Design of Theory Solvers in CVC4

Andrew Reynolds
8 April 2014
Satisfiability Modulo Theories

• SMT solvers used for:
  – Software/hardware verification
  – Automated Theorem Proving
  – Scheduling and Planning
SMT Solver : CVC4

• Joint project between NYU and U of Iowa
• State of the art successor of CVC3
• Based on DPLL(T) framework
• Supports wide range of theories
Theories supported by CVC4

• From SMT Lib:
  – Uninterpreted functions
  – Linear Integer and Real Arithmetic
  – Arrays
  – BitVectors

• Others:
  – Inductive Datatypes
  – Strings
  – Sets
  – Floating Points (coming soon)
SMT Example

\[ x + 5 = \text{read}(A, 5) \land (A \neq B \lor \text{read}(B, 5) \leq f(x)) \]
SMT Example

\[ x + 5 = \text{read}( A, 5 ) \land ( A \neq B \lor \text{read}( B, 5 ) \leq f(x) ) \]

\[ \Downarrow \text{Abstract to Propositional Logic} \]

\[ P \land ( Q \lor R ) \]
SMT Example

\[ x + 5 = \text{read}( A, 5 ) \land ( A \neq B \lor \text{read}( B, 5 ) \leq f(x) ) \]

\[ \Downarrow \text{Abstract to Propositional Logic} \]

\[ P \land ( Q \lor R ) \]

true \quad \text{true}
SMT Example

\[ x + 5 = \text{read}(A, 5) \land (A \neq B \lor \text{read}(B, 5) \leq f(x)) \]

\[ \Downarrow \text{Abstract to Propositional Logic} \]

\[ P \land (Q \lor R) \]

\[ \Downarrow \text{Find satisfying assignment} \]

\[ x + 5 = \text{read}(A, 5), A \neq B \]

\[ \Rightarrow \text{Determine if consistent according to theory} \]
DPLL(T) Framework

**Input:** Clause set $F$

**SAT Solver**
- $F$ is sat
- $F$ is unsat
- UNSAT
- Clauses to add to $F$

**Theory Solvers**
- Satisfying Assignment $M$
- M is T-consistent
- M is T-inconsistent
- SAT

The framework iterates between SAT solving and theory solving to determine satisfiability.
Architecture of CVC4

• CVC4 combines:
  – Off-the-shelf SAT solver (MiniSAT)
  – Multiple theory solvers
• Managed by Theory Engine
CVC4 : Theory Engine

Clause set F

SAT Solver

Theory Engine

Distribute literals from M to theory solver(s)

Theory UF
Theory Arith
Theory Arrays

Satisfying Assignment M
CVC4 : Theory Engine

Clause set F 

Satisfying Assignment M

**SAT Solver**

**Theory Engine**
- Distribute literals from M to theory solver(s)
- Ask theory solvers whether their subset of M is T-consistent

Theory Engine:
- Theory UF
- Theory Arith
- Theory Arrays
- ...

Clauses to add to F
Ask if $M_X$ is consistent

Assertion Queue

"$M_X$"

Conflict: $C \subseteq M_X$ is T-inconsistent, add $\neg C$ to F

Lemma(s): Add a clause D to F

Theory X
Handling Equality

• Challenge: *Equality* reasoning is common to all theories, e.g.
  
  – $x + 1 = y$
  – `read(A, i) ≠ read(A, j)`
  – $l_1 = \text{cons}(e, l_2)$

• *Idea*: Theory solvers use *Equality Engine* data structure
Equality Engine Data Structure

- Takes input a set of equalities and disequalities
  - Performs Congruence Closure
  - Maintains equivalence classes
  - Explains/reports conflicts

Equality Engine

\[ a = c, \quad f(b) = a, \quad f(a) \neq c \]

- Conflict?
Theory Solver: Equality Engine

Input Assertions

Is $M_X$ consistent?

$M_X$

Inferred Assertions

Theory Reasoning

Equality Engine

Conflict

Lemma(s)
Case: Inductive Datatypes

Equality Engine

\[
\text{cons}(e, l_1) = \text{cons}(e, l_3), \\
l_1 = l_2, \\
l_2 \neq l_3
\]

Theory

Reasoning

\[
\text{cons}(e, l_1), \\
\text{cons}(e, l_3) \\
l_3 \neq e
\]
Case: Inductive Datatypes

Equality Engine

From \( \text{cons}(e, l_1) = \text{cons}(e, l_3) \), we can infer \( l_1 = l_3 \).
Case : Inductive Datatypes

Equality Engine

From \( \text{cons}(e, l_1) = \text{cons}(e, l_3) \),
\l_1 = l_2, 
\l_2 \not= l_3 

We have conflict:
\( \text{cons}(e, l_1) = \text{cons}(e, l_3) \),
\l_1 = l_2, 
\l_2 \not= l_3 

From \( \l_1 = l_3, \)
\l_1 \not= l_3 

Theory Reasoning
Theory Solver (summary)

• Most theory solvers rely on Equality Engine for:
  – Computing equivalence classes of current terms
  – Reporting most conflicts
  – Performing (eager) T-propagation

• Supplement with Theory Reasoning:
  – Adds assertions inferred from current state
  – May add other lemmas to system when necessary
Theory Interface (extended)

- In addition to check if assertions T-consistent,
  - propagate, T-propagate literals
  - explain, explain why literals were T-propagated
  - collectModelInfo, get model for curr assertions
  - Others:
    - getNextDecision
    - staticLearn
    - preSolve
    - ...

Theory X
Support for Theory Development

• Equality Engine data structure
• Associated “kinds” file
  – Contains specifications for:
    • Signature Definition (symbols in the theory)
    • Term normalization
    • Type checking
    • Properties of the theory
      – Interaction when performing theory combination
  – Auto-generates necessary code for each of these
• Automatic Integration into Theory Engine
Questions?
DPLL(T) Search : Incremental Checking

Propagate : Literals $p_1, \ldots, p_n$ must be added to $M$

Decide : Add literal $d$ to $M$
DPLL(T) Search : Incremental Checking

Propagate : Literals $p_1, \ldots, p_n$ must be added to $M$

Decide : Add literal $d$ to $M$

Weak Effort Check

- Check if $p_1, \ldots, p_n$ are already $T$-inconsistent
- Should be efficient
- Can be incomplete
DPLL(T) Search: Incremental Checking

Propagate: Literals $p_1, \ldots, p_n$ must be added to $M$.

Decide: Add literal $d$ to $M$.

Weak Effort Check

- M is a complete assignment
- Determine if $M$ is T-inconsistent
- Must be complete

Strong Effort Check

- Determine if $M$ is T-inconsistent
- Must be complete