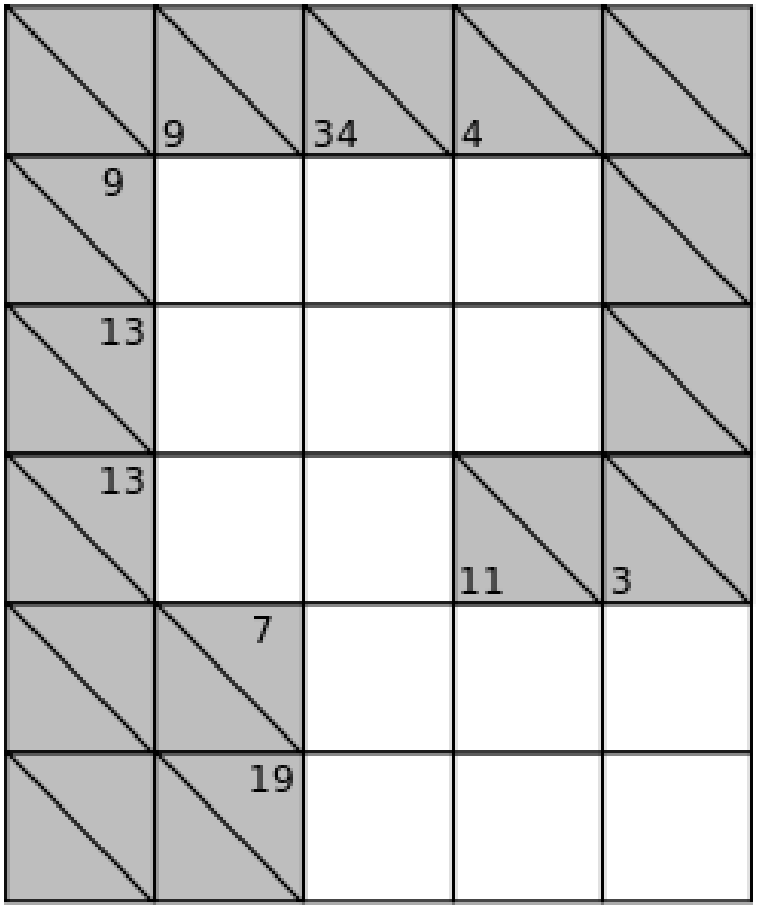
UNIVERSITÀ DEGLI STUDI
DI TRENTO

Outline

1. Advanced SMT solving
2. Homeworks

Solving Kakuro

Homework 4.1: 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

Exercise 4.2: task manager

Your PC needs to complete 5 different tasks (A,B,C,D and E) to correctly save a file. There are some constraints about the order execution of the tasks:

- We can execute A after D is completed.
- We can execute B after C and E are completed.
- We can execute E after B or D are completed.
- We can execute C after A is completed.

Which is the task that will execute for last?

Homeworks

Homework 4.3: 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
```