Z3 Tutorial

Installation of Z3 Solver

Supported Languages

- Python
- C
- C++
- Java

Python Installation

- Requires Python3 and pip

```
pip install z3-solver
```

C / C++ Installation

- Build from source

```
git clone github.com/Z3Prover/z3
cd z3
python scripts/mk_make.py
cd build
make
sudo make install
make examples
```

How Z3 works

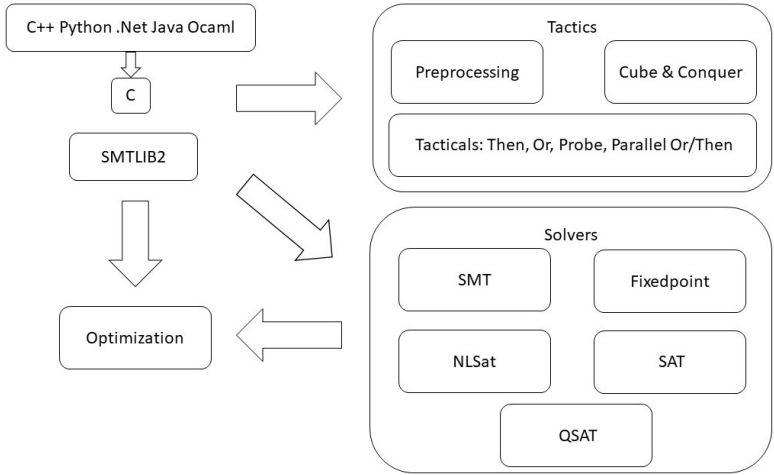

Figure: Overall system architecture of Z3.

Source: https://theory.stanford.edu/ nikolaj/programmingz3.html#sec-intro

List Comprehension

Create integer vectors:

$A = (a_0, a_1, \ldots, a_4) B = (b_0, b_1, \ldots, b_4)$

Element - wise operations:

$$A_i - B_i, A_i = B_i$$

```
from z3 import *

A = [Int(f'a{i}') for i in range(5)]
B = [Int(f'b{i}') for i in range(5)]

A_minus_B = [A[i] - B[i]
             for i in range(5)]

print(A_minus_B)

A_eq_B = [A[i] == B[i]
          for i in range(5)]

print(And(A_eq_B))
```

Output: [a0 - b0, a1 - b1, a2 - b2, a3 - b3, a4 - b4]
And(a0 == b0, a1 == b1, a2 == b2, a3 == b3, a4 == b4)

0 - 1 Knapsack Problem with Cardinality Constraint

- We are given a set of **6 items**, each with has its weight and associated profit:

$$(w, p) = \{(4, 6), (8, 12), (5, 8), (6, 7), (3, 4), (7, 11)\}$$

- The knapsack has a maximum capacity of **20 units**.
- Each item can either be selected or not selected.

Question: Is the model *satisfiable* when the *number of selected items is fixed to exactly four?*
Objective: *Maximize* the total profit subject to this constraint.

0 - 1 Knapsack Problem with Cardinality Constraint (contd.)

Sets

$$I = \{1, 2, 3, 4, 5, 6\}$$

Parameters

$w_i =$ weight of i-th item

$w_1 = 4, w_2 = 8, w_3 = 5, w_4 = 6,$

$w_5 = 3, w_6 = 7$

$p_i =$ profit from i-th item

$p_1 = 6, p_2 = 12, p_3 = 8, p_4 = 7,$

$p_5 = 4, p_6 = 11$

$C = 20$

$k = 4$

Decision Variables

$$x_i \in \{0, 1\}, \quad \forall i \in I$$

Maximize $\mathcal{Z}$

$$\mathcal{Z} = \sum_{i \in I} p_i x_i$$

Subject to

$$\sum_{i \in I} w_i x_i \leq C$$

$$\sum_{i \in I} x_i = k$$

0 - 1 Knapsack Problem with Cardinality Constraint (contd.)

```
n = 6
x = [Bool(f"x_{i}")
    for i in range(1, n+1)]

weights = [4, 8, 5, 6, 3, 7]
profits = [6, 12, 8, 7, 4, 11]
opt = Optimize()
opt.add(Sum(x) == 4)

weight_terms = []
for i in range(n):
    term = x[i] * weights[i]
    weight_terms.append(term)
total_weight_expr = Sum(weight_terms)
opt.add(total_weight_expr <= 20)

profit_terms = []
for i in range(n):
    term = x[i] * profits[i]
    profit_terms.append(term)
total_profit_expr = Sum(profit_terms)

opt.maximize(total_profit_expr)
print(opt.check())
print(opt.model())
```

0 - 1 Knapsack Problem with Cardinality Constraint (contd.)

Output:

$$
\begin{aligned}
&sat \\
&[i6 = False, \\
&i1 = True, \\
&i2 = True, \\
&i3 = True, \\
&i4 = False, \\
&i5 = True, \\
&total\_weight = 20, \\
&total\_profit = 30]
\end{aligned}
$$

Coin Selection Problem

Given a set of coins C, what is the minimum number coins required to provide sum S.
Example:

$$C = \{1, 2, 4, 7, 8, 10\}, \quad S = 15$$

Decision Variables: $x_i \in \{0, 1\}, \quad i = 1, \ldots, 6$
where

$$x_i = \begin{cases} 1 & \text{if coin } C_i \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

Constraint: $\sum_{i=1}^{6} C_i x_i = S$
Objective Function: $\min \sum_{i=1}^{6} x_i$

Coin Selection Problem using Z3

```
coins = [1,2,4,7,8,10]
n = len(coins)
x = []
for i in range(n):
    x.append(Int(f"x_{i}"))
opt = Optimize()
constraints = []

for i in range(n):
    constraints.append(
        Or(x[i] == 0,
           x[i] == 1))
total_value = 0
for i in range(n):
    total_value += coins[i]*x[i]
constraints.append(
    total_value == 15)
opt.add(constraints)
num_coins = Sum(x)
opt.minimize(num_coins)
if opt.check() == sat:
    m = opt.model()
    selected = [coins[i]
        for i in range(n)
        if m[x[i]] == 1]
    print(selected)
```

Output: Selected coins: [7, 8]
Number of coins: 2

Z3 Capabilities: Simplification (Logic)

- Simplifies logical expressions

Example:

$$\begin{aligned}(\neg a) \wedge b \wedge (a \wedge b) &= (\neg a) \wedge a \wedge b \\ &= \textbf{False} \wedge b \\ &= \textbf{False}\end{aligned}$$

```
from z3 import *

a, b = Bools('a b')
expr = And(Not(a), b, And(a, b))
simplified = simplify(expr)

print(simplified)
```

Output: `False`

Z3 Expression Simplification

- Combines terms of similar degrees
- Removes redundant constraints
- Simplifies logical formulas

Example:

1. $x + y + 2x + 1 = 3x + y + 1$
2. $x \leq y + x + 2$
   $\iff \quad y + 2 \geq 0$
3. $\text{And}(x + 1 \geq 3,$
   $x^2 + y^2 + 3x^2 - 2y^2 \geq 2)$
   $\iff \quad x \geq 2 \ \wedge \ 4x^2 - y^2 \geq 2$

```
x = Int('x')
y = Int('y')
print(simplify(x + y + 2*x + 1))
print(simplify(x <= y + x + 2))
print(simplify(And(x + 1 >= 3,
    x**2 + y**2 + 3*x**2 -2*y**2>= 2)))
```

Output:

$1 + 3 * x + y$

$-2 <= y$

$And(x >= 2, 4 * x * *2 + -1 * y * *2 >= 2)$

Z3 Capabilities: Constraint Satisfaction

- Solve equations and inequalities

Example:

Find integers $x, y \in \mathbb{Z}$

such that:

$$x + y = 10$$

$$x \geq 0,\ y \geq 0$$

$$x \bmod 2 = 0$$

$$x \in \{0, 2, 4, 6, 8, 10\} \quad \text{and} \quad y = 10 - x$$

$$(x, y) \in \{(0, 10), (2, 8), (4, 6), (6, 4), (8, 2), (10, 0)\}$$

```
x = Int('x')
y = Int('y')
s = Solver()
s.add(x + y == 10)
s.add(x >= 0, y >= 0)
s.add(x % 2 == 0)
if s.check() == sat:
    print(s.model())
```

Output: [x = 10, y = 0]

Z3 Capabilities: Incremental Solving (Push / Pop)

- `push()` : saves the current state of the solver, allowing us to add temporary constraints.
- `pop()` : restores the solver to the previous saved state, removes any constraints added after the last `push()`.

Case 1

$$x \geq 0$$
$$x > 10$$

$\Rightarrow \quad x \in \{11, 12, 13, \dots\}$

Case 2

$$x \geq 0$$
$$x < -2$$

$\Rightarrow$ No solution

Output:

SAT case 1: [x = 11]

UNSAT case 2

```
x = Int('x')
s = Solver()
s.add(x >= 0)
s.push()
s.add(x > 10)
if s.check() == sat:
    print("SAT case 1:",
        s.model())
s.pop()
s.add(x < -2)
if s.check() == sat:
    print("SAT case 2:",
        s.model())
```

Z3 Capabilities: Optimization

- Minimize or maximize objectives

$p, q \in \mathbb{Z}_{\geq 0}$
Maximize: $5p + 3q$
Subject to:

$$2p + 4q \leq 12$$
$$3p + q \leq 9$$
$$p \geq 0,\ q \geq 0$$

```
p = Int('p')
q = Int('q')
opt = Optimize()
opt.add(2*p + 4*q <= 12)
opt.add(3*p + q <= 9)
opt.add(p >= 0, q >= 0)
opt.maximize(5*p + 3*q)
if opt.check() == sat:
    print(opt.model())
```

Output: [q = 2, p = 2]

Traversing Z3 Expressions

- Z3 provides functions for traversing expressions

Expression e : $x + y \geq 4$

num_args : 2

1st child : $x + y$

2nd child : 4

children : $[x + y,\ 4]$

operator : $\geq$

```
x = Int('x')
y = Int('y')
e  = x + y >= 4
e2 = x - 2*y <= 10
print("Expression e:")
print("  num_args:", e.num_args())
print("  1st child:", e.arg(0))
print("  2nd child:", e.arg(1))
print("  children:", e.children())
print("  operator:", e.decl().name())
% print("\nExpression e2:")
% print("  num_args:", e2.num_args())
% print("  1st child:", e2.arg(0))
% print("  2nd child:", e2.arg(1))
% print("  children:", e2.children())
% print("  operator:", e2.decl().name())
```

Satisfiability Modulo Theories (SMT)

- SMT is an extension of the SAT problem.
- Instead of only checking True/False, SMT checks whether a formula is satisfiable under specific theories such as arithmetic, arrays, or bit-vectors.

Figure: A basis of SMT Solver

Classification of SMT Theories

- **Equality-Based Theories**
  - Equality with Uninterpreted Functions (EUF)
- **Arithmetic Theories**
  - Linear Arithmetic
  - Nonlinear Arithmetic
- **Fixed-Precision Numeric**
  - Bit-Vectors
  - Floating-Point
- **Data Structure Theories**
  - Arrays
  - Algebraic Datatypes
    - Lists
    - Trees, etc.
  - Strings / Sequences
- **Relation and Order**
  - Special Relations
  - Transitive Closure

SMT Theory (Bit-Vector Operations)

- Arithmetic: +, −, ∗
- Bitwise: AND (&), OR (|), XOR (^), NOT (~)

Example: 8-bit vectors x, y

- $x = 00101101_2 = 45$
- $y = 11010010_2 = 210$

```
x = BitVec('x', 8)
y = BitVec('y', 8)
s = Solver()
s.add(x + y == 100)
s.add(x & y == 32)
s.add(x ^ y == 68)
print("Satisfiability:", s.check())
```

Output:

Satisfiability:unsat

SMT Theory (Arithmetic Equalities)

Multiple solutions may exist, Z3 returns one

- Integer and Real arithmetic with coefficients
- Comparisons: $=, \neq, <, \leq, >, \geq$

$$x, y \in \mathbb{Z}$$
$$a, b \in \mathbb{R}$$
$$3x + 2y = 16$$
$$2x - y = 6$$
$$2a + 3b = 8$$
$$4a - b = 2$$

```
x, y = Ints('x y')
a, b = Reals('a b')
s = Solver()
s.add(3*x + 2*y == 16)
s.add(2*x - y == 6)
s.add(2*a + 3*b == 8)
s.add(4*a - b == 2)
print("Satisfiability:", s.check())
if s.check() == sat:
    print("Model:")
    print(s.model())
```

Output (one feasible solution)

Satisfiability: sat

Model:

[a = 1, x = 4, b = 2, y = 2]

SMT Theory (Arithmetic Inequalities)

Multiple solutions may exist, Z3 returns one

- Integer and Real arithmetic with coefficients
- Comparisons: $=, \neq, <, \leq, >, \geq$

$$x, y \in \mathbb{Z}$$
$$a, b \in \mathbb{R}$$
$$3x + 2y \leq 16$$
$$2x - y \geq 6$$
$$2a + 3b < 8$$
$$4a - b \geq 2$$

```
x, y = Ints('x y')
a, b = Reals('a b')
s = Solver()
s.add(3*x + 2*y <= 16)
s.add(2*x - y >= 6)
s.add(2*a + 3*b < 8)
s.add(4*a - b >= 2)
print("Satisfiability:", s.check())
if s.check() == sat:
    print("Model:")
    print(s.model())
```

Output (one feasible solution):

Satisfiability: sat

Model:

[b = 0, x = 0, a = 1/2, y = -6]

SMT Theory (Mixed Boolean and Polynomial Example)

- Z3 can handle Boolean and polynomial constraints simultaneously.
- Z3 finds values that satisfy both types of constraints.

$$p, q \in \{\text{True}, \text{False}\}$$
$$x, y \in \mathbb{R}$$
$$p \Rightarrow q$$
$$\neg q \vee p$$
$$x + y > 5$$
$$x^2 + y^2 < 20$$
$$x < 2 \vee y > 1$$
$$(p \wedge x > 0) \vee (\neg p \wedge y < 3)$$

```
p = Bool('p')
q = Bool('q')
x = Real('x')
y = Real('y')
solve(
    Implies(p, q),
    Or(Not(q), p),
    x + y > 5,
    x**2 + y**2 < 20,
    Or(x < 2, y > 1),
    If(p, x > 0, y < 3)
)
print(simplify(And(p, x > 1)))
print(simplify(Or(Not(q), y**2 < 10)))
```

Output:

$[y = 5/2,\ q = \text{False},\ x = 3,\ p = \text{False}]$

$\text{And}(p, \text{Not}(x \leq 1))$

$\text{Or}(\text{Not}(q), \text{Not}(10 \leq y^2))$

Fruit Purchase Puzzle

We are given **exactly Rs. 100** and we must buy **exactly 100 fruits**. Three types of fruits are available: *apples*, *bananas*, and *oranges*. Price of each apple is Rs. 10, price of each banana is Rs. 2, and price of each orange is Rs. 0.50, respectively.
We have been given the below conditions:

- At least one apple, one banana, and one orange must be purchased.
- The number of bananas must be at least twice the number of apples.
- The number of oranges must be even.

The Goal is to determine the number of apples, bananas, and oranges such that all constraints are satisfied.

Fruit Selection Optimization Model

$$I = \{0, 1, 2\}$$

$0 = \text{apple}, \quad 1 = \text{banana}, \quad 2 = \text{orange}$

Decision Variables:

$$x_i \in \mathbb{Z}, \quad i \in I$$

Constraints:

$$x_i \geq 1$$

$$\sum_{i \in I} x_i = 100$$

$$10x_0 + 2x_1 + 0.5x_2 = 100$$

$$x_1 \geq 2x_0$$

$$x_2 \equiv 0 \pmod 2$$

Fruit Purchase Puzzle (contd.)

```
items = ["apple", "banana", "orange"]
n = len(items)
x = [Int(f"x_{i}") for i in range(n)]
opt = Solver()
for i in range(n):
    opt.add(x[i] >= 1)
opt.add(Sum(x) == 100)
opt.add(10*x[0] + 2*x[1] + 0.5*x[2] == 100)
opt.add(x[1] >= 2*x[0])
opt.add(x[2] % 2 == 0)
if opt.check() == sat:
    m = opt.model()
    print(m)
    # for i in range(n):
    #     print(items[i], "=", m[x[i]])
```

Output:

$[x_2 = 72,$
$x_1 = 27,$
$x_0 = 1]$

Eight Queens Problem

Place eight queens on an 8×8 chessboard such that no two queens attack each other.

Attack Rules (Chess Queen):

- Same **row**
- Same **column**
- Same **diagonal**

A valid configuration must ensure that:

1. No two queens share the same row
2. No two queens share the same column
3. No two queens share the same diagonal

Eight Queens Problem (Contd.)

Sets

Rows: $R = \{1, 2, 3, 4, 5, 6, 7, 8\}$

Columns: $C = \{1, 2, 3, 4, 5, 6, 7, 8\}$

Decision Variables

$Q_i =$ column position of queen in row i

$Q_i \in \mathbb{Z}, \quad \forall i \in R$

Constraints

1. Domain Constraint:

$1 \leq Q_i \leq 8 \quad \forall i \in R$

2. Column Constraint:

$Q_i \neq Q_j \quad \forall i, j \in R, \ i < j$

3. Diagonal Constraints:

$Q_i - Q_j \neq i - j \quad \forall i < j$

$Q_i - Q_j \neq j - i \quad \forall i < j$

Eight Queens Problem (Contd.)

```
Q = [Int(f"Q_{i+1}") for i in range(8)]
s = Solver()

# Domain constraints
for i in range(8):
    s.add(Q[i] >= 1)
    s.add(Q[i] <= 8)

# Column constraints
for i in range(8):
    for j in range(i+1, 8):
        s.add(Q[i] != Q[j])
# Diagonal constraints
for i in range(8):
    for j in range(i+1, 8):
        s.add(Q[i] - Q[j] != i - j)
        s.add(Q[i] - Q[j] != j - i)
```

```
if s.check() == sat:
    model = s.model()
    print("Solution:")
    for i in range(8):
        print("Row", i+1,
              "-> Column",
              model[Q[i]])
```

Output:

```
Solution:
Row 1 -> Column 4
Row 2 -> Column 2
Row 3 -> Column 8
Row 4 -> Column 6
Row 5 -> Column 1
Row 6 -> Column 3
Row 7 -> Column 5
Row 8 -> Column 7
```

Assignment 4

Consider an electric vehicle charging station (CS). The CS has k number of charging ports. Each port can charge one vehicle at a given time point. $k = 1$ is the slowest charging port. Higher value of k indicates a faster charging port. A faster charging port will have a higher cost for charging. Let us assume the price of charging one time unit for different ports are $p_1, \ldots, p_k$. An EV sends a charging request to the CS with the following information - time of arrival (a_i), time of departure (d_i), and charge time (c_i) when it is connected to the slowest charging port. For k-th port charging time will be $\lceil \frac{c_i}{k} \rceil$. Suppose on a given day the CS knows all upcoming requests for the whole day. What will be the most cost effective schedule for charging of vehicles? A vehicle needs to be charged uninterruptedly on a single port only such that it receives desired amount of charge-time before its departure time.

```
% number of ports - K
K 5
% Price for ports per time unit
P 5 12 17 23 32
% vehicle requests: id arrival-time  departure-time charge-time
V 1 10 24 12
V 2 2  22  6
```

Assignment 4

- Submit 'assg04.cpp' or 'assg04.py'
- Submit 'genTestcases.cpp' or 'genTestcases.py'