Logic and Computability

Lecture 5

Introduction to Z3

Bettina Könighofer
bettina.koenighofer@iaik.tugraz.at

Stefan Pranger
stefan.pranger@iaik.tugraz.at

What is Z3?

- Solver for Satisfiability Modulo Theories

What is Z3?

- Solver for **Satisfiability** *Modulo Theories*
  - we know how to check satisfiability ✓

What is Z3?

- Solver for **Satisfiability** *Modulo Theories*
  - we know how to check satisfiability ✓
  - … until now: Only propositional logic!

What is Z3?

- Solver for **Satisfiability** *Modulo Theories*
  - we know how to check satisfiability ✓
  - … until now: Only propositional logic!

- Z3 allows us to efficiently answer decision problems including
  - Integers, Reals, Arithmetic
  - BitVectors, uninterpreted Functions, Arrays,
  - etc.

What is Z3?

- Solver for **Satisfiability** *Modulo Theories*
  - we know how to check satisfiability ✓
  - … until now: Only propositional logic!

- Z3 allows us to efficiently answer decision problems including
  - Integers, Reals, Arithmetic
  - BitVectors, uninterpreted Functions, Arrays,
  - etc.

- More on Theories starting from next week

What is Z3?

- Solver for **Satisfiability** *Modulo Theories*
  - we know how to check satisfiability ✓
  - … until now: Only propositional logic!
- Z3 allows us to efficiently answer decision problems including
  - Integers, Reals, Arithmetic
  - BitVectors, uninterpreted Functions, Arrays,
  - etc.
- More on Theories starting from next week
- Today: Basics Principles of Z3 and First Problems

Background

- Developed by Microsoft Research
  - https://github.com/Z3Prover/z3

Background

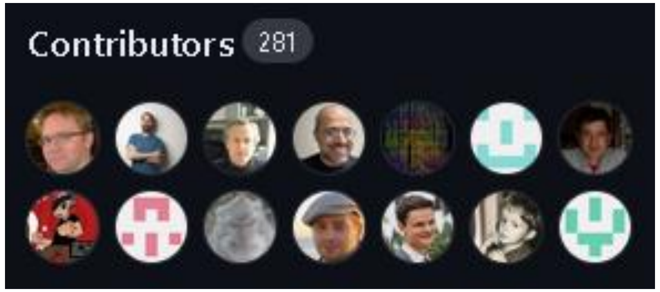
- Developed by Microsoft Research
  - https://github.com/Z3Prover/z3

Christoph Wintersteiger

Lev Nachmanson

Leonardo de Moura

Nikolaj Bjørner



Background

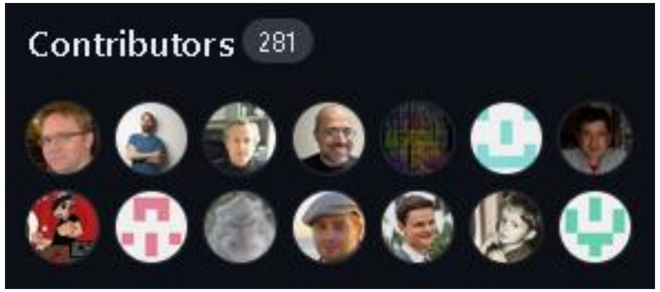
- Developed by Microsoft Research
  - https://github.com/Z3Prover/z3

Christoph Wintersteiger

Lev Nachmanson

Leonardo de Moura

Nikolaj Bjørner



- SMT-LIB2 – A standardized language for Problems in SMT

Principles

- Is $\neg a \land (a \lor b)$ satisfiable?
- What do we need to describe a problem for the solver?

Principles

- Is $\neg a \wedge (a \vee b)$ satisfiable?
- What do we need to describe a problem for the solver?
  - Variables (of a specific Sort),

```
(declare-const a Bool)
(declare-const b Bool)
```

Principles

- Is $\neg a \land (a \lor b)$ satisfiable?
- What do we need to describe a problem for the solver?
  - Variables (of a specific Sort),

```
(declare-const a Bool)
(declare-const b Bool)
```

  - Constraints, and

```
(assert (not a) )
(assert (or a b) )
```

Principles

- Is $\neg a \land (a \lor b)$ satisfiable?
- What do we need to describe a problem for the solver?
  - Variables (of a specific Sort),

    ```
    (declare-const a Bool)
    (declare-const b Bool)
    ```

  - Constraints, and

    ```
    (assert (not a) )
    (assert (or a b) )
    ```

  - Checking for Satisfiability

    ```
    (check-sat)
    ```

A Simple Example in SMT-LIB2

```
(declare-const a Bool)
(declare-const b Bool)
(assert (not a) )
(assert (or a b) )
(check-sat)
(get-model)
```

Background

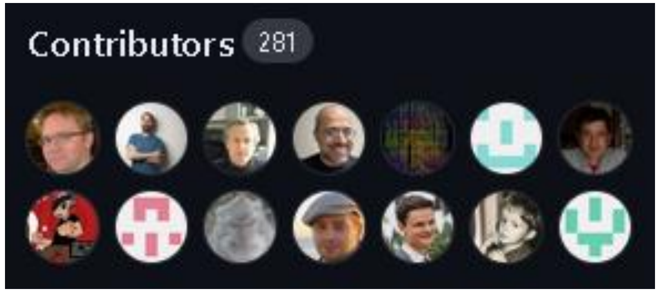
- Developed by Microsoft Research
  - https://github.com/Z3Prover/z3

Christoph Wintersteiger

Lev Nachmanson

Leonardo de Moura

Nikolaj Bjørner



- SMT-LIB2 – A standardized language for Problems in SMT
- API for C++, Python, Julia, etc.

Installing

- We will use the `Python API`:
  - `pip install z3-solver`

Installing

- We will use the `Python API`:
  - `pip install z3-solver`

- Optionally, you may install z3 natively:
  - `sudo apt-get install z3` (Via aptitude for Ubuntu, etc.)
  - https://www.nuget.org/packages/Microsoft.Z3/ (Windows)
  - https://jfmc.github.io/z3-play (online)

Python API

- User-friendly interface for `SMT-LIB2`
- Used in the Programming Assignment

- Variables (of a specific Sort),

```
(declare-const a Bool)
(declare-const b Bool)
```

- Constraints, and

```
(assert (not a) )
(assert (or a b) )
```

- Checking for Satisfiability

```
(check-sat)
```

Python API

- User-friendly interface for `SMT-LIB2`
- Used in the Programming Assignment

- Variables (of a specific Sort),

```
(declare-const a Bool)
(declare-const b Bool)
```

→

```
a = Bool("a")
b = Bool("b")
```

- Constraints, and

```
(assert (not a) )
(assert (or a b) )
```

- Checking for Satisfiability

```
(check-sat)
```

Python API

- User-friendly interface for `SMT-LIB2`
- Used in the Programming Assignment

- Variables (of a specific Sort),

```
(declare-const a Bool)
(declare-const b Bool)
```

→

```
a = Bool("a")
b = Bool("b")
```

- Constraints, and

```
(assert (not a) )
(assert (or a b) )
```

→

```
solver = Solver()
solver.add(Not(a))
solver.add(Or(a,b))
```

- Checking for Satisfiability

```
(check-sat)
```

Python API

- User-friendly interface for `SMT-LIB2`
- Used in the Programming Assignment

- Variables (of a specific Sort),

```
(declare-const a Bool)
(declare-const b Bool)
```

→

```
a = Bool("a")
b = Bool("b")
```

- Constraints, and

```
(assert (not a) )
(assert (or a b) )
```

→

```
solver = Solver()
solver.add(Not(a))
solver.add(Or(a,b))
```

- Checking for Satisfiability

```
(check-sat)
```

→

```
solver.check()
```

Python API

```
from z3 import *

a, b = Bools("a b")

solver = Solver()
solver.add(Not(b))
solver.add(Or(a,b))

print(solver.sexpr())
result = solver.check()
model = solver.model()
print(result)
print(model)
```

Python API

- Constraints

```
(assert (not a) )        →    solver.add(Not(a))
(assert (or a b) )       →    solver.add(Or(a,b))
```

- Provides Methods for Connectives:
  - **`And()`**, **`Or()`**, **`Not()`**, **`Implies()`**, ==, ^, etc.

Python API

- Constraints

```
(assert (not a) )        →    solver.add(Not(a))
(assert (or a b) )             solver.add(Or(a,b))
```

- Provides Methods for Connectives:
  - **`And()`**, **`Or()`**, **`Not()`**, **`Implies()`**, ==, ^, etc.
- Method to check whether two statements can be distinct:
  - **`Distinct(a,b)`**

Python API

- Constraints

```
(assert (not a) )
(assert (or a b) )
```

→

```
solver.add(Not(a))
solver.add(Or(a,b))
```

- Provides Methods for Connectives:
  - **`And()`**, **`Or()`**, **`Not()`**, **`Implies()`**, ==, ^, etc.
- Method to check whether two statements can be distinct:
  - **`Distinct(a,b)`**
- Operator overloading:
  - +, -, >>, <<, etc.

Python API

- Constraints

```
(assert (not a) )        →    solver.add(Not(a))
(assert (or a b) )             solver.add(Or(a,b))
```

- Provides Methods for Connectives:
  - **`And()`**, **`Or()`**, **`Not()`**, **`Implies()`**, ==, ^, etc.
- Method to check whether two statements can be distinct:
  - **`Distinct(a,b)`**
- Operator overloading:
  - +, -, >>, <<, etc.

- Reference: https://z3prover.github.io/api/html/namespacez3py.html

A First Example

- We want to show that the following formulas are equal:
  - $p \rightarrow q$
  - $\neg p \vee q$

A First Example

- $p \rightarrow q == \neg p \vee q$?

```
from z3 import *

solver = Solver()
a, b = Bools("a b")
l, r = Bools("l r")

solver.add(l == Implies(a, b))
solver.add(r == Or(Not(a), b))
solver.add(Distinct(r,l) )

result = solver.check()
print(result)
```

Back to SMT-LIB2

- $p \rightarrow q == \neg p \wedge q$?

```
from z3 import *

solver = Solver()
a, b = Bools("a b")
l, r = Bools("l r")

solver.add(l == Implies(a, b))
solver.add(r == Or(Not(a), b))
solver.add(Distinct(r,l))
print(solver.sexpr())

result = solver.check()
print(result)
```

BitVectors

- Z3 allows us to use so-called *theories*
- We have a first look at BitVectors

BitVectors

- Z3 allows us to use so-called *theories*
- We have a first look at BitVectors

- Syntax:
  - `bv = BitVector("bv", <size>)`

BitVectors

- Z3 allows us to use so-called *theories*
- We have a first look at bitvectors

- Syntax:
  - `bv = BitVector("bv", <size>)`

- `BitVectors` respect under-/overflow behaviour!
  - In contrast to Z3’s integers

Operations on BitVectors

- The BitVector Sort respects overloaded operators:

Operations on BitVectors

- The BitVector Sort respects overloaded operators:
  - <,>, <=, +, -, <<, >>, /, etc.

Operations on BitVectors

- The BitVector Sort respects overloaded operators:
  - <,>, <=, +, -, <<, >>, /, etc.
  - Caution: These are signed interpretations

Operations on BitVectors

- The BitVector Sort respects overloaded operators:
  - <,>, <=, +, -, <<, >>, /, etc.
  - Caution: These are signed interpretations
  - Use `ULT`, `UGT`, `ULE` for unsigned interpretations

Equivalence Checking for BitVectors

- We want to prove the equivalence of the following
  - `(((y & x)*-2) + (y + x))`
  - x $\oplus$ y

Weird XOR

```
from z3 import *

x = BitVec('x', 32)
y = BitVec('y', 32)

output = BitVec('output ', 32)

s = Solver()
s.add(x^y==output)
s.add(Distinct(((y & x)* -2) + (y + x),output))

print(s.check())
```

Overflow Behaviour

- We want to check whether z3 can find a model for the following
  - `x = BitVector("x", 8)`
  - `(x + 1 < x - 1)`

Operations on BitVectors

- The BitVector Sort respects overloaded operators:
  - <,>, <=, +, -, etc.
  - Caution: These are signed interpretations
  - Use `ULT`, `UGT`, `ULE` for unsigned interpretations

- Overflow and Underflow
  - `BVAddNoOverflow, BVAddNoUnderflow`
  - `BVMulNoOverflow, BVMulNoUnderflow`

Variables in a Satisfying Model

- Variables and Expressions are stored in z3-specific classes
- We can use `solver.model().decls()` to iterate through all declared variables
  - Use `.as_long()` to convert a BitVector to a Python Integer

```
model = solver.model()
for var in solver.model.decls():
    print(f"{var}: {model[var]}(:{type(model[var])})")
```

Overflow Behaviour

- We want to check whether the statement TODO
  - `(x + 1 < x - 1)`
- We need to add
  - `BVNoOverflow(x,1,True)`
  - `BVNoUnderflow(x,1,True)`
- Functions that evaluate to `False` when Over-/Underflow would occur in the model

Assignment Sheet

- 4 Exercises + 1 Bonus Exercise

Assignment Sheet

- 4 Exercises + 1 Bonus Exercise
- You are allowed to work in groups of 2
  - If you do so, please add your information into the README

Assignment Sheet

- 4 Exercises + 1 Bonus Exercise
- You are allowed to work in groups of 2
  - If you do so, please add your information into the README
- Deadline: 05. 06. 2024