Lecture 14
SMT Solvers

Zvonimir Rakamarić
University of Utah

slides acknowledgements: Leonardo de Moura

This Time

- SMT solvers
  - What are they?
  - How they work?

Many Theories

- Theory of equality
- Peano arithmetic
- Presburger arithmetic
- Linear integer arithmetic
- Reals
- Rationals
- Arrays
- Recursive data structures
- …

Combination of Theories

- In practice, we often need a combination of theories
- Example:

  $x+2=y \rightarrow f(select(store(a,x,3),y-2)=f(y-x+1)$

- Problem: given satisfiability procedures for conjunction of literals of Theory_1 and Theory_2, how to decide satisfiability of their combination?

Satisfiability Modulo Theories (SMT) Solver

- Satisfiability checker with built-in support for useful theories
  - Arithmetic
  - Equality with uninterpreted functions
  - Arrays
  - …
- Combines a SAT solver with theory solvers
- Next generation of reasoning engines
  - Automatic
  - Fast

SMT Solvers, Library, Competition

- Solvers
  - AProve, Barcelogic, Boolector, CVC4, MathSAT5, OpenSMT, SMTInterpol, SOLONAR, STP2, veriT, Yices, Z3
- SMT-LIB
  - Standardizes various theories and input format
  - Library of benchmarks
  - http://www.smtlib.org
- SMT-COMP
  - Annual competition
  - http://www.smtcomp.org

Applications

- Test case generation
- Verifying compilers
- Software verification
- Hardware verification
- Equivalence checking
- Type checking
- Model based testing
- Scheduling and planning
- …

Nelson-Oppen Combination Procedure

- Initial State
  - F is a conjunction of literals over $\Sigma_1 \cup \Sigma_2$
- Purification
  - Preserving satisfiability transform F into $F_1 \wedge F_2$, such that $F_i \in \Sigma_i$
- Interaction
  - Deduce an equality $x = y$ if $F_1 \rightarrow x = y$, where x and y are common (shared) variables
  - Update $F_2 := F_2 \wedge x = y$
  - And vice-versa
  - Repeat until no further changes

Nelson-Oppen Combination Procedure

- Component procedures
  - Use individual decision procedures to decide whether F_i is satisfiable
- Return
  - If both return yes, return yes
  - No, otherwise
- Remark:

$F_i \rightarrow x = y$ iff $F_i \wedge x \neq y$ is not satisfiable

Purification Example

$f(x - 1) - 1 = x \wedge f(y) + 1 = y$

Nelson-Oppen Procedure Example I

$$x + y = z \land f(z) = z \land f(x + y) \neq z$$

Nelson-Oppen Procedure Example II

$x+2=y \wedge f(select(store(a,x,3), y-2)) \neq f(y-x+1)$

Building an Efficient Solver

Eager Approach

- Translate formula into equisatisfiable propositional formula and use off-the-shelf SAT solver
- Why "eager"?
  - Search uses all theory information from the beginning
- Can use best available SAT solver
- Sophisticated encodings are need for each theory
- Sometimes translation and/or solving too slow

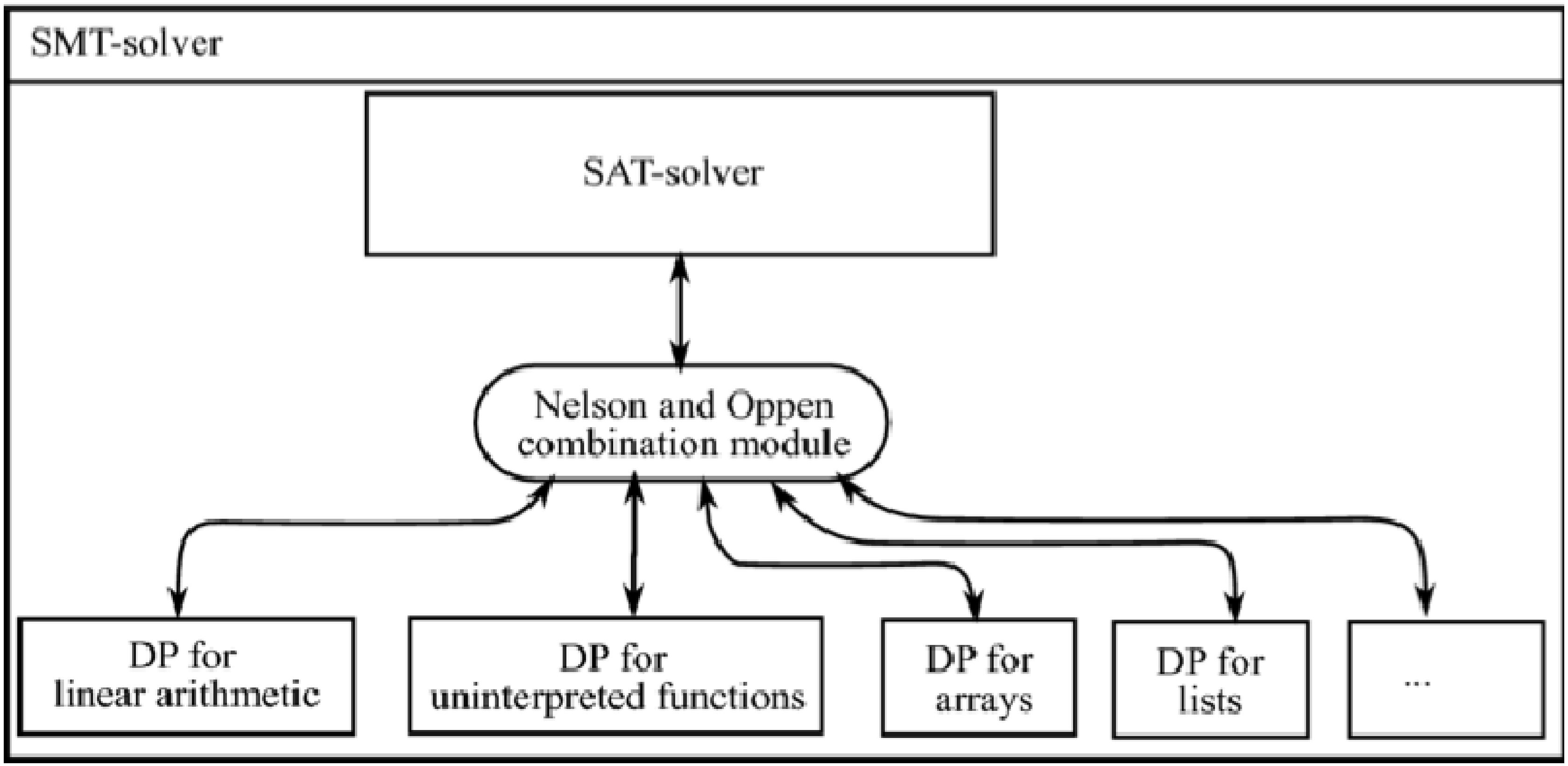
Lazy Approach: SAT + Theories I

- Independently developed by several groups
  - CVC (Stanford)
  - ICS (SRI)
  - MathSAT (Univ. Trento, Italy)
  - Verifun (HP)
- Motivated by the breakthroughs in SAT solving
  - DPLL algorithm
  - Various optimizations and heuristics

Lazy Approach: SAT + Theories II

- SAT solver
  - Manages the boolean structure and assigns truth values to the atoms in a formula
- Theory solvers
  - Efficiently validate (partial) assignments produced by the SAT solver
- When a theory solver detects unsatisfiability, a new clause (lemma) is created

Basic architecture



Naïve Approach

- Example
  - Suppose SAT solver assigns
    $\{x = y \rightarrow T, y = z \rightarrow T, f(x) = f(z) \rightarrow F\}$
  - Theory solver detects conflict
  - Lemma is created
    $\neg(x = y) \vee \neg(y = z) \vee f(x) = f(z)$
- Potential problems
  - Lemmas are imprecise (not minimal)
  - Theory solver is "passive"
    - It just detects conflicts
    - There is no propagation step
  - Backtracking is expensive
    - Restart from scratch when a conflict is detected

Theory Solvers

- Basic requirements
  - Deduce equalities between variables
  - Compute lemmas (conflict sets)
    - As precise as possible
- Extra desired features
  - Theory propagation
  - Incrementality
  - Backtracking

Equality Generation

- Combination of theories strongly relies on the propagation of deduced equalities
- Every theory solver has to support it

Precise Lemmas I

- Example
  - $\{a_1 = T, a_2 = F, a_3 = F\}$ is inconsistent
  - Lemma is $\neg a_1 \vee a_2 \vee a_3$
- An inconsistent set A is redundant if $A' \subset A$ is also inconsistent
- Redundant inconsistent sets imply
  - Imprecise lemmas
  - Ineffective pruning of the search space

Precise Lemmas II

- Noise of a redundant set is $A \setminus A_{min}$
- Imprecise lemma is useless in any partial assignment where an atom in the noise has a different assignment
- Example
  - Suppose a_1 is in the noise
  - Then $\neg a_1 \vee a_2 \vee a_3$ is useless when $a_1 = F$

Theory Propagation

- SAT solver is assigning truth values to the atoms in a formula
- Partial assignment produced by the SAT solver may imply truth values of unassigned atoms
- Example

  $x = y \wedge y = z \wedge (f(x) \neq f(z) \vee f(x) = f(w))$

  Partial assignment $\{x = y \rightarrow T, y = z \rightarrow T\}$ implies $f(x) = f(z)$
- Reduces the number of conflicts and the search space

Incrementality

- Theory solvers constantly receive new constraints and restart the process
  - Augmented partial assignments from SAT solver
  - Equalities coming from other theory solvers
- Do not restart from scratch
  - Reuse what you learned so far

Efficient Backtracking

- One of the most important improvements in SAT was efficient backtracking
- Extreme (inefficient) approach in theory solvers
  - Restart from scratch on every conflict
- Efficient approach
  - Restore to a logically equivalent state
- Backtracking should be included in the design of theory solvers

Ideal Theory Solver

- Efficient in real benchmarks
- Produces precise lemmas
- Supports theory propagation
- Incremental
- Efficient backtracking

Dealing with Quantifiers

Quantifier Instantiation

- SMT solvers use heuristic quantifier instantiation using E-matching (matching modulo equalities)
- Divide input formula into ground and quantified portion
- Check ground portion for satisfiability
  - If SAT then extend with ground terms instantiated from the quantified part
    - Often leverage user-provided *triggers*
  - If UNSAT then report UNSAT
- Repeat

Example

$\forall$ x: f(g(x)) = x { f(g(x)) } (trigger)

a = g(b),

b = c,

f(a) $\neq$ c

Limitations

- Users often have to manually provide patterns
  - Automatic inference of patterns is fragile
- Bad user provided patterns
  - False positives (wrong SAT answers)
  - Nonterminating executions

Trigger too Restrictive

$\forall$ x: f(g(x)) = x { f(g(x)) }

g(a) = c,

g(b) = c,

a $\neq$ b

- Results in false positives

Trigger too Restrictive

- More “liberal” pattern:
  $\forall$ x: f(g(x)) = x { g(x) }
  g(a) = c,
  g(b) = c,
  a $\neq$ b
- Instantiate:
  f(g(a)) = a,
  f(g(b)) = b
- Implies that a=b

Matching Loop

$\forall\, x{:}\ f(x) = g(f(x))\ \ \{\ f(x)\ \ \}$
$\forall\, x{:}\ g(x) = f(g(x))\ \{\ g(x)\ \}$
$f(a) = c$

- Instantiate:
  $f(a) = g(f(a))$
  $g(f(a)) = f(g(f(a)))$
  …

- Results in executions that do not terminate