UNIVERSITÀ
DI TRENTO
Department of
Information Engineering and Computer Science

Automated Reasoning and Formal Verification

Laboratory 4

Gabriele Masina
gabriele.masina@unitn.it
https://github.com/masinag/arfv2025

Università di Trento

March 26, 2025

Outline

1. Advanced SMT solving
   - Cybersecurity applications
   - Pseudo-Boolean for SMT solving
   - Automating the Encoding — Dealing with quantifiers
2. Homeworks

Black Hat Hacker

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 this part of code executed by the machine:

```
# the key is the concat of 3 32-bit numbers a, b and c
assert isMultiple(a,5)
assert a | b == 2022
assert a - b > 1000
assert isAverage(c, [a,b])
assert a * c <= 0x0017c1cc
login()
```

You have only one opportunity to log in, can you guess the key?

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

Black hat hacker: constant conversion

MathSAT does not provide implicit type conversion of integers to bit vectors.

- The simplest alternative is directly setting the number using the instruction:

  `(_ bv<number> <size>)`

  E.g.: `(_ bv2022 32)`

- 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 <size> <integer>)`

  E.g.: `(_ to_bv 3) (- 2)`

- You can also write numbers using the binary or hexadecimal representation using the the prefix `#b` or `#x`, respectively.

  E.g., to represent 143 in bit-vector of size 8, you can write: `#b01001111` or `#x8F`

Black hat hacker: variables

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

- 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 `(BitVec, Int)` $\mapsto$ `Bool`.
- `isAverage` can be defined as a 3-arity function `(BitVec, BitVec, BitVec)` $\mapsto$ `Bool`.
- Other than that,the encoding is simply to write using SMT-LIB functions.

Checking Algorithms

Exercise 4.2: Check Security... Again

This code that analyzes a 5-digit number in $[10000, 99999]$ with distinct digits:

```
void check_security(num) {
    security = 4
    if (num is multiple of 3 or 5, but not both)
        security = security - 1
    if (the sum of digits is a multiple of 10)
        security = security - 1
    if (the number is palindrome)
        security = security - 1
    if (the digits are in ascending order (including equality))
        security = security - 1
    return security
}
```

Is there an input that provides the security value 2. Answer using MathSAT.

Pseudo-Boolean variables for Cardinality Constraints

- We can express `ExactlyTwo` constraints with a Boolean variable for each condition. But what if we had `ExactlyFour`? The number of cases grows very fast!
- A common way to express cardinality constraints (i.e., *exactly/at most/at least k conditions are true*) is to use **Pseudo-Boolean variables**.
- A Pseudo-Boolean variable is an `Int` variable taking values in $\{0, 1\}$ only. It represents a Boolean value, but it can be used to *count*.
- We can have a counter incremented by 1 by every verified condition B_i.

  Exactly k : $\sum_{i=1}^{n} B_i = k$
  At most k : $\sum_{i=1}^{n} B_i \leq k$
  At least k : $\sum_{i=1}^{n} B_i \geq k$

Pseudo-Boolean variables: How To Implement It

- **Solution 1**: define each `Bool` variable as `Int`, bounding them to the range $[0, 1]$.
- **Solution 2**: define each variable `B` as `Bool` and then use a `sum` of `(ite B 1 0)` to easily get their combined sum.

Encoding SMT Problems with PySMT

Solving SMT problems with PySMT: https://pysmt.readthedocs.io/en/latest/

- We can define variables of different types:

```
a = Symbol("a", BOOL)
x = Symbol("x", REAL)
y = Symbol("y", INT)
```

- We can express theory constraints:

```
phi  = Or(a, GE(Plus(x, ToReal(y)), Real(3.5)))
phi1 = a | (x + ToReal(y) >= 3.5) # also with infix operators!
```

- Solvers are invoked in the usual way:

```
with Solver("msat") as solver:
    solver.add_assertion(phi)
    if solver.solve():
        print(solver.get_model())
```

Private investigations

Exercise 4.3: Who killed Agatha?

You are a private investigator and you have been called to solve a crime. When you arrive at the crime scene, the police have already established the following facts:

- Someone who lives in Dreadbury Mansion killed Aunt Agatha.
- Agatha, the butler, and Charles are the only ones living in Dreadbury Mansion.
- 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.

The police have asked you to find out who killed Aunt Agatha. Can you help them?

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**.
- We will use PySMT to automate the encoding.

Private investigations: variables

First let's define constants, predicates and functions to describe the problem:

Constants : *agatha*, *butler*, *charles* (for each person, we use a different `Int`).

Predicates/Functions : *hates* : $(\texttt{Int}, \texttt{Int}) \mapsto \texttt{Bool}$, *richer* : $(\texttt{Int}, \texttt{Int}) \mapsto \texttt{Bool}$

Variables : *killer* of type `Int`

Private investigations: SMT formulas (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 formulas (2)

- Agatha hates everyone except the butler.
  $\forall x.(x \neq butler \rightarrow hates(agatha, x))$
- 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

Richer is a function such that:

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

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

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?

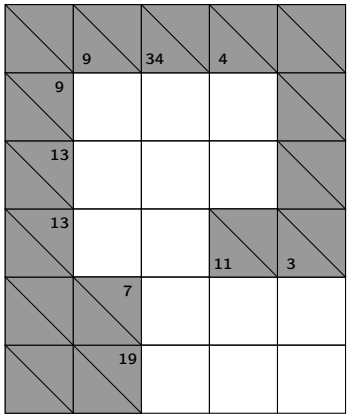
Outline

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?

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?