CSC2512
Advanced Propositional Reasoning

CSC2512: SMT Solvers

- SMT == Sat Modulo Theory Solvers
- Basic idea is to augment a SAT solver with special purpose "theory" solvers that can more efficiently solve certain sub-formulas.
- We gain from the effectiveness of the theory solver **and** from the fact that the user can now express some of their problem in the language of one of the theory solvers.
- Typical SMT solvers implement a variety of theories.

CSC2512: Theories

- A “theory” in SMT is a **specialized** logical language that contains a **specific set** of function and predicate symbols (perhaps with equality) and might also contain a **specific** domains of values that the variables/constants/functions can take on or **specific** meanings for the predicate symbols.
- From the point of view of first-order logic, a theory is a sorted first-order language with a specific signature (set of function and predicate symbols) and potentially a specific **domain.**

CSC2512: Theories

- Given a theory we can express terms in that theory by using variables, the constants, and functions of the theory.
  - E.g., if any rational number is a constant of the theory, every variable has the REALS as a domain of possible values, and +, -, * are the binary function symbols then

    $$\mathbf{1 + \tfrac{1}{2}*x + 2*y}$$

    is an example term. (x and y are variables).
- Given the predicate symbols we can apply any k-ary predicate to k terms to obtain an **atomic** formula
  - E.g., if '=' is a binary predicate symbol whose meaning is standard equality of the REALS

    $$\mathbf{1 + \tfrac{1}{2} * x + 2 * y = 4 * z}$$

    is an example atomic formula.

CSC2512: Theories

- We can apply negation to an atomic formula to get a new formula
    - E.g., if $1 + ½ * x + 2 * y = 4 *z$ is an atomic formula, then
      $$\neg(1 + ½ * x + 2 * y = 4 *z)$$
      is a formula.
    - Normally we would rewrite this as
      $1 + ½ * x + 2 * y \neq 4 *z$
- Atomic formulas and their negations are called literals.
- We can also have conjunctions and disjunctions of literals
    - E.g., $(1 + ½ * x + 2 * y = 4 *z) \wedge z = 1$

- Note that usually we do not allow quantifiers. So the formulas of the theory are qff—quantifier free formulas

CSC2512: Theories

- Theories are designed to have restricted sets of predicates/functions and particular interpretations so that we can effectively determine whether or not a **quantifier free formula is satisfiable.**

- That is, does there exist a value for the variables such that the formula is true
  - I.e., if the formula is a conjunction each conjunct is true, if the formula is a disjunction at least one disjunct is true.
  - E.g., $(1 + \frac{1}{2} * x + 2 * y = 4 * z) \land z = 1$ is satisfiable. The assignment $x=2$, $y=1$, $z=1$ satisfies it.

CSC2512: Uninterpreted Function Symbols

- This theory does not restrict the domains of the variable, nor the number or types of the constants and function symbols. However it only contains a single predicate symbol equality '='
- So in this theory we can express things like
  $f(f(a)) = a \land g(a) \neq g(f(a)) \land f(f(f(a))) = a$
- Note we could also state things like
  $z = 2 \land z = 1 \land 2 = 1$

BUT '1' and '2' no longer have their standard interpretation! (So it is misleading to use these symbols).

CSC2512: Uninterpreted Function Symbols

- We can determine if a conjunction of literals is satisfiable (again under the interpretation that the function and constant symbols have no special meaning) by using an nlog(n) congruence closure algorithm.

CSC2512: Uninterpreted Function Symbols

- Let **E** be the set of equalities. We use the equalities in **E** to form equivalence classes.
- We continue to augment the set **E** by the rule, if s_i and t_i are in the same equivalence class (i.e., if $s_i = t_i$) for i = 1, …, k. Then we can add the equality

  $f(s_1, \ldots, s_k) = f(t_1, \ldots, t_k)$ to **E**

  for any k-ary function symbol f mentioned in **E.**
  and recompute the equivalence classes.
- We continue this until we can no longer find any new equivalence.
- **Finally we verify that for all inequalities $t_1 \neq t_2$, that t_1 and t_2 are NOT in the same equivalence class.**
  - **If they are then the conjunction is UNSAT, else it is SAT.**

CSC2512: Uninterpreted Function Symbols

- NOTE this procedure checks the SAT status of a conjunction of literals (atomic formula and their negations).
- **If we had an arbitrary quantifier free formula $\boldsymbol{\varphi}$** (conjunction/disjunctions) we could convert $\boldsymbol{\varphi}$ to DNF and then check each conjunct to see if it is SAT. $\boldsymbol{\varphi}$ is SAT iff at least one of its conjuncts is.

- **HOWEVER this could be very expensive as the DNF might contain an exponential number of terms.**

CSC2512: Real Arithmetic

- In this theory you there is a constant symbol for every rational number; the function symbols +, -, *; and the single predicate symbol ≤.
- In addition all terms (constant symbols and function applications) are interpreted as being REALS.
- Tarsky proved that even with quantifiers a set of formulas written in this language and under this interpretation of the terms is **decidable.**
- However, the procedure to decide if a conjunction of formulas in this theory is satisfiable requires doubly exponential time. So arbitrary formulas in this theory can't efficiently be reasoned with.

CSC2512: Real Arithmetic

- LRA is the restriction of this theory to linear expressions.
  - The function * can only be applied if at least one of its operands is a rational constant.
- So in LRA we can express quantifier free formulas like $2x + 1/2y + 3z \leq 4 \wedge \neg(6z + y \leq 5)$
- Note that $=, \neq, <, >, \geq$ can be encoded as negations or disjunctions of $\leq$. So we assume that we have access to these predicates as well.

- Satisfiability of this fragment can be tested with **Fourier–Motzkin elimination.**

CSC2512: Real Arithmetic

Fourier–Motzkin elimination:

1. Eliminate inequalities $t_1 \neq t_2 \rightarrow t_1 < t_2 \lor t_2 < t_1$
   Replace weak inequalities $t_1 \leq t_2 \rightarrow t_1 < t_2 \lor t_1 = t_2$
   These steps introduce disjunctions.
2. Convert to DNF, so that we have a set of conjunctions of equalities/strict inequalities. (**This step could be exponential**).
3. Process **each conjunct** looking for a single satisfiable conjunct.

CSC2512: Real Arithmetic

Process each Conjunct looking for a single satisfiable conjunct.

1. Convert each equality $t_1 = t_2$ to the form $x = t_3$ where x is a variable and t_3 is an expression not containing x.

   If this can't be done the conjunct either converts to a trivial equality t = t and can be dropped

   Or to an unsatisfiable equality, e.g., 2 = 3 and the conjunct is unsatisfiable.

2. Replace x everywhere else by t_3 and drop the equality $x = t_3$
3. This leaves us with a conjunct containing only strict inequalities. Now we can apply Fourier-Motzkin elimination on the strict inequalities.

CSC2512: Real Arithmetic

Fourier–Motzkin elimination on the strict inequalities

1. Pick a variable x to eliminate and rewrite all inequalities containing x to the form
   (1) $x < t_2$ or (2) $t_1 < x$.
2. For every pair of x inequalities one of the form (1) and one of the form (2)
   Add the inequality $t_1 < t_2$
3. Remove all inequalities containing x.
4. Repeat 4-6 for each variable.
5. The remaining inequalities contain only rational constants and we can check if they are all SAT. If any are not then this conjunct is UNSAT.

CSC2512: Real Arithmetic

This procedure can also be exponential.

Modern SMT solvers use Simplex based algorithms to solve a conjunctive set of linear equations.

As with **uninterpreted functions** we can check a conjunction effectively, but checking an arbitrary formula requires a potentially exponential conversion to **DNF**

CSC2512: Arrays

This theory contains three types of variables and constaints:

- Arrays, indices, array elements

And two function symbols **read** and **write**

read is a function of an array and an index to an element

write is a function of an array, an index, and an element to a new array

These function symbols are constrained by the axioms:

1. $\forall a{:}\mathsf{A}\,\forall i{:}\mathsf{I}\,\forall v{:}\mathsf{E}\ \mathsf{read}(\mathsf{write}(a,i,v),i) = v,$
2. $\forall a{:}\mathsf{A}\,\forall i,j{:}\mathsf{I}\,\forall v{:}\mathsf{E}\ i \neq j \Rightarrow \mathsf{read}(\mathsf{write}(a,i,v),j) = \mathsf{read}(a,j),$
3. $\forall a\,\forall b{:}\mathsf{A}\ (\forall i{:}\mathsf{I}\,\mathsf{read}(a,i) = \mathsf{read}(b,i)) \Rightarrow a = b.$

CSC2512: Arrays

- Different methods have been developed to reason about **conjunctions of formulas** involving these functions and equality.

- E.g., **read**(a,i) = x ∧ **read**(b,i) ≠ x ∧ x ≠ w ∧
  a = **write**(b,j,w) is UNSAT

  - If i = j then
    x = **read**(a,i)
    = **read**(a,j) // i = j
    = **read**(**write**(b,j,w), j) // a = **write**(b,j,w)
    = w //Axiom 1
    with x ≠ w this is UNSAT

CSC2512: Arrays

- Different methods have been developed to reason about **conjunctions of formulas** involving these functions and equality.

- E.g., **read**(a,i) = x ∧ **read**(b,i) ≠ x ∧ x ≠ w ∧
  a = **write**(b,j,w) is UNSAT

  - If i ≠ j then
    x = **read**(a,i)
    = **read**(**write**(b,j,w), i) // a = **write**(b,j,w)
    = **read**(b, i) // axiom 2
    ≠ x
    Also UNSAT

CSC2512: Arrays

- This of reasoning can be done with congruence closure as it corresponds to a sequence of equalities.

CSC2512: Other theories

- SMT solvers implement other theories as well including integer arithmetic, difference logic, languages for reasoning about bit vectors…
- In general these techniques can deal effectively with **conjunctions of literals in the theory.**

CSC2512: Integrating Theory Reasoning

- Given a collection of atomic formulas in a theory, e.g., {$2x + 3y \le 4z$, $2/3z \le y$, ...} we map each different atomic formula to a new propositional variable.
  - $2x + 3y \le 4z \rightarrow v1$
  - $2/3z \le y \rightarrow v2$
- Then given a negated atomic formula we map that to the negated propositional variable
- We map conjunctions and disjunctions of literals to the corresponding conjunctions and disjunctions of propositional literals.
- This mapping is called the Boolean Abstraction B

CSC2512: Integrating Theory Reasoning

- E.g.,
  - $2x + 3y \leq 4z \rightarrow v1$
  - $2/3z \leq y \rightarrow v2$
  - $x + z \leq 6 \rightarrow v3$

- Now the formula

  $((2x + 3y > 4z) \vee (x + z > 6)) \wedge (2x + 3y \leq 4z)$

  maps to the propositional formula

  $$(\neg v1 \vee \neg v2) \wedge v3$$

CSC2512: Integrating Theory Reasoning

- The reverse mapping (the Concretization) maps the propositional literals back to the atomic formulas or to the negation of these atomic formulas
    - $v1 \rightarrow 2x + 3y \leq 4z$
    - $\neg v2 \rightarrow 2/3\, z > y$
    - $v1 \land \neg v2 \rightarrow (2x + 3y \leq 4z) \land (2/3\, z > y)$

CSC2512: Simplest Lazy Approach

Given a formula F over theory literals (conjunctions and disjunctions) construct the corresponding Boolean Abstraction which is a propositional formula P.
Then

```
1.  P = ToCNF(P)
2.  (sat?, $\pi$) = SatSolve(P)
        // $\pi$ is a set of true literals that satisfies P true
3.  if not sat?
        return F is UNSAT
4.  else C = Concretization($\pi$)
      // C is a conjunction of theory literals
5.  tsat? = TheorySolve(C)
      // Check if this conjunction of theory literals is SAT
      // Using the Theory Solver
6.  If tsat?
        return F is SAT
7.  not-$\pi$ = $\vee \{\neg l \mid l \in \pi\}$
        //not-$\pi$ is a clause that blocks the model $\pi$
8.  P = P + not-$\pi$
        //Add this clause to P
9.  GOTO 2
        //Find another conjunct that might satisfy F.
```

CSC2512: Simplest Lazy Approach

The models of P correspond to conjuncts in the DNF of F.

So the SAT solver is computing these conjuncts and then checking if they are SAT.

If any of them are SAT, F is SAT.

We might get lucky and find a satisfying conjunct quickly, where as actually converting F to DNF might take a long time.

But if F is UNSAT this simple method will eventually have to examine all conjunctions of F's DNF.

CSC2512: More Sophisticated Approaches

Simplest extension is to ask the TheorySolver to return a reason why the conjunct it was passed is UNSAT.

```
5.  tsat? = TheorySolve(C)
       // Check if this conjunction of theory literals is SAT
       // Using the Theory Solver
```

➔

```
  (tsat?, Conflict) = TheorySolve(C)
   // Check if this conjunction of theory literals is SAT
   // Using the Theory Solver. If UNSAT return a subset
   // negated literals of C one of which must be made
   // true. I.e., a clausal reason.
```

CSC2512: More Sophisticated Approaches

```
(tsat?, Conflict) = TheorySolve(C)
 // Check if this conjunction of theory literals is SAT
 // Using the Theory Solver. If UNSAT return a subset
 // negated literals of C one of which must be made
 // true. I.e., a clausal reason.
```

E.g., TheorySolve($2x + 3y \leq 4z$, $2/3z \leq y$, $x + z > 6$) if this conjunction of theory literals is UNSAT might return

Conflict = $(\neg(2x + 3y \leq 4z) \vee \neg(x + z > 6))$
these two literals were sufficient to cause UNSAT

CSC2512: More Sophisticated Approaches

(tsat?, Conflict) = TheorySolve(C)

Now instead of adding
$\quad$ not-π = $\bigvee \{\neg l \mid l \in \pi\}$

To the SAT formula we add the Boolean abstraction of Conflict.

If the Conflict extraction process is good, i.e., it can produce short conflicts, this can cut down the number of potential theory conjuncts that have to be tested exponentially.

Note that if we can find a (theory) MUS of C this could be a good conflicts (a locally minimal one).

CSC2512: More Sophisticated Approaches

We could also call the Theory solver eagerly.

As we build up the SAT trail making literals true, we can ask if theory solver if the theory conjunct corresponding to the currently true literals is SAT.

If not we backtrack the SAT solver.

Could also the theory solver to generate new theory literals forced by the currently true theory literals—a form of Theory Propagation.

Can also ask the theory solver for an theory explanation for the literals it forces—the Boolean abstraction of these explanations forms a reason clause in the SAT solver and we can now do Theory augmented clause learning.

Which of these if effective depends on how much time the Theory Solver takes.

CSC2512: Reading for Next time

H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *Proceedings of the 16th International Conference on Computer Aided Verification, CAV'04 (Boston, Massachusetts)*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004.

Murphy Berzish, Vijay Ganesh and Yunhui Zheng. Z3str3: A string solver with theory-aware heuristics. In Proceedings of 2017 Formal Methods in Computer Aided Design, {FMCAD} 2017.