Satisfiability Modulo Theories

DPLL solves Satisfiability fine on some problems but not others
   Does not do well on proving multipliers correct
       pigeon hole formulas
       cardinality constraints
   Can do well on bounded model checking
       but often it does not
   Is intended for propositional formulas

SMT combines subsolvers for certain classes of first order formulas with a DPLL SAT solver
What is a first order formula?

Quantification (exists, forall)

Predicates: for all 2-colorings of the numbers from 1 to \( n \) there exists an arithmetic progression of length \( l \) among numbers of the same color.

\[
\forall S \subset \{1...n\} (\exists S' \subset S P(S', l) \lor \exists S'' \subset \overline{S} P(S'', l))
\]
Satisfiability Modulo Theories

What is a first order theory?
Axioms, deduction rules
Theory: all axioms plus whatever can be deduced from them

Examples:

Real numbers: let $\mathcal{R}$ denote the set of real numbers

1. Operators $+$ and $\cdot$ exist, $\mathcal{R} + \mathcal{R} \rightarrow \mathcal{R}$, $\mathcal{R} \cdot \mathcal{R} \rightarrow \mathcal{R}$
2. $\{0, 1\} \subset \mathcal{R}$, $1 \cdot a \rightarrow a$, $0 + a \rightarrow a$
3. $a + b = b + a$, $a \cdot b = b \cdot a$
4. $a + (b + c) = (a + b) + c$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
5. $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
6. $\forall a \in \mathcal{R}$ $\exists -a, a^{-1} \in \mathcal{R}$ $a + (-a) = 0$, $a \cdot a^{-1} = 1$ ($a \neq 0$)
7. Operator $\geq$: $\forall x, y, z \in \mathcal{R}$ $x \geq y \rightarrow x + z \geq y + z$
   $\forall x, y \in \mathcal{R}$ $x \geq 0$ and $y \geq 0 \rightarrow x \cdot y \geq 0$
8. $LUB(S \subset \mathcal{R})$ exists if $S \neq \emptyset$, $S$ has upper bound
Satisfiability Modulo Theories

What is a first order theory?

More examples:

- Theory of linear arithmetic \((ax + by <= c)\)
- Theory of bit vectors \((\text{concat}(bv_1, bv_2) == bv_3)\)
- Theory of arrays \((\text{arr}[i := v_1][j] = v_2)\)
- Theory of uninterpreted functions \((f(f(f(x)))) == x)\)
Satisfiability Modulo Theories

What can SMT do better than SAT?

Boolean variables are replaced by predicates from various theories.

The resulting language makes it easier to express properties.

Examples: dataflow of words, as well as bits.

\( a < x < b \): simple tests involving three bit vectors.

but, as a collection of clauses:

\[(x_d \land \neg a_d) \lor (\neg x_d \land \neg a_d \land x_{d-1} \land \neg a_{d-1}) \ldots\]
Satisfiability Modulo Theories

How does it work?
The SAT solver takes care of reasoning
When needed, it consults a theory solver which decides
the validity of predicates.
Satisfiability Modulo Theories

How to use it? Example: Yices

Formula example: \( i - 1 = j + 2, f(i - 3) \neq f(j + 6) \)

Applications: equivalence checking, bounded model checking, test case generation, embedded in theorem provers

Example: yices ex1.ys

\[
\begin{align*}
&\text{(define f::(-> int int))}
&\text{(define i::int)}
&\text{(define j::int)}
&\text{(assert (= (- i 1) (+ j 2)))}
&\text{(assert (/= (f (+ i 3)) (f (+ j 6))))}
\end{align*}
\]

Obviously false
Satisfiability Modulo Theories

Example: yices ex2.ys

(define x::int)
(define y::int)
(define z::int)
(assert (= (+ (* 3 x) (* 6 y) z) 1))
(assert (= z 2))
(check)

Need the (check) to show unsatisfiable

$3 * x + 6 * y$ gives multiples of 3 $(3, 0, -3, -6, \ldots)$
Example: yices -e ex3.ys

(define x::int)
(define y::int)
(define f::(-> int int))
(assert (/= (f (+ x 2)) (f (- y 1))))
(assert (= x (- y 4)))
(check)

Counterexample: $x = 0, y = 4 \rightarrow f(2) = 1, f(3) = 5$
Satisfiability Modulo Theories

Example: yices -e ex4.ys

(define f::(-> int int))
(define i::int)
(define j::int)
(define k::int)
(assert+ (= (+ i (* 2 k)) 10))
(assert+ (= (- i 1) (+ j 2)))
(assert+ (= (f k) (f i)))
(assert+ (/= (f (+ i 3)) (f (+ j 6))))
(check)
(retract 2)
(check)

1. unsat core ids: 2 4
2. sat, $i = 4$, $k = 3$, $j = 8$, $f(3) = 15$, $f(4) = 15$, $f(7) = 16$, $f(14) = 17$

Assertions made with `assert+` can be retracted.
Lemmas discovered in first check are reused in second.
Satisfiability Modulo Theories

Example: yices -e ex5.ys

(define f::(-> int int))
(define i::int)
(define j::int)
(define k::int)
(assert+ (= (+ i (* 2 k)) 10) 10)
(assert+ (= (- i 1) (+ j 2)) 20)
(assert+ (= (f k) (f i)) 30)
(assert+ (/= (f (+ i 3)) (f (+ j 6))) 15)
(max-sat)

Returns:

sat
unsatisfied assertion ids: 4
(= i 4) (= k 3) (= j 1) (= (f 3) 8) (= (f 4) 8) (= (f 7) 9)
cost: 15
Satisfiability Modulo Theories

Example: yices -e f1.ys

(define A1::(-> int int))
(define A2::(-> int int))
(define v::int) (define w::int)
(define x::int) (define y::int)
(define g::(-> (-> int int) int))
(define f::(-> int int))
(assert (= (update A1 (x) v) A2))
(assert (= (update A1 (y) w) A2))
(assert (/= (f x) (f y)))
(assert (/= (g A1) (g A2)))
(check)

unsat

Yices does not distinguish between functions and arrays
Remove (assert (/= (f x) (f y))) to get

sat (= x 1) (= v 2) (= y 1) (= w 2) (= (A2 1) 2)
  (= (A1 1) 3) (= (g A1) 4) (= (g A2) 5)
Satisfiability Modulo Theories

Example: yices -e f2.ys

(define f::(-> int int))
(assert (or (= f (lambda (x::int) 0))
          (= f (lambda (x::int) (+ x 1))))))
(define x::int)
(assert (and (>= x 1) (<= x 2)))
(assert (>= (f x) 3))
(check)

Returns:

sat
(= x 2) (= (f 2) 3) (= (f 4) 5)
Satisfiability Modulo Theories

Example: yices -e dt.ys

(define-type list
  (datatype (cons car::int cdr::list) nil))
(define l1::list)
(define l2::list)
(assert (not (nil? l2)))
(assert (not (nil? l1)))
(assert (= (car l1) (car l2)))
(assert (= (cdr l1) (cdr l2)))
(assert (/= l1 l2))
(check)

unsat

so l1 and l2 must be the same!
Satisfiability Modulo Theories

Example: yices -e bv.ys

(define b::(bitvector 4))
(assert (= b (bv-add 0b0010 0b0011)))
(check)

sat b=0b0101
Satisfiability Modulo Theories

Example: yices -e d.ys

(define x::real)
(define y::int)
(define floor::(-> x::real
  (subtype (r::int) (and (>= x r) (< x (+ r 1))))))
(assert (and (> x 5) (< x 6)))
(assert (= y (floor x)))
(check)

sat (= x 11/2) (= y 5) (= (floor 11/2) 5)

State property of uninterpreted function
Example: yices -e q.ys

(define f::(-> int int))
(define g::(-> int int))
(define a::int)
(assert (forall (x::int) (= (f x) x)))
(assert (forall (x::int) (= (g (g x)) x)))
(assert (/= (g (f (g a))) a))
(check)

unsat

Quantifier example