Automated Reasoning 2020

Lecture 4: Encoding into reasoning problems

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2020-08-29

Topic 4.1

Z3 solver

Solver basic interface

- Input : formula
- Output: sat/unsat

If satisfiable, we may ask for a satisfying assignment.

Exercise 4.1
What can we ask from a solver in case of unsatisfiability?

Z3: SMT solver

- Written in C++
- Provides API in C++ and Python
- We will initially use python interface for quick ramp up
- Later classes we will switch to C++ interface

Installing Z3 (Ubuntu-18.04)

```
$sudo apt-get install z3
```

Not tested on 20.04

Locally Installing a version of Z3 (Linux)

Let us install `z3-4.7.1`. You may choose another version.

- Download

  `https://github.com/Z3Prover/z3/releases/download/z3-4.7.1/z3-4.7.1-x64-ubuntu-16.04.zip`

- Unzip the file in some folder. Say

  /path/z3-4.7.1-x64-ubuntu-16.04/

- Update the following environment variables

```
$export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/z3-4.7.1-x64-ubuntu-16.04/bin
$export PYTHONPATH=$PYTHONPATH:/path/z3-4.7.1-x64-ubuntu-16.04/bin/python
```

- After the setup the following call should throw no error

  $python3 /path/z3-4.7.1-x64-ubuntu-16.04/bin/python/example.py

Topic 4.2

Using solver

Steps of using Z3 via python interface

```
from z3 import *        # load z3 library

p1 = Bool("p1")         # declare a Boolean variable
p2 = Bool("p2")

phi = Or( p1, p2 )      # construct the formula
print(phi)              # printing the formula

s = Solver()            # allocate solver
s.add( phi )            # add formula to the solver
r = s.check()           # check satisfiability
if r == sat:
    print("sat")
else:
    print("unsat")      # save the script test.py
                        # run \$python3 test.py
```

Get a model

```
r = s.check()
if r == sat:
    m = s.model()       # read model
    print(m)            # print model
else:
    print("unsat")
```

Exercise 4.2

What happens if we run `m = s.model()` in the unsat case?

Solve and print model

```
from z3 import *

# packaging solving and model printing
def solve( phi ):
  s = Solver()
  s.add( phi )
  r = s.check()
  if r == sat:
      m = s.model()
      print(m)
  else:
      print("unsat")

 # we will use this function in later slides
```

Pointer and variable

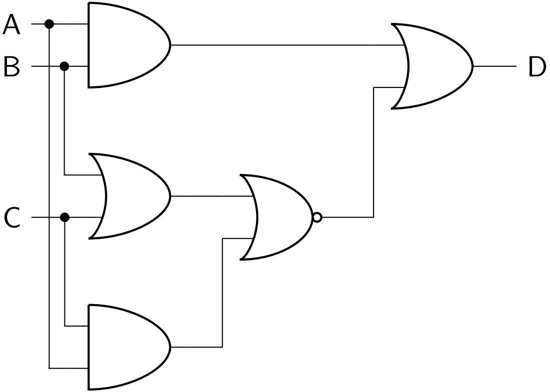
There is a distinction between the Python variable name and the propositional variable it holds.

```
from z3 import *      # load z3 library
x = Bool("y")         # creates Propositional variable y
z = x                 # python pointer z also holds variable y
```

Exercise: encoding Boolean circuit

Exercise 4.3

Using Z3, find the input values of A, B, and C such that output D is 1.



We know you can do it! Please do not shout the answer. Please make computer find it.

Design of solvers: context vs. solver

Any complex software usually has a context object.

The context consists of a formula store containing the constructed formulas.

Z3 Python interface instantiates a default context. Therefore, we do not see it explicitly.

A `Solver` is a solving instance. There can be multiple solvers in a context.

The `Solver` solves only the added formula.

Solving rational(real) arithmetic

```
x = Real('x')
y = Real('y')
phi = And(x + y > 5, x > 1, y > 1)
solve( phi )
```

For linear arithmetic
Real == rational

Solving integer arithmetic

```
x = Int('x')
y = Int('y')
phi = And(x + y > 5, x > 1, y > 1)
solve( phi )
```

Exercise: bounded model checking

Exercise 4.4

Using Z3, find the inputs x and y such that the assert fails.

```
int foo( int x, int y ) {
  int z = 3*x + 2*y - 3;
  if( y > 0 )
    assert( z != 0 );
}
```

Solving bit precise

```
x = BitVec('x',32) # declare name and bit length
y = BitVec('y',32)
phi = And(x + y > 5, x > 1, y > 1)
solve( phi )
```

- Bit lengths must match in an operation
- Largely solved by bit blasting
- Far more expensive to solve!

converting Bit-vector formulas into Boolean formulas by replacing vectors by bits and operation by circuits.

Exercise : observe overflow behavior

Exercise 4.5

Give a bit-vector formula that is satisfiable due to overflow of addition, but in infinite precision it is unsatisfiable.

Uninterpreted functions

```
x = Int('x')
y = Int('y')

# declaring Int -> Int function
h =  Function('h', IntSort(), IntSort() )

phi = And( h( x ) > 5, h( y ) < 2 )

solve( phi )
```

Exercise:

Exercise 4.6

Give a satisfying model of the following formula

$$g(x, y) < 0 \land g(y, x) > 0 \land y = x$$

Uninterpreted sorts

```
u = DeclareSort('U') # declaring new sort

c =  Const('c', u )  # declaring a constant of the sort

f =  Function('f', u, u ) # declaring a function of the sort

# declaring a predicate of the sort
P =  Function('P', u, BoolSort() )

phi = And( f(c) == c,  P( f(c) ),  Not( P(c) ) )

solve( phi )
```

Exercise 4.7

Get model after dropping the third atom. Interpret the model.

Commentary: Hint: the solver also chooses domains for the uninterpreted sorts, and the models of the functions are presented in terms of the domains.

Quantifiers

```
u = DeclareSort('U')
H = Function('Human', u, BoolSort() )
M = Function('Mortal', u, BoolSort() )

# Humans are mortals
x = Const('x', u )
all_mort = ForAll( x, Implies( H(x), M(x) ) )

s = Const('Socrates', u )
thm = Implies( And( H(s), all_mort ), M(s) )

solve( Not(thm) )  # negation of a valid theorem
                   # is unsatisfiable
```

Exercise: solving quantified formulas

Exercise 4.8

Prove/disprove if the following statement is valid.

There is someone such that if the one drinks, then everyone drinks

Exercise 4.9

Write a formula that only accepts infinite models. Encode the formula in Z3 and get model.

Formula handling

```
a = Bool('a')
b = Bool('b')
ab = And( a, b )

# accessing sub-formulas
print(ab.arg(0))
print(ab.arg(1))

# accessing the symbol at the head
ab_decl = ab.decl()
name = ab_decl.name()
if name == "and":
    print("Found an And")
```

Quantified formula handling

```
u = DeclareSort('U')
H = Function('Human', u, BoolSort() )
M = Function('Mortal', u, BoolSort() )
x = Const( 'x', u )
y = Const( 'y', u )

all_mort = ForAll( x, Implies( H(x), M(x) ) )
print(all_mort.body())
# Output: Implies(Human(Var(0)), Mortal(Var(0)))
# Var(0) is FOL variable
# Naming quantified variables using DeBruijn index

alt = ForAll( x, Exists( y, Implies( H(x), M(y) ) ))
print(alt.body().body())
# Output: Implies(Human(Var(1)), Mortal(Var(0)))
```

Topic 4.3

SMT2 format

API vs Input language

- Each solver has their own API
- We need a common input format for
  - interoperability and
  - database of problems

Standard format for SMT solvers

SMT2 is a standard input format for SMT solvers.

`http://smtlib.cs.uiowa.edu/language.shtml`

- Formulas are written in prefix notation (why?)

```
(>= (* 2 x) (+ y z))
```

- There is a simple type system. Similar to Z3 API.
- Solver interacts like a stack

File format

An SMT2 file has five parts

1. Preamble declarations
2. Sort declarations
3. Variable declarations
4. Asserting formulas
5. Solving commands

Preamble declaration

- Set configurations of the solvers

```
(set-logic QF_UFLIA)                ;setting Theory/Logic

(set-option :produce-proofs true)   ;enable proof generation if input is unsat
```

Sort declarations

- Declare new sorts of the variables

```
(declare-sort symbol numeral)
```

Example 4.1

```
(declare-sort U 0)     ; new sort with no parameters
(declare-sort Arr 2)   ; new sort with two parameters
```

Variable declarations

- Declare variables and functions that may be used in the formulas

```
(declare-fun symbol (sort*) sort)
```

Example 4.2

```
(declare-fun x () Int)                       ;declare variable
(declare-fun f (Int) Int)                    ;declare a function with one argument
(declare-fun g (Int Int) Int)                ;declare a function with two arguments
(declare-fun h ((Arr U Int) Int) Int);declare a function with two argument
```

Sorts with parameters

Asserting formulas

- Formulas are asserted in a sequence

Example 4.3

```
(assert (>= (* 2 x) (+ y z)))
(assert (< (f x) (g x x)))
(assert (> (f y) (g x x)))
```

Commands

- Commands are the actions that solver needs to do

Example 4.4

```
(check-sat) ; checks if the conjunction of asserted formula is sat
(get-model) ; returns a model if the formulas are sat
```

Stack interaction

The standard is designed to be interactive

- Asserted formulas are pushed in the stack of the solver
- `(push)` command places marker on the stack
- `(pop)` removes the formulas upto the last marker

Example 4.5

```
(push)
(assert (= x y))
(check-sat)
(pop)
```

After the pop the solver state goes back to the last push. Useful in interactive use of solver.

Full example

```
(set-logic QF_UFLIA)
(set-option :produce-proofs true)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(declare-fun f (Int) Int)
(declare-fun g (Int Int) Int)
(assert (>= (* 2 x) (+ y z)))
(assert (< (f x) (g x x)))
(assert (> (f y) (g x x)))
(check-sat)
(get-model)
(push)
(assert (= x y))
(check-sat)
(pop)
(exit)
```

Demo

`http://rise4fun.com/z3`

Topic 4.4

Problems

Exercise : Python programming

Exercise 4.10

Write a Python program that generates a random graph in a file `edges.txt` for n nodes and m edges, which are given as command line options.
Please store edges in `edges.txt` as the following sequence of tuples

```
10,12
30,50
....
```

Exercise 4.11

Write a program that reads a directed graph from `edges.txt` and finds the number of strongly connected components in the graph

Exercise 4.12

Write a program that reads a directed graph from `edges.txt` and finds the cliques of size k, which is given as a command line option.

Integer vs. Reals

Exercise 4.13

Consider the following constraints

$$3x - y \geq 2 \wedge 3y - z \geq 3 \wedge 3 \geq x + y$$

Solve the above constraints using SMT solver under the following theories

- *Reals (QF_LRA)*
- *Int (QF_LIA)*

Proving theorems

Exercise 4.14

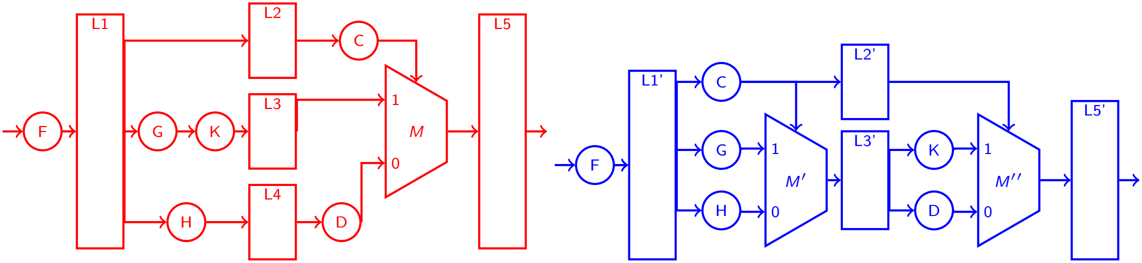
Prove/disprove the following theorems using a solver

- *Sky is blue. Space is black. Therefore sky and space are blue or black.*
- *Hammer and chainsaw are professional tools. Professional tools and vehicles are rugged. Therefore, hammers are rugged.*

Exercise: translation validation

Exercise 4.15

Show that the following two circuits are equivalent.



Ls are latches, circles are Boolean circuts, and Ms are multiplexers.

Source: http://www.decision-procedures.org/slides/uf.pdf

Write a function: compute linear coefficient

Exercise 4.16

Find coefficient of each variable in a linear term. If the term is non-linear, throw an exception.

Examples:

$x - 2x + y + 4$ *should return* $[4, -1, 1]$ *if variables are ordered* $[x, y]$.

$x - x + 4y - 2(2y)$ *should return* $[0, 0, 0]$ *if variables are ordered* $[x, y]$.

$(x + 1) * y$ *should throw an exception*

Write a function: find positive variables

Exercise 4.17

Find the set of Boolean variables that occur only positively in a propositional logic formula.

An occurrence of a variable is positive if there are even number of negations from the occurrence to the root of the formula.

Examples:

Only q *occurs positively in* $p \wedge \neg(\neg q \wedge p)$.

p *occurs positively in* $\neg\neg p$.

p *does not occur positively in* $\neg p$.

p *and* q *occur positively in* $(p \vee \neg r) \wedge (r \vee q)$.

Write a function: find quantifier alternation depth

Exercise 4.18

Compute quantifier alternations depth of a sentence.

Maximum number of quantifier type switches is any path from an atom to the root.

Examples: quantifier alternations depth of

$\forall x.\ \exists y.\ E(x,y)$ *is 1.*

$\forall x.\ \exists y.\ \forall z.\ E(x,y,z)$ *is 2.*

$\forall x.\ \forall y.\ E(x,y)$ *is 0.*

$\forall x.\ (\ (\ \exists y.\ H(x,y)\ ) \Rightarrow G(x))$ *is 0.* *($\exists$ under negation is $\forall$ and vice-versa)*

$\forall x.\ (\ (\ \exists y.\ H(x,y)\ ) \Rightarrow \exists z.\ G(x,z))$ *is 1.*

Write a function: find unrelated constraints

Exercise 4.19

Consider a formula F consists of only a conjunction of atoms. Find the subsets of F that have disjoint set of uninterpreted symbols.

Examples:

$x = y \wedge x = z \wedge P(u)$ *has two unrelated subsets* $\{x = y, x = z\}$ *and* $\{P(u)\}$

$x + y = 3 \wedge z + u \geq 10$ *has two unrelated subsets* $\{x + y = 3\}$ *and* $\{z + u \geq 10\}$, *while they have a common interpreted symbol* $+$.

Write a function: find maximum occurring symbol

Exercise 4.20

Consider a formula F. Find the uninterpreted symbol in F that occurs most often.

Examples:

x *occurs most often in* $g(g(x,x), g(x,x))$.

f *occurs most often in* $f(x,y) = f(x,b) \wedge f(2,3) > 10$.

D *occurs most often in* $\exists x.(D(x) \Rightarrow D(x+1))$. *quantified variables are not counted.*

Write a function: find common symbols

Exercise 4.21

Consider formulas F_1 and F_2. Find uninterpreted symbols the occur both in F_1 and F_2.

Examples:

$\{x, f\}$ *occurs* $f(x) > 3$ *and* $f(y) < x$, *but not* y

$\{f\}$ *occurs* $f(x) > 3$ *and* $\forall x. f(x) > y$, *but not* x *and* y.

$\{f, x\}$ *occurs* $f(x) > 3$ *and* $x > 20 \vee \forall x. f(x) > y$. *quantified variables are not counted.*

End of Lecture 4