Practical SMT Session

Aina Niemetz Mathias Preiner

Stanford University

SAT/SMT/AR Summer School 2018
July 3-6, 2018
Manchester, UK

Introduction

In this session we will use PySMT (https://github.com/pysmt/pysmt)

Install locally

```
pip install pysmt
```

```
pysmt-install --btor      # Install Boolector
# If you didn't install cvc4 beforehand, skip this
pysmt-install --cvc4      # Install CVC4
pysmt-install --msat      # Install MathSAT
pysmt-install --z3        # Install Z3
```

```
pysmt-install --env
```

Alternatively, use **VirtualBox**[1] or **Docker**[2] image.

[1] https://drive.google.com/file/d/1PbGEqhGD68AyXLSp-7mjhLtba0VG2sea/view?usp=sharing
[2] https://github.com/pysmt/pysmt-docker

PySMT

PySMT

- a solver-agnostic Python wrapper for SMT
- supports a multitude of solvers

  SMT:

  - Boolector (`http://boolector.github.io`)
  - CVC4 (`http://cvc4.cs.stanford.edu`)
  - MathSAT (`http://mathsat.fbk.eu`)
  - Yices (`http://yices.csl.sri.com`)
  - Z3 (`https://github.com/Z3Prover/z3`)

  SAT:

  - PicoSAT (`http://fmv.jku.at/picosat`)

PySMT

Include Shortcuts and Typing from PySMT

```
from pysmt.shortcuts import *
from pysmt.typing import *
```

- Shortcuts defines wrappers for the most commonly used functions

  https://pysmt.readthedocs.io/en/latest/api_ref.html#module-pysmt.shortcuts

- Typing defines SMT types (sorts)

  https://pysmt.readthedocs.io/en/latest/api_ref.html#module-pysmt.typing

Note: You can also import functions individually:

```
from pysmt.shortcuts import Symbol
from pysmt.typing import INT
```

PySMT - Shortcuts

- `Symbol` create variables and (first order) constants

```
a = Symbol("a")                  # By default sort BOOL
x = Symbol("x", INT)             # Integer sort
b = Symbol("b", BVType(32))      # Bit-vector sort of size 32
```

- `TRUE`, `FALSE`, `Bool`, `Int`, `BV` Theory constants

```
y = Int(2)
z = BV(3, 4)                     # Bit-vector value 3, size 4
```

- `And`, `Or`, `Not`, `Implies`, `Iff` Boolean operators

```
And(LE(y, x), GE(Int(10), x))  #  y ≤ x ∧ 10 ≥ x
```

PySMT - Shortcuts

- `Equals`, `NotEquals`, `AllDifferent` — (Dis)Equality
  `LE`, `LT`, `GE`, `GT` — Inequality
- `Minus`, `Plus`, `Times`, `Div` — Arithmetic operators
  Note: not for bit-vectors!
- `BVAdd`, `BVSub`, `BVMul` — Arithmetic BV operators
  `BVUDiv`, `BVSDiv`
- `BVNot`, `BVAnd`, `BVOr`, `BVXor` — Bit-wise operators
  `BVLShl`, `BVLShr`, `BVAShr`
- `Ite` — If-then-else

PySMT - Typing

- `BOOL` Boolean sort

```
a = Symbol("a")                    # By default sort BOOL
a = Symbol("a", BOOL)
True(), False()                    # Boolean values
```

- `INT` Integer sort

```
x = Symbol("x", INT)               # Integer sort
Int(2)                             # Integer value
```

- `REAL` Real sort

```
y = Symbol("y", REAL)               # Real sort
Real(1.5)                           # Real value: 1.5
Real((3, 2))                        # Real value: 3 / 2
```

PySMT - Typing

- `BVType(size)` Bit-vector sort of given size

```
b = Symbol("b", BVType(32))     # Bit-vector sort of size 32
BV(3, 32)                       # Bit-vector value
```

- `ArrayType(index_type, element_type)` Array sort

```
ArrayType(INT, REAL)
ArrayType(BVType(8), BVType(16))
```

PySMT - Solver Instantiation

```
btor = Solver(name='btor')        # Boolector
cvc4 = Solver(name='cvc4')        # CVC4
msat = Solver(name='msat')        # MathSAT
yices = Solver(name='yices')      # Yices
z3 = Solver(name='z3')            # Z3

btor.add_assertion(...)
```

```
with Solver(name='btor') as solver:
    solver.add_assertion(...)
```

PySMT - Asserting Formulas

```
BV32 = BVType(32)

a = Symbol('a', BV32)
b = Symbol('b', BV32)
c = Symbol('c', BV32)

solver = Solver(name='btor')

solver.add_assertion(Equals(a, b))    # a = b
solver.add_assertion(NotEquals(b, c)) # b != c
...

# Solve a = b && b != c
res = solver.solve()
...
```

PySMT - Example

```
with Solver() as solver:

    a = Symbol('a', INT)
    b = Symbol('b', INT)

    solver.add_assertion(Equals(a, b))

    # assertion 1: a = b
    res = solver.solve()    # SAT (res == True)

    if res:
        print(solver.get_model())
        print('value a: {}'.format(solver.get_value(a)))
        print('value b: {}'.format(solver.get_value(b)))
```

PySMT - Example (cntd.)

```
solver.push()              # Create new context

solver.add_assertion(NotEquals(a, b))

# assertion 1: a = b
# assertion 2: a != b
res = solver.solve()     # UNSAT (res == False)

solver.pop()               # pop context -> pop assertion 2

# assertion 1: a = b
res = solver.solve()     # SAT (res == True)
```

Exercises

Branchless $abs(x)$

Absolute Value $abs(x)$

$$x < 0 \ ? \ -x \ : \ x$$

Prove that the branchless versions of function $abs(x)$ from page 18 of Hacker's delight[3] are correct.

Alternatives of branchless $abs(x)$ **(32 bit)**

$y := x \gg_s 31$ (arithmetic right shift, BVAShr in PySMT)

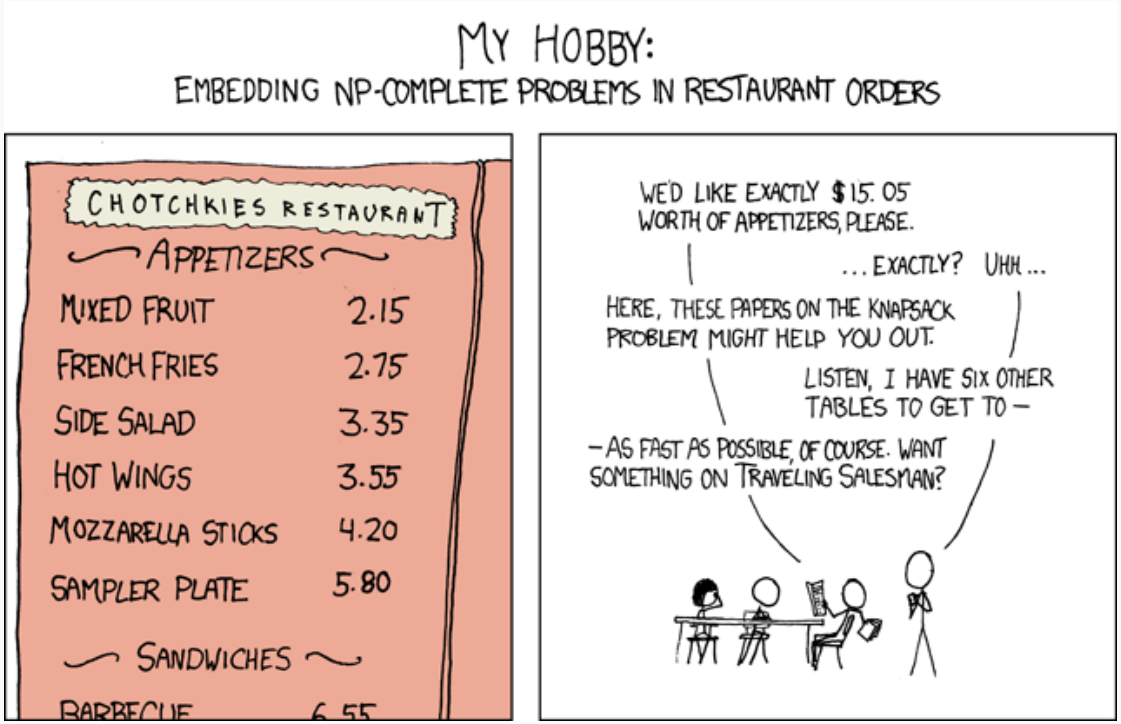
Alternative 1: $(x \oplus y) - y$

Alternative 2: $(x + y) \oplus y$

Alternative 3: $x - ((2 \cdot x) \ \& \ y)$

[3] http://www.hackersdelight.org/basics2.pdf

XKCD 287



https://xkcd.com/287/

How many combinations of appetizers exist that are exactly worth \$15.05? What appetizer combinations are possible?

Note: You can pick more than one appetizer of a kind (5x french fries, . . .).

Sudoku

Fill in the blanks (marked as **STUB**) in sudoku.py.

Sudoku Rules for 3x3

- Each of the 3x3 squares contains numbers 1-9
- Each number can only appear once in each row, column, and square.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Note: sudoku.py should handle 2x2, 4x4, ...

Pseudorandom Number Generator

Given a function *rand*() that generates pseudorandom numbers based on the following **linear congruential generator (LCG)** algorithm[4].

$$X_{i+1} = (1019357 \cdot X_i + 30129) \,\%\, (1 \ll 17)$$

- What is the maximum number of consecutive iterations of *rand*() % 47 that produce the number 42?
- What is the starting seed X_0?

Fill in the blanks (marked as **STUB**) in lcg.py.

C Code Example

```
uint32_t rand(uint32_t x) { return (1019357 * x + 30129) % (1 << 17); }
uint32_t x, x0, n = 0;
x = x0 = ?;
while((x = rand(x)) % 47 == 42) { n++; }
```

[4] https://en.wikipedia.org/wiki/Linear_congruential_generator

Bounded Model Checking

Fill in the blanks (marked as **STUB**) in bmc.py.

Check if safety property P holds for 10 iterations.

- Unroll the loop 10 times or until property P is violated
- Check for each iteration if property P holds

C Code

```
int main () {
  bool turn;                      // input
  uint32_t a = 0, b = 0;          // states
  for (;;) {
    turn = read_bool ();
    assert (a != 3 || b != 3); // property P
    if (turn) a = a + 1;          // next(a)
    else      b = b + 1;          // next(b)
  }
}
```

Quote Martin: "If you like this, you will love https://www.cprover.org/cbmc"

Unroll

$a_0 = 0 \wedge b_0 = 0$

$\ldots$ check if P holds for a_0, b_0

$a_1 = next(a_0) \wedge b_1 = next(b_0)$

$\ldots$ check if P holds for a_1, b_1

$a_2 = next(a_1) \wedge b_2 = next(b_1)$

$\ldots$

More Exercises

For more exercises/examples check out:

- PySMT Tutorial:
  `https://pysmt.readthedocs.io/en/latest/tutorials.html`
- Dennis Yurichev's *SAT/SMT by example*:
  `https://yurichev.com/writings/SAT_SMT_by_example.pdf`