

LOGIC | SATISFIABILITY MODULO THEORIES

# SMT BASICS

WS 2020 / 2021 (342.208)



Armin Biere

[biere@jku.at](mailto:biere@jku.at)

Martina Seidl

[martina.seidl@jku.at](mailto:martina.seidl@jku.at)

**Institute for Formal Models and Verification**

Johannes Kepler Universität Linz

Version 2020.1



JOHANNES KEPLER  
UNIVERSITY LINZ



**JOHANNES KEPLER  
UNIVERSITÄT LINZ**

# Satisfiability Modulo Theories (SMT)

## Example

$$f(x) \neq f(y) \wedge x + u = 3 \wedge v + y = 3 \wedge u = a[z] \wedge v = a[w] \wedge z = w$$

- formulas in first-order logic
  - usually without quantifiers, variables implicitly existentially quantified with sorted / typed symbols
  - including functions / constants / predicates are interpreted
  - SMT quantifier reasoning weaker than in first-order theorem proving (FO)
  - much richer language compared to propositional logic (SAT)
- no need to axiomatize “theories” using axioms with quantifiers
  - important theories are “built-in”: **uninterpreted functions, equality, arithmetic, arrays, bit-vectors . . .**
  - focus is on decidable theories, thus fully automatic procedures
- state-of-the-art SMT solvers essentially rely on SAT solvers
  - SAT solver enumerates solutions to a propositional skeleton
  - propositional and theory conflicts recorded as propositional clauses
  - DPLL(T), CDCL (T), read DPLL modulo theory T or CDCL modulo T
- SMT sweet spot between SAT and FO: many (industrial) applications
  - standardized language SMTLIB used in applications and competitions

# Buggy Program

```
int middle (int x, int y, int z) {
    int m = z;
    if (y < z) {
        if (x < y)
            m = y;
        else if (x < z)
            m = y;
    } else {
        if (x > y)
            m = y;
        else if (x > z)
            m = x;
    }
    return m;
}
```

this program is supposed to return the middle (median) of three numbers

# Test Suite for Buggy Program

middle (1, 2, 3) = 2

middle (1, 3, 2) = 2

middle (2, 1, 3) = 1

middle (2, 3, 1) = 2

middle (3, 1, 2) = 2

middle (3, 2, 1) = 2

middle (1, 1, 1) = 1

middle (1, 1, 2) = 1

middle (1, 2, 1) = 1

middle (2, 1, 1) = 1

middle (1, 2, 2) = 2

middle (2, 1, 2) = 2

middle (2, 2, 1) = 2

■ This black box test suite has to be generated manually.

■ How to ensure that it covers all cases?

■ Need to check outcome of each run individually and determine correct result.

■ Difficult for large programs.

■ Better use specification and check it.

# Specification for Middle

let  $a$  be an array of size 3 indexed from 0 to 2

$$\begin{aligned} & a[i] = x \wedge a[j] = y \wedge a[k] = z \\ \wedge \\ & a[0] \leq a[1] \wedge a[1] \leq a[2] \\ \wedge \\ & i \neq j \wedge i \neq k \wedge j \neq k \\ \rightarrow \\ & m = a[1] \end{aligned}$$

median obtained by sorting and taking middle element in the order coming up with this specification is a manual process

# Encoding of Middle Program in Logic

<code>int m = z;</code>	
<code>if (y &lt; z) {</code>	$(y < z \wedge x < y \rightarrow m = y)$
<code>if (x &lt; y)</code>	$\wedge$
<code>m = y;</code>	$(y < z \wedge x \geq y \wedge x < z \rightarrow m = y)$
<code>else if (x &lt; z)</code>	$\wedge$
<code>m = y;</code>	$(y < z \wedge x \geq y \wedge x \geq z \rightarrow m = z)$
<code>} else {</code>	$\wedge$
<code>if (x &gt; y)</code>	$(y \geq z \wedge x > y \rightarrow m = y)$
<code>m = y;</code>	$\wedge$
<code>else if (x &gt; z)</code>	$(y \geq z \wedge x \leq y \wedge x > z \rightarrow m = x)$
<code>m = x;</code>	$\wedge$
<code>}</code>	$(y \geq z \wedge x \leq y \wedge x \leq z \rightarrow m = z)$
<code>return m;</code>	
<code>}</code>	

this formula can be generated automatically by a compiler

# Translating Checking of Specification as SMT Problem

let  $P$  be the encoding of the program, and  $S$  of the specification

program is correct if " $P \rightarrow S$ " is valid

program has a bug if " $P \rightarrow S$ " is invalid

program has a bug if negation of " $P \rightarrow S$ " is satisfiable (has a model)

program has a bug if " $P \wedge \neg S$ " is satisfiable (has a model)

# Checking Specification as SMT Problem Example

$$\begin{aligned} &(y < z \wedge x < y \rightarrow m = y) && \wedge \\ &(y < z \wedge x \geq y \wedge x < z \rightarrow m = y) && \wedge \\ &(y < z \wedge x \geq y \wedge x \geq z \rightarrow m = z) && \wedge \\ &(y \geq z \wedge x > y \rightarrow m = y) && \wedge \\ &(y \geq z \wedge x \leq y \wedge x > z \rightarrow m = x) && \wedge \\ &(y \geq z \wedge x \leq y \wedge x \leq z \rightarrow m = z) && \wedge \\ &a[i] = x \wedge a[j] = y \wedge a[k] = z && \wedge \\ &a[0] \leq a[1] \wedge a[1] \leq a[2] && \wedge \\ &i \neq j \wedge i \neq k \wedge j \neq k && \wedge \\ &m \neq a[1] \end{aligned}$$



# Encoding with Linear Integer Arithmetic in SMTLIB2

```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int) (declare-fun m () Int)
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y))) ; fix by replacing last 'y' by 'x'
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun a () (Array Int Int))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))
(check-sat) (get-model) (exit)
```

# Checking Middle Example with Z3

```
$ z3 middle-buggy.smt2
```

```
sat
```

```
(model
```

```
  (define-fun i () Int 1)
```

```
  (define-fun a () (Array Int Int) (_ as-array k!0))
```

```
  (define-fun j () Int 0)
```

```
  (define-fun k () Int 2)
```

```
  (define-fun m () Int 2281)
```

```
  (define-fun z () Int 2283)
```

```
  (define-fun y () Int 2281)
```

```
  (define-fun x () Int 2282)
```

```
  (define-fun k!0 ((x!1 Int)) Int
```

```
    (ite (= x!1 2) 2283
```

```
    (ite (= x!1 1) 2282
```

```
    (ite (= x!1 0) 2281 2283))))
```

```
)
```

```
$ z3 middle-fixed.smt2
```

```
unsat
```

see also <http://rise4fun.com>

# Encoding with Bit-Vector Logic in SMTLIB2

```
(set-logic QF_AUFBV)
(declare-fun x () (_ BitVec 32)) (declare-fun y () (_ BitVec 32))
(declare-fun z () (_ BitVec 32)) (declare-fun m () (_ BitVec 32))
(assert (=> (and (bvult y z) (bvult x y) ) (= m y)))
(assert (=> (and (bvult y z) (bvuge x y) (bvult x z)) (= m y))) ; fix last 'y'->'x'
(assert (=> (and (bvult y z) (bvuge x y) (bvuge x z)) (= m z)))
(assert (=> (and (bvuge y z) (bvugt x y) ) (= m y)))
(assert (=> (and (bvuge y z) (bvule x y) (bvugt x z)) (= m x)))
(assert (=> (and (bvuge y z) (bvule x y) (bvule x z)) (= m z)))
(declare-fun i ()(_ BitVec 2)) (declare-fun j ()(_ BitVec 2)) (declare-fun k ()(_ BitVec 2))
(declare-fun a ()(Array (_ BitVec 2) (_ BitVec 32)))
(assert (and (bvule #b00 i) (bvule i #b10) (bvule #b00 j) (bvule j #b10)))
(assert (and (bvule #b00 k) (bvule k #b10)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (bvule (select a #b00) (select a #b01)))
(assert (bvule (select a #b01) (select a #b10)))
(assert (distinct i j k)) (assert (distinct m (select a #b01)))
(check-sat) (get-model) (exit)
```

# Checking Middle Example with Boolector

```
$ boolector -m middle32-buggy.smt2
sat
...
2 11001101100011110101101111001001 x
3 01101101100011110101101110000001 y
4 11101011000011110101100111010001 z
5 01101101100011110101101110000001 m
28 01 i
29 00 j
30 10 k
31[00] 01101101100011110101101110000001 a
31[01] 11001101100011110101101111001001 a
31[10] 11101011000011110101100111010001 a

$ boolector middle32-fixed.smt2
unsat
```

see also <http://fmv.jku.at/boolector>

# Theory of Linear Real Arithmetic (LRA)

- constants: integers, rationals, etc.
- predicates: equality  $=$ , disequality  $\neq$ , inequality  $\leq$  (strict  $<$ ) etc.
- functions: addition  $+$ , subtraction  $-$ , multiplication  $\cdot$  by constant only

## Example

$$z \leq x - y \wedge x + 2 \cdot y \leq 5 \wedge 4 \cdot z - 2 \cdot x \geq y$$

- we focus on conjunction of inequalities as in the example first
- equalities “=” can be replaced by two inequalities “ $\leq$ ”
  - disequalities replaced by disjunction of strict inequalities
- combination with SAT allows arbitrary formulas (not just conjunctions)
- related to optimization problems solved in operation research (OR)
  - OR algorithms are usually variants of the classic SIMPLEX algorithm

# Fourier-Motzkin Elimination Procedure by Example

$$z \leq x - y \quad \wedge \quad x + 2 \cdot y \leq 5 \quad \wedge \quad 4 \cdot z - 2 \cdot x \geq y$$

pick *pivot* variable, e.g.  $x$ , and *isolate* it on one side with coefficient 1

$$\begin{aligned} z + y &\leq x & \wedge & & x &\leq 5 - 2 \cdot y & \wedge & & 4 \cdot z - y &\geq 2 \cdot x \\ z + y &\leq x & \wedge & & x &\leq 5 - 2 \cdot y & \wedge & & 2 \cdot z - 0.5 \cdot y &\geq x \\ z + y &\leq x & \wedge & & x &\leq 5 - 2 \cdot y & \wedge & & x &\leq 2 \cdot z - 0.5 \cdot y \end{aligned} \quad (1)$$

eliminate  $x$  by adding  $A \leq B$  for all inequalities  $A \leq x$  and  $x \leq B$

$$\begin{aligned} z + y &\leq 5 - 2 \cdot y & \wedge & & z + y &\leq 2 \cdot z - 0.5 \cdot y \\ z &\leq 5 - 3 \cdot y & \wedge & & 1.5 \cdot y &\leq z \end{aligned} \quad (2)$$

and same procedure with new pivot variable, e.g.  $z$ , and eliminate  $z$

$$\begin{aligned} 1.5 \cdot y &\leq 5 - 3 \cdot y \\ y &\leq 10/9 \end{aligned} \quad (3)$$

(3) has (as one) solution  $y = 0 \in (-\infty, 10/9]$  or  $y = 1 \in (-\infty, 10/9]$

(2) then allows  $z = 0 \in [0, 5]$  or  $z = 2 \in [1.5, 2]$

(1) then forces  $x = 0$  or forces  $x = 3$  thus *satisfiable*

# Theory of Uninterpreted Functions and Equality

- functions as in first-order (FO): sorted / typed without interpretation
- equality as single interpreted predicate
  - congruence axiom*  $\forall x, y: x = y \rightarrow f(x) = f(y)$
  - similar variants for functions with multiple arguments
  - always assumed in FO if equality is handled explicitly (interpreted)
- uninterpreted functions allow to abstract from concrete implementations
  - in hardware (HW) verification abstract complex circuits (e.g. multiplier)
  - in software (SW) verification abstract sub routine computation
- *congruence closure* algorithms using fast union-find data structures
  - start with all terms (and sub-terms) in different equivalence classes
  - if  $t_1 = t_2$  is an asserted literal merge equivalence classes of  $t_1$  and  $t_2$
  - for all elements of an equivalence class check congruence axiom
    - let  $t_1$  and  $t_2$  be two terms in the same equivalence class
    - if there are terms  $f(t_1)$  and  $f(t_2)$  merge their equivalence classes
  - continue until the partition of terms in equivalence classes stabilizes
  - if asserted disequality  $t_1 \neq t_2$  exists with  $t_1, t_2$  in the same equivalence class then *unsatisfiable* otherwise *satisfiable*

# Congruence Closure By Example

assume flattened structure where all sub-terms are identified by variables

$$[x \mid y \mid t \mid u \mid v]$$

$$\underbrace{x = y} \wedge x = g(y) \wedge t = g(x) \wedge u = f(x, t) \wedge v = f(y, x) \wedge u \neq v$$

asserted literal  $x = y$  puts  $x$  and  $y$  in to the same equivalence class

$$[x \ y \mid t \mid u \mid v]$$

$$x = y \wedge \underbrace{x = g(y) \wedge t = g(x)} \wedge u = f(x, t) \wedge v = f(y, x) \wedge u \neq v$$

apply congruence axiom since  $x$  and  $y$  in same equivalence class



# Congruence Closure By Example

$$[x \ y \ t \mid u \mid v]$$

$$x = y \wedge x = g(y) \wedge t = g(x) \wedge \underbrace{u = f(x, t) \wedge v = f(y, x)}_{\text{apply congruence axiom since } y, x \text{ and } t \text{ are all in same equivalence class}} \wedge u \neq v$$

apply congruence axiom since  $y, x$  and  $t$  are all in same equivalence class

$$[x \ y \ t \mid u \ v]$$

$$x = y \wedge x = g(y) \wedge t = g(x) \wedge u = f(x, t) \wedge v = f(y, x) \wedge u \neq v$$

$u$  and  $v$  in the same equivalence class but  $u \neq v$  asserted  
thus *unsatisfiable*