Satisfiability Modulo Theories Basics

Version 2015.1

Armin Biere (biere@jku.at)
Martina Seidl (martina.seidl@jku.at)
### Satisfiability Modulo Theories (SMT)

**Example**

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

- formulas in first-order logic
  - usually without quantifiers, variables implicitly existentially quantified
  - but with sorted / typed symbols and
  - 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
```c
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

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

$$a[i] = x \land a[j] = y \land a[k] = z$$
$$\land$$
$$a[0] \leq a[1] \land a[1] \leq a[2]$$
$$\land$$
$$i \neq j \land i \neq k \land j \neq k$$
$$\rightarrow$$
$$m = a[1]$$

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

(y < z ∧ x < y → m = y)
∧
(y < z ∧ x ≥ y ∧ x < z → m = y)
∧
(y < z ∧ x ≥ y ∧ x ≥ z → m = z)
∧
(y ≥ z ∧ x > y → m = y)
∧
(y ≥ z ∧ x ≤ y ∧ x > z → m = x)
∧
(y ≥ z ∧ x ≤ y ∧ x ≤ z → m = z)

this formula can be generated automatically by a compiler
Checking Specification as SMT Problem

let \( P \) be the encoding of the program, and \( S \) of the specification

program is correct if “\( P \to S \)” is valid

program has a bug if “\( P \to S \)” is invalid

program has a bug if negation of “\( P \to S \)” is satisfiable (has a model)

program has a bug if “\( P \land \neg S \)” is satisfiable (has a model)

\[
\begin{align*}
(y < z \land x < y \to m = y) \land \\
(y < z \land x \geq y \land x < z \to m = y) \land \\
(y < z \land x \geq y \land x \geq z \to m = z) \land \\
(y \geq z \land x > y \to m = y) \land \\
(y \geq z \land x \leq y \land x > z \to m = x) \land \\
(y \geq z \land x \leq y \land x \leq z \to m = z) \land \\
a[i] = x \land a[j] = y \land a[k] = z \land \\
a[0] \leq a[1] \land a[1] \leq a[2] \land \\
i \neq j \land i \neq k \land j \neq k \land \\
m \neq a[1]
\end{align*}
\]
(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
x 10111000111111001011111011111011
y 01111000111111001011111011111011
z 11110000111111011011111011111001
m 01111000111111001011111011111011
i 01
j 00
k 10
a[10] 11110000111111011011111011111001
a[01] 10111000111111001011111011111011
a[00] 01111000111111001011111011111011

$ 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

\[ z \leq x - y \land x + 2 \cdot y \leq 5 \land 4 \cdot z - 2 \cdot x \geq y \]

- we focus on conjunction of inequalities as in the example first
- equalities “=” can be replaced by two inequalities “≤”
  - 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

\[ z \leq x - y \quad \land \quad x + 2 \cdot y \leq 5 \quad \land \quad 4 \cdot z - 2 \cdot x \geq y \]

pick \textit{pivot} variable, e.g. \(x\), and \textit{isolate} it on one side with coefficient 1

\[
\begin{align*}
  z + y & \leq x \quad \land \quad x \leq 5 - 2 \cdot y \quad \land \quad 4 \cdot z - y \geq 2 \cdot x \\
  z + y & \leq x \quad \land \quad x \leq 5 - 2 \cdot y \quad \land \quad 2 \cdot z - 0.5 \cdot y \geq x \\
  z + y & \leq x \quad \land \quad x \leq 5 - 2 \cdot y \quad \land \quad x \leq 2 \cdot z - 0.5 \cdot y
\end{align*}
\]

(1)

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

\[
\begin{align*}
  z + y & \leq 5 - 2 \cdot y \quad \land \quad z + y \leq 2 \cdot z - 0.5 \cdot y \\
  z & \leq 5 - 3 \cdot y \quad \land \quad 1.5 \cdot y \leq z
\end{align*}
\]

(2)

and same procedure with new pivot variable, e.g. \(z\), and eliminate \(z\)

\[
\begin{align*}
  1.5 \cdot y & \leq 5 - 3 \cdot y \\
  y & \leq 10/9
\end{align*}
\]

(3)

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

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

(1) then forces \(x = 0\) forces \(x = 3\) thus \textit{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
Example for Uninterpreted Functions and Equality

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

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

\[x = y \land x = g(y) \land t = g(x) \land u = f(x, t) \land v = f(y, x) \land u \neq v\]

asserted literal \(x = y\) puts \(x\) and \(y\) in the same equivalence class

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

\[x = y \land x = g(y) \land t = g(x) \land u = f(x, t) \land v = f(y, x) \land u \neq v\]

apply congruence axiom since \(x\) and \(y\) in same equivalence class

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

\[x = y \land x = g(y) \land t = g(x) \land u = f(x, t) \land v = f(y, x) \land u \neq v\]

apply congruence axiom since \(y\), \(x\) and \(t\) are all in same equivalence class

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

\[x = y \land x = g(y) \land t = g(x) \land u = f(x, t) \land v = f(y, x) \land u \neq v\]

apply congruence axiom since \(y\), \(x\) and \(t\) are all in same equivalence class

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

\[x = y \land x = g(y) \land t = g(x) \land u = f(x, t) \land v = f(y, x) \land u \neq v\]

\(u\) and \(v\) in the same equivalence class but \(u \neq v\) asserted thus \textit{unsatisfiable}