Z3 – THEOREM PROVER

BY LUKAS PESCHEL AND SOPHIA RASKOPF

CONTENT

- Introduction
- Recap of Logic Theory
- Overall system architecture
- Theories
- Solvers
- Advantages vs. Disadvantages
- Assignment
- Quiz

INTRODUCTION - WHAT IS Z3?

- Developed by Microsoft Research to solve problems during software verification
- Z3 is an efficient *SMT* Solver ➡ stands for Satisfiability Modulo Theories
- SMT
  - is the problem of *determining whether a mathematical formula is satisfiable*
    - E.g. $x + 1 = y$ is satisfiable when $x = 1$ and $y = 2$
  - includes theory of *integers, reals, arrays, data types, bit vectors and pointers*
  - can be viewed as a language of *first order logic*

INTRODUCTION - USAGE OF Z3

What can it be used for?

SMT Solvers determine whether a formula in the language of quantifier-free first-order logic is satisfiable or not with respect to background theories

- For example: the theory of linear arithmetic using integers or reals

How can we interact with Z3?

- Over SMTLIB2 scripts as text file
- Pipe to Z3 using API calls from a high-level programming language (e.g. Python)

PROPOSITIONS

- Proposition = a statement which is true or false
- Operators:

Symbol	English
$\wedge$	AND
$\vee$	OR
$\neg$	NOT

TRUTH TABLES

- Ask: According to the statements in the table, are p *and* q true?

Are p **AND** q true?

p	q	p ∧ q
T	T	T
T	F	F
F	T	F
F	F	F

Is p **OR** q true?

p	q	p V q
T	T	T
T	F	T
F	T	T
F	F	F

Is P **FALSE**?

p	¬ p
T	F
F	T

COMPOUND PROPOSITION

- Made of several propositions
- To solve: Start with smallest part, end with actual proposition

p	q	¬ q	p ∧ ¬ q	¬ (p ∧ ¬ q)
T	T	F	F	T
T	F	T	T	F
F	T	F	F	T
F	F	T	F	T

FIRST-ORDER LOGIC

- Uses quantifiers
- Existential quantifier:

$$(\exists\, x \in A)p(x)$$

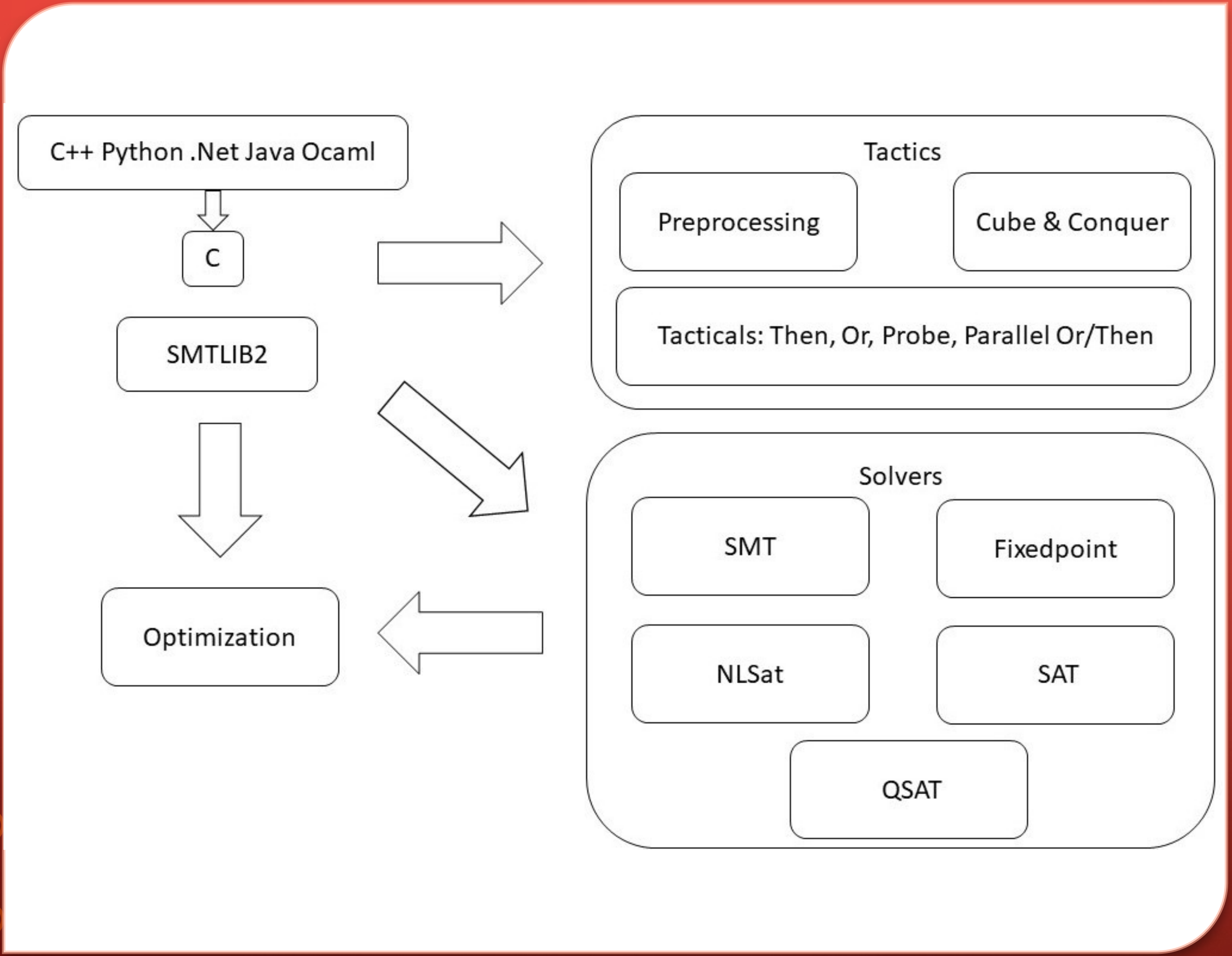
- Reads: “There exists an x in A such that p(x) is true”
  - “There exists” = quantifier, x = variable

$$(\exists\, x \in \{2, 3, 4, 5\})\ \ p(x)$$
$$p(2) = true$$

  - 2 exists in {2, 3, 4, 5} such that p(x) is true

SORTS

- Types are called sorts in Z3, e.g.:
  - Boolean Sort
  - Integer and Real Sort
  - Bit Vector Sort
  - DataType Sort
  - Relation Sort
  - Sequence Sort
- Every formula or term has a sort



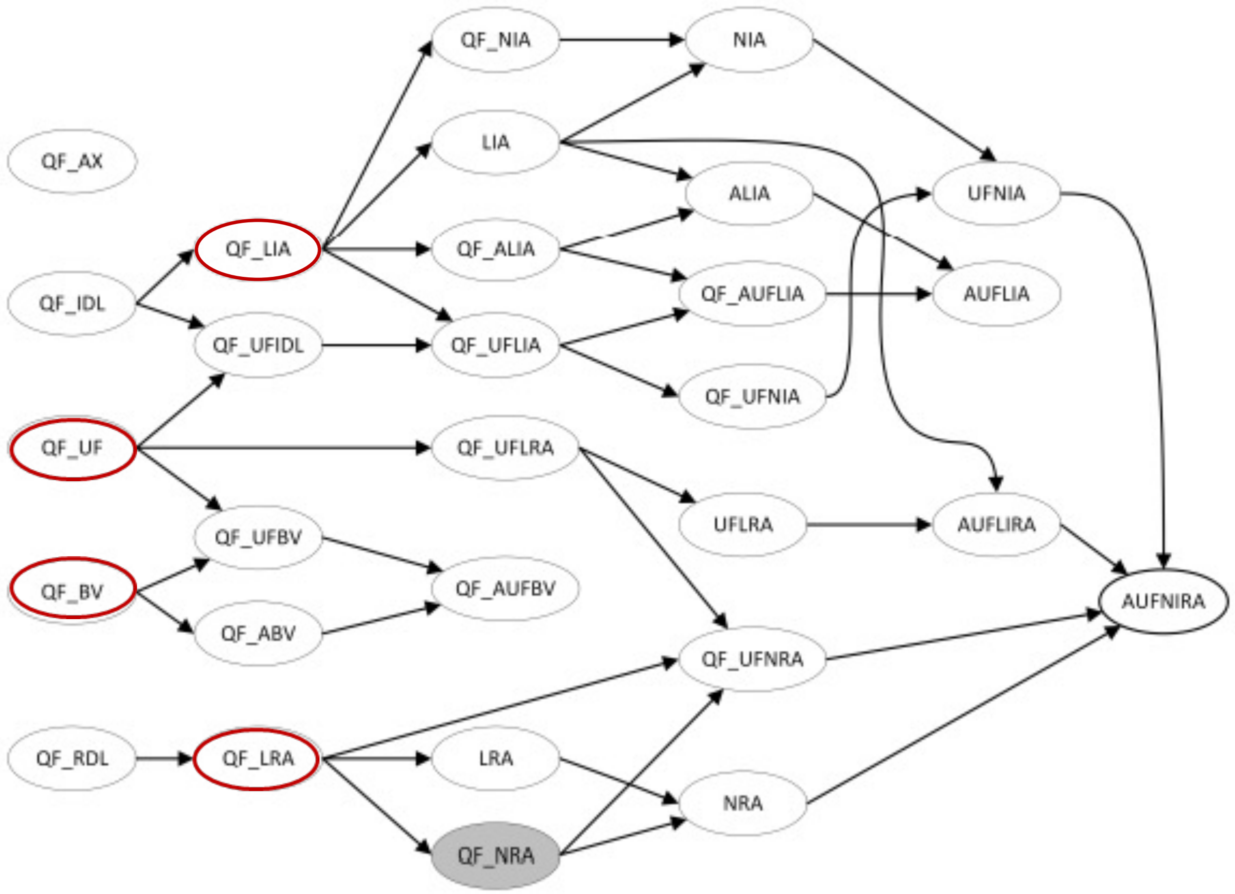
OVERALL SYSTEM ARCHITECTURE

- Accessible via an API which is available in C++, Python, .Net, Java and Ocaml
- Tactics: transform assertions to sets of assertions
- Solvers: used to check satisfiability of assertions

THEORIES - BASIC THEORY DEFINITION

- Theory is a model that is based on a set of axioms
- Formula is satisfiable under a theory, if there exists a model M that satisfies the formula under the theory T
- If there is a procedure p that checks whether any quantifier-free formula is satisfiable or not, then the satisfiability problem for a theory T is decidable
  - Meaning: p is a decision procedure for T

THEORIES - SMTLIB



THEORIES – EQUALITY AND UNINTERPRETED FUNCTIONS

- Theory of uninterpreted functions with equality denoted by SMTLIB as QF_UF
- Uninterpreted functions are functions with name and arity but no interpretation
- Allows boolean connectives (Λ, V etc.) and equalities, as well as unequalities

THEORIES – EQUALITY AND UNINTERPRETED FUNCTIONS

$a = b, b = c, d = e, b = s, d = t :$ $\quad a, b, c, s \quad d, e, t$

$a = b, b = c, d = e, b = s, d = t, a \neq d :$ $\quad a, b, c, s \quad d, e, t$

- EUF formulas based on *union-find*
- Sequence of equality assertions produce one equivalence class

- Possible: check for disequalties by checking if the equivalence classes associated with two disequal items are the same

THEORIES – EQUALITY AND UNINTERPRETED FUNCTIONS

- *Union-find* alone can be insufficient when functions are introduced
- This is where the *congruence closure* comes into play
- Congruence closure algorithms maintain a congruence relation given by a sequence of pairs of terms (i.e., equations) without variables
    - Example: Equation a=b belongs to the congruence generated by:
        - b=d, f(b)=d, and f(d)=a

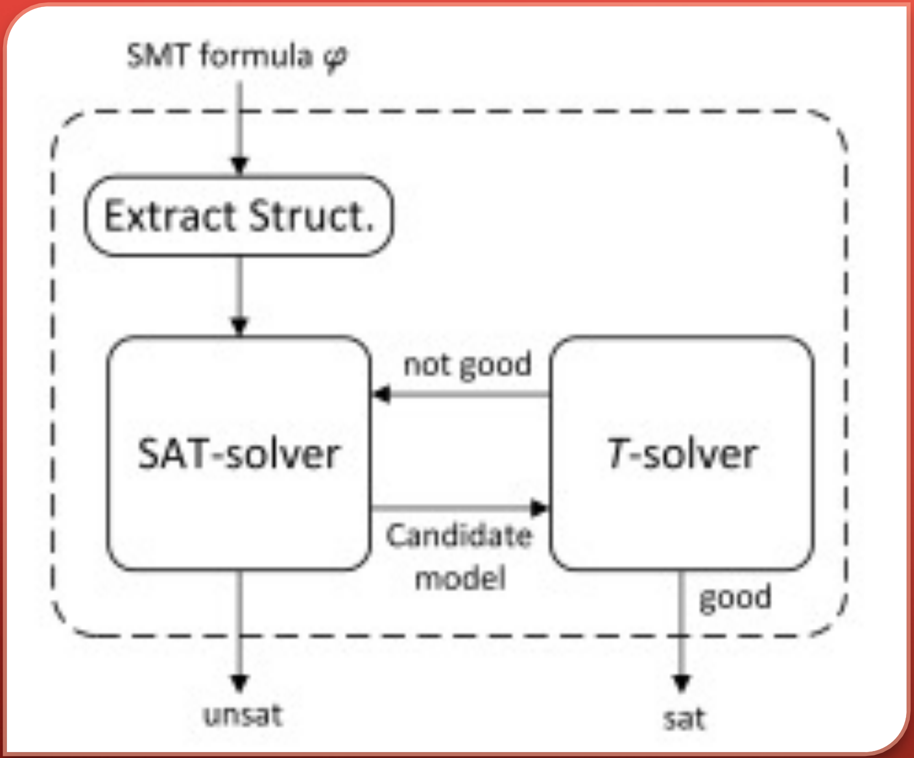
THEORIES – LINEAR ARITHMETIC

- Theory of linear arithmetic denoted by SMTLIB as LIA, LRA, QF_LIA and QF_LRA
- Definitions state that arithmetical functions +, - and * are supported
- However, * is restricted to be form of c*x where c is a constant and x is a variable

THEORIES – BIT-VECTORS

- Theory of bit-vectors is denoted by the SMTLIB as QF_BV
- Represents every number as a fixed-size sequence of bits
- In addition to standard arithmetic functions, it allows mixing bit-wise operations like NOT, AND, OR, XOR, as well as bit shifts
- *Bit-blasting approach* to solve: reduction of bit-vector constraints to propositional logic by treating each bit in a bit-vector as propositional variable

SOLVERS - SOLVING APPROACH



*SAT-Solver = Boolean Satisfiability

- Formula translated into propositional formula
- Passed to SAT-Solver* to decide satisfiability
- SAT-Solver and T-Solver interact with each other to decide T-consistency of candidate models

SOLVERS - TYPICAL STRUCTURE OF Z3 SCRIPT

(Tie ∨ Shirt) ∧ (¬ Tie ∨ Shirt) ∧ (¬ Tie ∨ ¬ Shirt)

(Tie *or* Shirt) *and* (*not* Tie *or* Shirt) *and* (*not* Tie *or* *not* Shirt)

```
from z3 import *
Tie, Shirt = Bools('Tie Shirt')
s = Solver()
s.add(Or(Tie, Shirt),
        Or(Not(Tie), Shirt),
        Or(Not(Tie), Not(Shirt)))
print(s.check())
print(s.model())
```

1. Import Z3 library
2. Declare variables
3. Create solver object
4. Add constraints to the solver
5. Check it (satisfiability). Should return sat
6. Obtain model

PYTHON VS SMTLIB2 SYNTAX: PROPOSITIONAL LOGIC

Python

```
from z3 import *
Tie, Shirt = Bools('Tie Shirt')
s = Solver()
s.add(Or(Tie, Shirt),
      Or(Not(Tie), Shirt),
      Or(Not(Tie), Not(Shirt)))
print(s.check())
print(s.model())
```

SMTLIB2

```
(declare-fun Tie() Bool)
(declare-fun Shirt() Bool)
(assert (or (Tie Shirt)
        (or (not Tie)(Shirt))
        (or (not Tie) (not Shirt)))
(check-sat)
(get-model)
```

```
sat
[Tie = False, Shirt = True]
```

PYTHON VS SMTLIB2 SYNTAX: SOLVING EQUATIONS

Python

```python
from z3 import *
x, y = Int('x'), Int('y')
s = Solver()
s.add(x + y == 42)
s.add(x < 6 * y)
s.add(x % 2 == 1)
print(s.check())
print(s.model())
```

SMTLIB2

```
(declare-const x Int)
(declare-const y Int)
(assert (= (+ x y) 42))
(assert (< (* 6 y)))
(check-sat)
(get-model)
```

```
sat
[x = 35, y = 7]
```

CUSTOM DATATYPES

```
from z3 import *
C = Datatype('Colour')
for c in ["red", "green", "blue"]:
    C.declare(c)
CSort = C.create()
s = Solver()
x = Const("x", CSort)
s.add(x != CSort.green)
s.add(x != CSort.red)
if s.check() != sat: exit(1)
print(s.model())
```

```
[x = blue]
```

- To declare your own datatype:
  - Write Datatype and provide a name for it
  - Declare constructor
  - Create it
  - X = colour variable

ADVANTAGES VS. DISADVANTAGES

- Handles a lot of different theories efficiently
- Can return models for satisfiable formulas
- Works in combination with different programming languages

- Integers and rationals are represented without roundings
  - operations by decision procedures may produce large numerals taking most execution time
- Not a lot documentation can be found besides the official documentation

WHATS NEXT?

- Demo
- 10-minute break
- Quiz: https://quizizz.com/join
  - Enter join code (from Teams chat)
- Assignment instructions can be found on https://sebivenlo.github.io/ESDE-2021-z3-workshop/
  - GitHub repo: https://github.com/sebivenlo/ESDE-2021-z3-workshop
  - Open tasks.py in a text editor and solve the tasks from the comments

THANK YOU FOR YOUR ATTENTION & PARTICIPATION!

SOURCES

- https://theory.stanford.edu/~nikolaj/programmingz3.html
- https://github.com/Z3Prover/z3
- https://sat-smt.codes/SAT_SMT_by_example.pdf
- https://spreadsheets.ist.tugraz.at/wp-content/uploads/sites/3/2015/06/DS_Hoefler.pdf
- https://www.cs.upc.edu/~oliveras/rta05.pdf
- https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.225.8231&rep=rep1&type=pdf
- https://z3prover.github.io/api/html/classz3_1_1sort.html (sorts)