CSE 403

Software Engineering

Advanced program analysis

A primer on solver-aided reasoning and verification

Z3

What is a SAT solver?

What is a SAT solver?

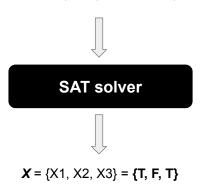
• Takes a **formula** (propositional logic) as input.

$$(X1 \ V \ X2) \ \land (\neg X1 \ V \ X3) \ \land (X1 \ V \ \neg X3) \ \land (\neg X2 \ V \ \neg X3)$$

What is a SAT solver?

- Takes a formula (propositional logic) as input.
- Returns a model (an assignment that satisfies the formula).

$$(X1 \ V \ X2) \ \land \ (\neg X1 \ V \ X3) \ \land \ (X1 \ V \ \neg X3) \ \land \ (\neg X2 \ V \ \neg X3)$$



What is Z3?

- An SMT (Satisfiability Modulo Theories) solver.
 - o Supports formulas for more complex data types
 - o Theories for Integers, Strings, Arrays, etc.

What is Z3?

- An SMT (Satisfiability Modulo Theories) solver.
 - Supports formulas for more complex data types
 - Theories for Integers, Strings, Arrays, etc.
 - o Examples for Integers:
 - a * 1 = a (identity element)
 - a + 0 = a (identity element)

What is Z3?

- An SMT (Satisfiability Modulo Theories) solver.
- Uses a standard language (SMT-LIB).
 - Print to the screen.
 - Declare variables and functions.

```
(echo "Running Z3...")
(declare-const a Int)
```

What is Z3?

- An SMT (Satisfiability Modulo Theories) solver.
- Uses a standard language (SMT-LIB).
 - o Print to the screen.
 - Declare variables and functions.
 - Define constraints.

```
(echo "Running Z3...")
(declare-const a Int)
(assert (> a 0))
```

What is Z3?

- An SMT (Satisfiability Modulo Theories) solver.
- Uses a standard language (SMT-LIB).
 - Print to the screen.
 - o Declare variables and functions.
 - Define constraints.
 - Check satisfiability and obtain a model.
 - o ..

```
(echo "Running Z3...")
(declare-const a Int)
(assert (> a 0))
(check-sat)
(get-model)
```

Which question does this code answer?

A first example

```
int simpleMath(int a, int b) {
   assert(b>0);
   if(a + b == a * b) {
     return 1;
   }
   return 0;
}
```

A first example

```
1 int simpleMath(int a, int b) {
2   assert(b>0);
3   if(a + b == a * b) {
4     return 1;
5   }
6   return 0;
7 }
```

```
(declare-const a Int)
(declare-const b Int)
(assert (> b 0))
(assert (= (+ a b) (* a b)))
(check-sat)
(get-model)
```

Does this method ever return 1? Let's ask Z3...

Does this method ever return 1?

A more complex example

```
1 int getNumber(int a, int b, int c) {
2    if (c==0) return 0;
3    if (c==4) return 0;
4    if (a + b < c) return 1;
5    if (a + b > c) return 2;
6    if (a * b == c) return 3;
7    return 4;
8 }
```

Reasoning about program equivalence

```
1 int add1(int a, int b) {
2   return a + b;
3 }
4 
5 int add2(int a, int b) {
6   return a * b;
7 }
```

Does this method ever return 3? What constraints must be satisfied?

Are these two methods semantically equivalent?

Reasoning about program equivalence

```
1 int add1(int a, int b) {
2   return a + b;
3 }
4 
5 int add2(int a, int b) {
6   return a * b;
7 }
```

```
(declare-const a Int)
(declare-const b Int)
(declare-const add1 Int)
(declare-const add2 Int)
(assert (= add1 (+ a b)))
(assert (= add2 (* a b)))
(assert (= add1 add2))
(check-sat)
(get-model)
```

Are these two methods semantically equivalent?

Reasoning about program equivalence

```
1 int add1(int a, int b) {
2   return a + b;
3 }
4 
5 int add2(int a, int b) {
6   return a * b;
7 }
```

```
(declare-const a Int)
(declare-const b Int)
(declare-const add1 Int)
(declare-const add2 Int)
(assert (= add1 (+ a b)))
(assert (= add2 (* a b)))
(assert (= add1 add2))
(check-sat)
(get-model)
```

Yes, for a=2 and b=2. What have we actually proven here?

Reasoning about program equivalence

```
1 int add1(int a, int b) {
2   return a + b;
3 }
4
5 int add2(int a, int b) {
6   return a * b;
7 }
```

```
(declare-const a Int)
(declare-const b Int)
(declare-const add1 Int)
(declare-const add2 Int)
(declare-const add2 Int)
(assert (= add1 (+ a b)))
(assert (= add2 (* a b)))
(assert (not (= add1 add2)))
(check-sat)
(get-model)
```

For **universal claims**, our goal is to **prove** the absence of counter examples (i.e., the defined constraints are **unsat**)!

Summary

- Solver-aided reasoning is used for testing and verification.
- SMT solvers:
 - o Provide one solution, if one exists.
 - Are commonly used to find counter-examples (or prove unsat).
 - Support many theories that can model program semantics.
 - o Usually support a standard language (SMT-lib).
- The challenge is to model a problem as a constraint system.
 - A few examples:
 - o Statistical test selection
 - Data-structure synthesis
 - Program synthesis
- Many higher-level DSLs and language bindings exist.