SMT Solver Z3

Testing, Quality Assurance, and Maintenance
Winter 2020

Prof. Arie Gurfinkel

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

COMMUNICATIONS of the ACM

Protecting Users of the Cyber Commons

The Cyberwar Threat

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

...TRACT

...int satisfaction problems arise in many diverse ar-
...uding software and hardware verification, type infer-
...atic program analysis, test-case generation, schedul-
...anning 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*
...or its variables. Some problems are more naturally
...d using richer languages, such as arithmetic. A sup-
...*theory* (of arithmetic) is then required to capture
...aning of these formulas. Solvers for such formulations
...monly called *Satisfiability Modulo Theories* (SMT)

SMT solvers have been the focus of increased recent atten-
tion thanks to technological advances and industrial applica-
tions. Yet, they draw on a combination of some of the most
fundamental areas in computer science as well as discover-
ies 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-
pleteness and incompleteness of logical theories, and finally
complexity theory. In this article, we present an overview of
the field of Satisfiability Modulo Theories, and some of its
applications.

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 bench-
marks 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-
lem. In this problem, there are n jobs, each composed of
m tasks of varying duration that have to be performed con-
secutively on m machines. The start of a new task can be
delayed as long as needed in order to wait for a machine
to become available, but tasks cannot be interrupted once

September 2011

UNIVERSITY OF WATERLOO

Example

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

Example

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

Arithmetic

Example

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

Array theory

Example

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

Uninterpreted function

UNIVERSITY OF
WATERLOO

Example

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

Example

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

By arithmetic, this is equivalent to

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

UNIVERSITY OF
WATERLOO

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: $\mathtt{read}(\mathtt{write}(v, i, x), i) = x$

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

UNIVERSITY OF
WATERLOO

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: $\texttt{read}(\texttt{write}(v, i, x), i) = x$

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

then, the formula is unsatisfiable

Example 2

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

UNIVERSITY OF
WATERLOO

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:

$$x \rightarrow 1$$
$$y \rightarrow 2$$
$$f(1) \rightarrow 0$$
$$f(2) \rightarrow 1$$
$$f(\ldots) \rightarrow 0$$

UNIVERSITY OF
WATERLOO

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

z3 Microsoft Research

Is this formula satisfiable?

```
 1 ; This example illustrates basic arithmetic and
 2 ; uninterpreted functions
 3
 4 (declare-fun x () Int)
 5 (declare-fun y () Int)
 6 (declare-fun z () Int)
 7 (assert (>= (* 2 x) (+ y z)))
 8 (declare-fun f (Int) Int)
 9 (declare-fun g (Int Int) Int)
10 (assert (< (f x) (g x x)))
11 (assert (> (f y) (g x x)))
12 (check-sat)
13 (get-model)
14 (push)
15 (assert (= x y))
16 (check-sat)
17 (pop)
18 (exit)
19
```

http://rise4fun.com/z3

UNIVERSITY OF
WATERLOO

17

Example

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

Is this formula satisfiable?

```
1  ;; Is this formula satisfiable?
2  (declare-fun b () Int)
3  (declare-fun c () Int)
4  (declare-fun a () (Array Int Int))
5  (declare-fun f (Int) Int)
6  (assert (= (+ b 2) c))
7  (assert (not (= (f (select (store a b 3) (- c 2))) (f (+ (- c b) 1)))))
8  (check-sat)
```

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()
```

```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

create constants

SMT solver

create constraints and add to solver

run solver. can take long time.

result is: sat, unsat, unknown

UNIVERSITY OF
WATERLOO

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 aruguments are distinct (expands to many disequalities)
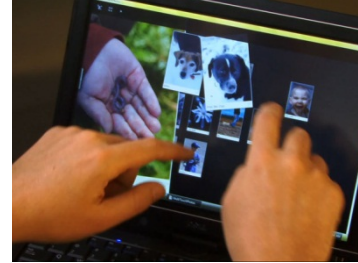
Propositional operators

- `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

UNIVERSITY OF
WATERLOO

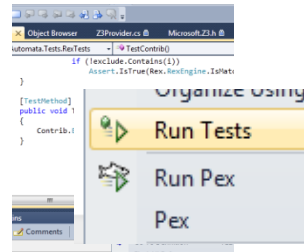# Job Shop Scheduling

Machines
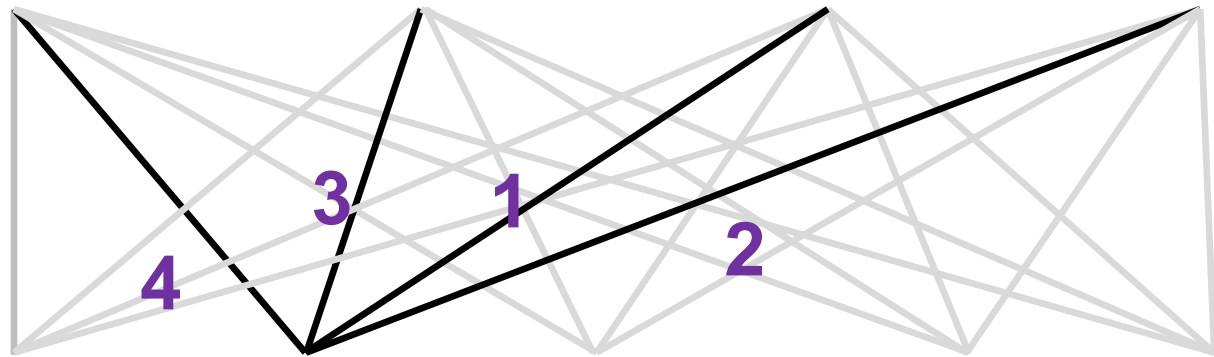
Tasks

Jobs
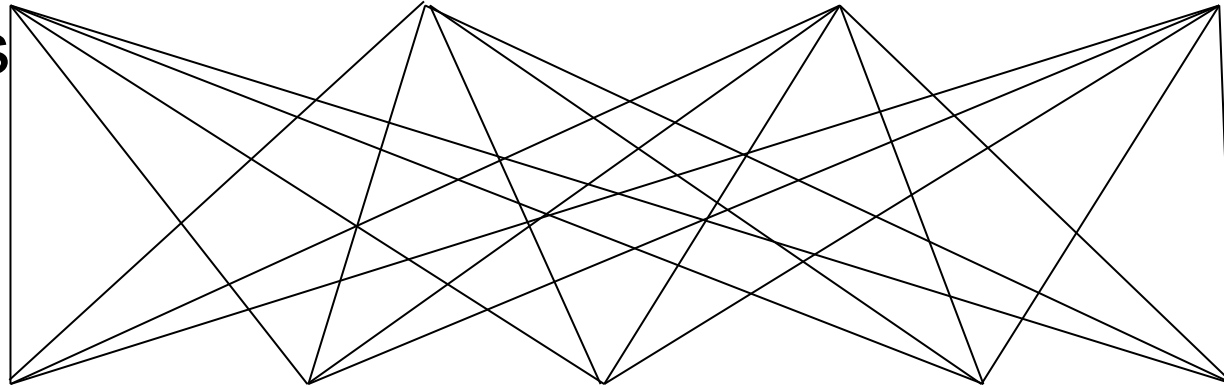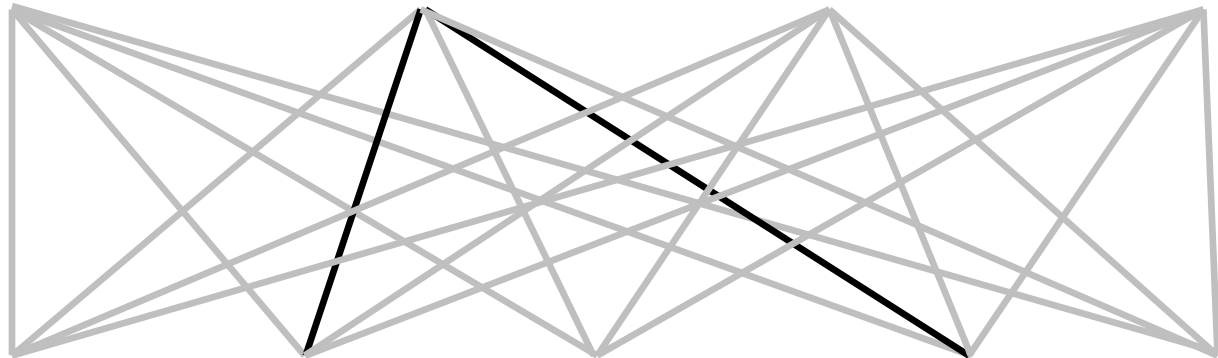
P = NP?

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

Job Shop Scheduling

Constraints:

Precedence: between two tasks of the same job

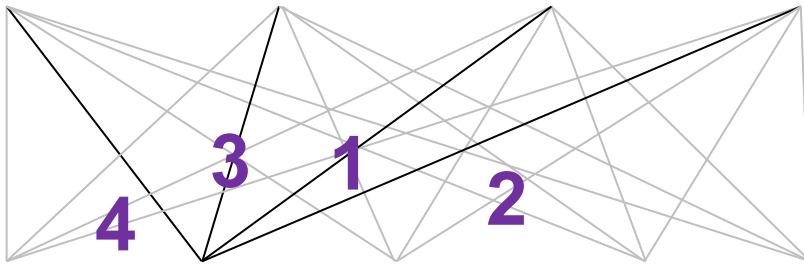
Resource: Machines execute at most one job at a time

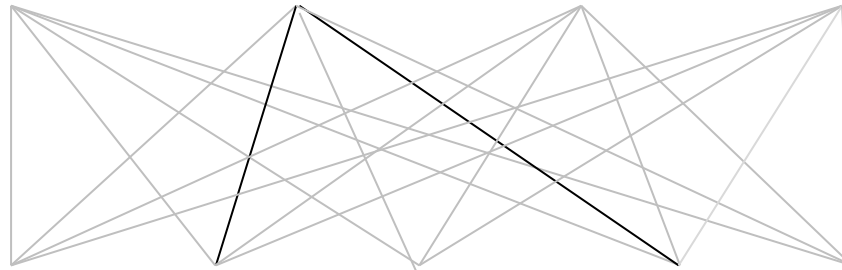$$[start_{2,2}..end_{2,2}] \cap [start_{4,2}..end_{4,2}] = \emptyset$$

Job Shop Scheduling

Constraints:

Precedence:

3 1

4   2

Resource:

$$[start_{2,2}..end_{2,2}] \cap [start_{4,2}..end_{4,2}] = \emptyset$$

Encoding:

$t_{2,3}$ - start time of job 2 on mach 3

$d_{2,3}$ - duration of job 2 on mach 3

$$t_{2,3} + d_{2,3} \leq t_{2,4}$$

Not convex

$$t_{2,2} + d_{2,2} \leq t_{4,2}$$
$$\vee$$
$$t_{4,2} + d_{4,2} \leq t_{2,2}$$

Job Shop Scheduling

$d_{i,j}$	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

$max = 8$

Solution

$t_{1,1} = 5$, $t_{1,2} = 7$, $t_{2,1} = 2$,
$t_{2,2} = 6$, $t_{3,1} = 0$, $t_{3,2} = 3$

Encoding

$(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8) \wedge$
$(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{2,1} + 3) \wedge (t_{2,2} + 1 \leq 8) \wedge$
$(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{3,1} + 2) \wedge (t_{3,2} + 3 \leq 8) \wedge$
$((t_{1,1} \geq t_{2,1} + 3) \vee (t_{2,1} \geq t_{1,1} + 2)) \wedge$
$((t_{1,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{1,1} + 2)) \wedge$
$((t_{2,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{2,1} + 3)) \wedge$
$((t_{1,2} \geq t_{2,2} + 1) \vee (t_{2,2} \geq t_{1,2} + 1)) \wedge$
$((t_{1,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{1,2} + 1)) \wedge$
$((t_{2,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{2,2} + 1))$

Dog, Cat, Mouse

Spend exactly 100 dollars and buy exactly 100 animals.

- Dogs cost 15 dollars,
- cats cost 1 dollar,
- and mice cost 25 cents each.

You have to buy at least one of each.

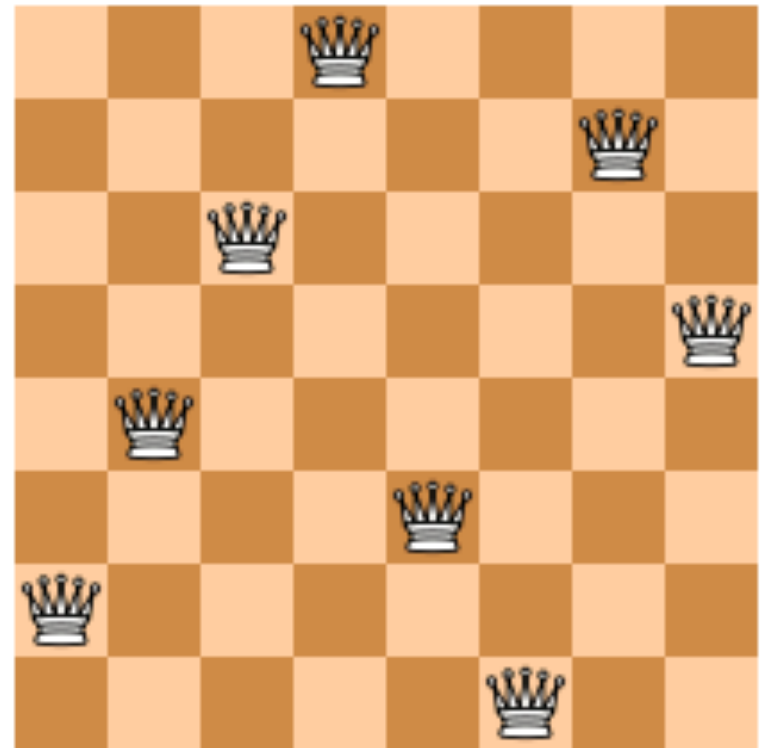How many of each should you buy?

Bit Tricks

Let `x`, `y` be a 32 bit machine integers (a bit-vector)

Show that `x!=0 && !(x & (x-1))` is true iff x is a power of 2

Show that x and y have different signs iff `x^y < 0`

Eight Queens Problem

Place 8 queens on an 8x8 chess board so that no two queen attacks one another

Incremental Interface

Z3 provides two interfaces for incremental solving that allow for adding and removing constraints

- push/pop, and assumptions

Constraints can be added at any time. This is not called incremental ☺

Push/Pop Interface

- Store current solver state by a call to push
  - `s.push ()` in Python, and `(push)` in SMT-LIB
- Restore previous state by a call to pop
  - `s.pop ()` in Python and `(pop)` in SMT-LIB

Incremental Interface: Assumptions

Requires two steps, but much more flexible than push/pop

1. tag constraints by fresh Boolean constants
    – e.g., use `(assert (=> p phi))` instead of `(assert phi)`
2. during check-sat, enable constraints by forcing tags to be true
    – e.g., use (check-sat p)

For example,

```
(assert (=> a0 c0))
(assert (=> a1 c1))
(assert (=> a2 c2))
(check-sat a0)          ; check whether c0 is sat
(check-sat a0 a2)       ; check whether c0 and c2 are sat
(check-set a1 a2)       ; check whether c1 and c3 are sat
```

Assumptions in Python Interface

Methods of `z3.Solver` class

- `check(self, *assumptions)` – check with assumptions
- `unsat_core(self)` – if the last call to `check` was unsat, returns the subset of assumptions that were actually used to show unsat