SMT Solver Z3

Testing, Quality Assurance, and Maintenance
Fall 2023

Prof. Arie Gurfinkel

UNIVERSITY OF WATERLOO

Satisfiability Modulo Theory (SMT)

Satisfiability is the problem of determining wither a formula F has a model

- if F is ***propositional***, a model is a truth assignment to Boolean variables
- if F is ***first-order formula***, a model assigns values to variables and interpretation to all the function and predicate symbols

SAT Solvers

- check satisfiability of propositional formulas

SMT Solvers

- check satisfiability of formulas in a ***decidable*** first-order theory (e.g., linear arithmetic, uninterpreted functions, array theory, bit-vectors)

(Optional) Background Reading: SMT

September 2011

…isfiability Modulo Theories: Introduction & Applications

Leonardo de Moura
Microsoft Research
One Microsoft Way
Redmond, WA 98052
leonardo@microsoft.com

Nikolaj Bjørner
Microsoft Research
One Microsoft Way
Redmond, WA 98052
nbjorner@microsoft.com

…RACT

…int satisfaction problems arise in many diverse ar- …uding software and hardware verification, type infer- …atic program analysis, test-case generation, schedul- …nning and graph problems. These areas share a …n trait, they include a core component using logical …s for describing states and transformations between …The most well-known constraint satisfaction problem …ositional *satisfiability*, SAT, where the goal is to de- …ether a formula over Boolean variables, formed using …connectives can be made *true* by choosing *true*/*false* …for its variables. Some problems are more naturally …ed using richer languages, such as arithmetic. A sup- … *theory* (of arithmetic) is then required to capture …ning of these formulas. Solvers for such formulations …monly called *Satisfiability Modulo Theories* (SMT)

SMT solvers have been the focus of increased recent attention thanks to technological advances and industrial applications. Yet, they draw on a combination of some of the most fundamental areas in computer science as well as discoveries from the past century of symbolic logic. They combine the problem of Boolean Satisfiability with domains, such as, those studied in convex optimization and term-manipulating symbolic systems. They involve the decision problem, com-…

key driving factor [4]. An important ingredient is a common interchange format for benchmarks, called SMT-LIB [33], and the classification of benchmarks into various categories depending on which theories are required. Conversely, a growing number of applications are able to generate benchmarks in the SMT-LIB format to further inspire improving SMT solvers.

There is a relatively long tradition of using SMT solvers in select and specialized contexts. One prolific case is theorem proving systems such as ACL2 [26] and PVS [32]. These use decision procedures to discharge lemmas encountered during interactive proofs. SMT solvers have also been used for a long time in the context of program verification and *extended static checking* [21], where verification is focused on assertion checking. Recent progress in SMT solvers, however, has enabled their use in a set of diverse applications, including interactive theorem provers and extended static checkers, but also in the context of scheduling, planning, test-case generation, model-based testing and program development, static program analysis, program synthesis, and run-time analysis, among several others.

We begin by introducing a motivating application and a simple instance of it that we will use as a running example.

1.1 An SMT Application - Scheduling

Consider the classical *job shop scheduling* decision prob-…

…delayed as long as needed in order to wait for a machine to become available, but tasks cannot be interrupted once

https://dl.acm.org/doi/10.1145/1995376.1995394

UNIVERSITY OF WATERLOO

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Example

$$b + 2 = c \wedge f(\mathtt{read}(\mathtt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Arithmetic

Example

$$b + 2 = c \land f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Array theory

Example

$$b + 2 = c \wedge f(\mathtt{read}(\mathtt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Uninterpreted function

UNIVERSITY OF WATERLOO

Example

$$b + 2 = c \wedge f(\mathtt{read}(\mathtt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), b)) \neq f(3)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), b)) \neq f(3)$$

then, by the array theory axiom: $\texttt{read}(\texttt{write}(v, i, x), i) = x$

$$b + 2 = c \wedge f(3) \neq f(3)$$

Example

$$b + 2 = c \wedge f(\mathtt{read}(\mathtt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\mathtt{read}(\mathtt{write}(a, b, 3), b)) \neq f(3)$$

then, by the array theory axiom: $\mathtt{read}(\mathtt{write}(v, i, x), i) = x$

$$b + 2 = c \wedge f(3) \neq f(3)$$

then, the formula is unsatisfiable

Example 2

$$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

Example 2

$$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

This formula is satisfiable

Example 2

$$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

This formula is satisfiable:

Example model:

$$x \rightarrow 1$$

$$y \rightarrow 2$$

$$f(1) \rightarrow 0$$

$$f(2) \rightarrow 1$$

$$f(\ldots) \rightarrow 0$$

SMT-LIB: http://smt-lib.org

International initiative for facilitating research and development in SMT

Provides rigorous definition of syntax and semantics for theories

SMT-LIB syntax

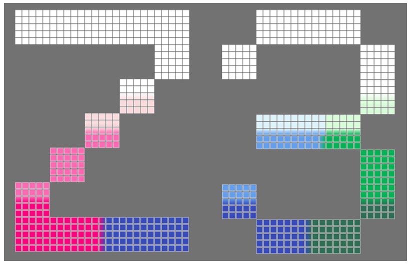
- based on s-expressions (LISP-like)
  - https://en.wikipedia.org/wiki/S-expression
- common syntax for interpreted functions of different theories
  - e.g. `(and (= x y) (<= (* 2 x) z))`
- commands to interact with the solver
  - `(declare-fun …)` declares a constant/function symbol
  - `(assert p)` conjoins formula p to the curent context
  - `(check-sat)` checks satisfiability of the current context
  - `(get-model)` prints current model (if the context is satisfiable)

- see examples at http://rise4fun.com/z3

SMT-LIB Syntax

```
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (>= (* 2 x) (+ y z)))
(declare-fun f (Int) Int)
(declare-fun g (Int Int) Int)
(assert (< (f x) (g x x)))
(assert (> (f y) (g x x)))
(check-sat)
(get-model)
```

SMT Example in Z3



Freeform Editing

Freeform Editing

Run Z3 on SMTLIB on the web!

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

(declare-fun f (Int) Int)
(declare-fun g (Int Int) Int)
(assert (< (f x) (g x x)))
(assert (> (f y) (g x x)))
(check-sat)
(get-model)

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

Run

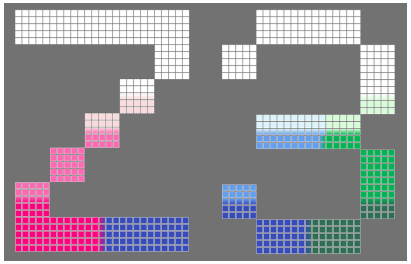
https://microsoft.github.io/z3guide/playground/Freeform%20Editing

UNIVERSITY OF WATERLOO

Example

$$b + 2 = c \wedge f(\mathtt{read}(\mathtt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

And now in Z3…



Freeform Editing

Freeform Editing

Run Z3 on SMTLIB on the web!

```
(set-option :produce-proofs true)
(declare-fun b () Int)
(declare-fun c () Int)
(declare-fun a () (Array Int Int))
(declare-fun f (Int) Int)
(assert (= (+ b 2) c))
(assert (not (= (f (select (store a b 3) (- c 2))) (f (+ (- c b) 1)))))
(check-sat)
(get-proof)
```

Run

https://microsoft.github.io/z3guide/playground/Freeform%20Editing

UNIVERSITY OF WATERLOO

```python
import z3


def main():
    b, c = z3.Ints("b c")
    a = z3.Array("a", z3.IntSort(), z3.IntSort())
    f = z3.Function("f", z3.IntSort(), z3.IntSort())
    solver = z3.Solver()
    solver.add(c == b + z3.IntVal(2))
    lhs = f(z3.Store(a, b, 3)[c - 2])
    rhs = f(c - b + 1)
    solver.add(lhs != rhs)
    res = solver.check()
    if res == z3.sat:
        print("sat")
    elif res == z3.unsat:
        print("unsat")
    else:
        print("unknown")


if __name__ == "__main__":
    main()
```

z3 python package

```python
import z3


def main():
    b, c = z3.Ints("b c")
    a = z3.Array("a", z3.IntSort(), z3.IntSort())
    f = z3.Function("f", z3.IntSort(), z3.IntSort())
    solver = z3.Solver()
    solver.add(c == b + z3.IntVal(2))
    lhs = f(z3.Store(a, b, 3)[c - 2])
    rhs = f(c - b + 1)
    solver.add(lhs != rhs)
    res = solver.check()
    if res == z3.sat:
        print("sat")
    elif res == z3.unsat:
        print("unsat")
    else:
        print("unknown")


if __name__ == "__main__":
    main()
```

create constants

SMT solver

create constraints and add to solver

run solver. can take long time.

result is: sat, unsat, unknown

Useful Z3Py Functions

All these functions are under python package `z3`

Create constants and values

- `Int(name)` – an integer constant with a given name
- `FreshInt(name)` – unique constant starting with name
- `IntVal(v), BoolVal(v)` – integer and boolean values

Arithmetic functions and predicates

- `+,-,/,<,<=,>,>=,==`, etc.
- `Distinct(a, b, …)` – the arugments are distinct (expands to many disequalities)

Propositional operators

- `And, Or, Not`

Methods of the `z3.Solver` class

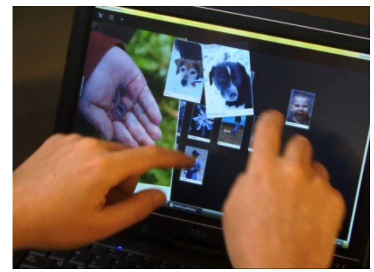
- `add(fml)` – add formula fml to the solver
- `check()` – returns z3.sat, z3.unsat, or z3.unknown (on failure to solve)
- `model()` – model if the result is sat

Methods of `z3.Model` class

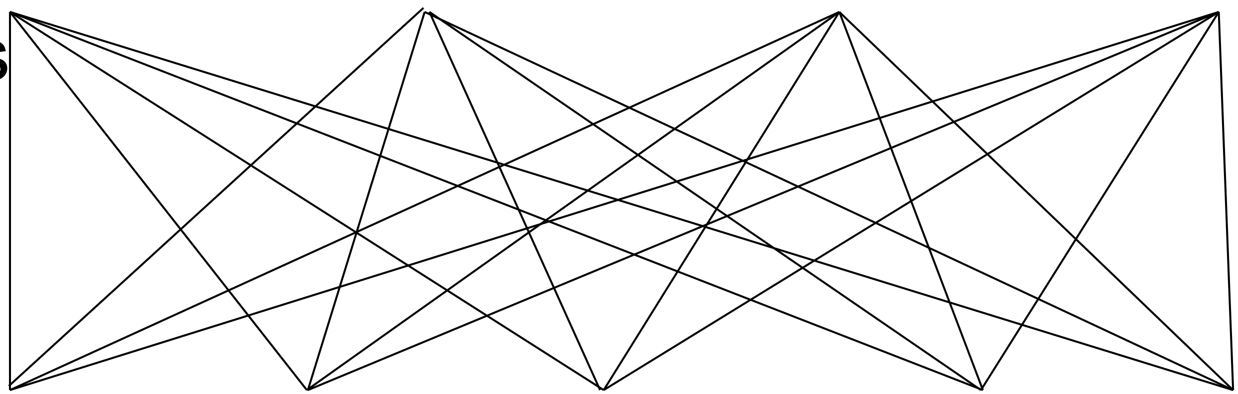
- `eval(fml)` – returns the value of fml in the model

Job Shop Scheduling





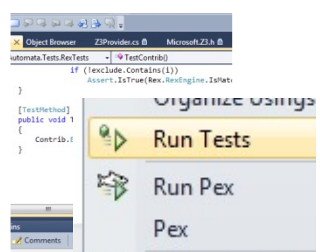

Machines



Tasks

Jobs

P = NP?

$$\zeta(s) = 0 \Rightarrow s = \frac{1}{2} + ir$$