Basics of SMT Solving Algorithms and Theories

Ian Johnson

April 29, 2009

Outline

Outline

Vocabulary and Preliminaries

Previous Strategies
- Eager Approaches
- Lazy Approaches

The DPLL(T) Framework

Useful Theories

Combining Theories

Reading

What is the SMT problem?

SMT stands for Satisfiability Modulo Theories, and is essentially a generalization of the SAT problem.
We say "modulo theories" because the Boolean predicates of SAT are now first order sentences in a logic.

- At least NP-complete and at most unbounded complexity problem with applications in AI, formal methods
- Input usually given in SMT-LIB format (CNF with sugar)

Definitions

- A theory T is a set of first order sentences.
- A formula F is *T-satisfiable* or *T-consistent* if $F \wedge T$ is satisfiable in the first order sense. Otherwise F is *T-inconsistent*.
- A partial assignment M is a T-model of a formula F if M is a T-consistent partial assignment and $M \models F$ (in the propositional sense).
- For two formulas F and G, we say $F \models_T G$ if $F \wedge \neg G$ is T-inconsistent.
- A *theory lemma* is a clause C such that $\emptyset \models_T C$.
- A *T-solver* is a decision* procedure that decides the T-satisfiability of conjunctions of ground literals.

A Brief Look at DPLL

Based on the idea of unit propagation:

$$M \parallel F, C \vee l \Rightarrow Ml \parallel F, C \vee l \text{ if } \begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases}$$

conflict-driven backjumping:

$$Ml^d N \parallel F, C \Rightarrow Ml' \parallel F, C \text{ if } \begin{cases} Ml^d N \models \neg C \text{ and there is} \\ \text{some clause } C' \vee l' \text{ such that:} \\ \quad F, C \models C' \vee l' \text{ and } M \models \neg C', \\ \quad l' \text{ is undefined in } M, \text{ and} \\ \quad l' \text{ or } \neg l' \text{occurs in } F \text{ or in } Ml^d N \end{cases}$$

and conflict-driven learning:

$$M \parallel F \Rightarrow M \parallel F, C \text{ if } \begin{cases} \text{each atom of } C \text{ occurs in } F \text{ or in } M \\ F \models C \end{cases}$$

Outline

Propositional Translation

Satisfiability-preserving translation to a propositional CNF formula.

Pros:

- Easy to do translations.
- Leverages the existant SAT-solving technology.

Cons:

- Not all theories can be translated this way.
- Translation causes exponential blow-up.
- Search starts only after entire problem is translated.
- Size of the problem usually consumes all resources before starting the search.

Calling External SAT Solver

The T-solver calls a SAT solver on the formula to get a (propositionally) satisfying assignment and checks its T-consistency. If inconsistent, the conflicting clause is added to the formula and sent back to the SAT solver.

Pros:

- Only have to write the T-solver.
- Again leverages the existant SAT-solving technology.

Cons:

- Search must complete entirely before T-inconsistency is reported.
- Search must start over at the beginning if last assignment failed.

Incremental T-solving

Communicates with the DPLL module to inform of T-inconsistency before an entire model is constructed, either at every decision or on every k decisions.
Pros:

- Early pruning of search space.

Cons:

- Not always effective. The T-solver should be faster in processing one additional input literal than in reprocessing from scratch, but for some theories this is impossible.
- Finding the "sweet spot", or the right k for the best performance is guess work.

On-line SAT solving

Builds off the incremental approach by allowing the T-solver to generate conflicting clauses to aid with backjumping.
Pros:

- Early pruning
- More aggressive pruning.

Cons:

- Conflicting clauses hard to generate.

Theory Propagation

Guides the search process by taking the current partial assignment and deriving other subterms of the formula. T-solver is no longer a *validator* for the DPLL search.

Pros:

- Analagous to the importance of unit propagation in DPLL.
- For many theories, this process exhaustively executed gives a great increase of performance.
- Exhaustively executed, this eliminates the need for unit propagation on theory lemmas.

Cons:

- Conflict analysis highly non-trivial.
- If not performed exhaustively, duplicate results are extraneously generated.

Outline

Transactions between DPLL and T-Solver

Sufficient communication for an incremental on-line solver with theory propagation is given in the following set of messages:

- Notify T-Solver that a certain literal has been set to true.
- Ask T-Solver to check the current partial assignment is T-inconsistent (with *strength*) and give an *explanation*.
- Ask T-Solver to identify currently undefined input literals that are T-consequences of M.
- Ask T-Solver to provide a justification for a T-entailment of a theory-propagated literal for conflict clause learning.
- Ask T-Solver to undo the last n notifications that a literal has been set to true.

Outline

Equality of Uninterpreted Functions (EUF)

Finds basic unsatisfiable errors such as

$$(f(f(a)) \neq b \vee f(f(f(b))) \neq b) \wedge f(a) = a \wedge a = b$$

by using a congruence closure algorithm to create congruence classes and checking the results against a list of suspected equalities and disequalities, inconsistencies are found.

Difference Logic

(Still NP-Complete) subset of integer linear arithmetic problems where all constraints are of the form

$$x - y \leq c$$

Questions in bounded model checking of timed automata along with questions of circuit timing analysis can be answered with this logic. Naïvely solvable using an iterative Bellman-Ford method, but better negative-weight cycle detection algorithms are known.

Bit-Vectors

NP-Complete theory that allows fixed-width bit vectors to have the following operators executed on them: Assignment $=$, named selection $[i : j]$, concatenation ::, arithmetic $\{+, -, *, <\}$ where $*$ is multiplication by a scalar, and bitwise operators $\{\textbf{AND}, \textbf{OR}, \textbf{NOT}\}$

Interpreted Sets and Bounded Quantification

An expressive NP-Complete theory of heap-manipulating loop-free and procedure-free programs. Strategy for proving a program T correct: compute $wp(T, true)$ and decide satisfiability of $\neg wp(T, true)$, giving $\neg wp(T, true)$ is unsatisfiable if and only if T does not go wrong.

$$\begin{aligned} T \in Stmt ::= & Assert(\varphi) \mid Assume(\varphi) \mid \\ & x := new \mid free(x) \mid x := t \mid \\ & f(x) := y \mid T_1; T_2 \mid T_1 \Box T_2 \end{aligned}$$

$$\begin{array}{lll} c & \in Integer & \\ x & \in Variable & \\ f & \in Function & \\ \varphi & \in Formula & ::= \alpha \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \\ \alpha & \in \forall Formula & ::= \gamma \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \wedge \alpha_2 \mid \forall x \in S.\alpha \\ \gamma & \in GFormula & ::= t_1 = t_2 \mid t_1 < t_2 \mid t_1 \xrightarrow{f} t_2 \xrightarrow{f} t_3 \mid \neg\gamma \\ t & \in Term & ::= c \mid x \mid t_1 - t_2 \mid t_1 + t_2 \mid f(t) \mid ite(t = t', t_1, t_2) \\ S & \in Set & ::= g^{-1}(t) \mid Btwn(f, t_1, t_2) \end{array}$$

Figure: Program statement syntax (top) and formula syntax (bottom)

Outline

Nelson-Oppen Method

A solver for the union of theories T_1 and T_2 can be constructed using the Nelson-Oppen procedure if the two have disjoint signatures ($\Sigma_1 \cap \Sigma_2 = \{=\}$) and are stably infinite (i.e. every satisfiable quantifier-free formula is satisfiable in an infinite model).

For Γ a set of literals of $\Sigma_1 \cup \Sigma_2$, want to *purify* Γ into $\Gamma_1 \wedge \Gamma_2$ such that $\Gamma_i \subseteq \Sigma_i^\alpha$ for $\alpha = \{\mathcal{V}(\Gamma_1) \cap \mathcal{V}(\Gamma_2)\}$. A partition ϕ (conjuction of many equalities and disequalities) of α is guessed and the individual solvers return if $\Gamma_i \wedge \phi$ is satisfiable.

A theory is *convex* iff for for all finite sets Γ of literals and for all non-empty disjunctions $\bigvee_{i \in I} u_i \simeq v_i$ of variables, $\Gamma \models_T \bigvee_{i \in I} u_i \simeq v_i$ iff $\Gamma \models_T u_i \simeq v_i$ for some $i \in I$.

If the two theories are convex, then this guessing can be changed into deducing the correct partition by propagating equalities (let Γ_2 know if $T_1 \cup \Gamma_1 \models x \simeq y$ and vice versa).

Model-Based Method

(i) Each theory $\mathcal{T}_i$ maintains a model $\mathcal{M}_i$ for Γ_i (or a subset of Γ_i).
(ii) Sometimes if $u^{\mathcal{M}_i} = v^{\mathcal{M}_i}$ then a case split is introduced for $u \simeq v$.
(iii) To satisfy newly assigned literals or te imply fewer equalities, models can be changed.
Rules added to SMT:

$\mathcal{M}$-Propagate: $\mathcal{M}, \Gamma \parallel F \Rightarrow \mathcal{M}, \Gamma(u \simeq v)^d \parallel F$ if $\begin{cases} u, v \in \mathcal{V}, (u \simeq v) \notin \mathcal{L} \\ u^{\mathcal{M}_i} = v^{\mathcal{M}_i} \\ \text{add } (u \simeq v) \text{ to } \mathcal{L} \end{cases}$

$\mathcal{M}$-Mutate: $\mathcal{M}, \Gamma \parallel F \Rightarrow \mathcal{M}', \Gamma \parallel F$ if $\{\mathcal{M}'$ is some variant of $\mathcal{M}$.
To minimize case splits, equivalence classes are kept for an equivalence relation $R_{\mathcal{M}}(u, v) \iff u^{\mathcal{M}} = v^{\mathcal{M}}$.
(i) "Opportunistic equality propagation": eagerly propagate equality deductions.
(ii) "Postponing model-based equality propagation": delay applying the rule $\mathcal{M}$-Propagate until all existing case splits have been performed.
(iii) For a mutated model, create a more *diverse* model $\delta(\mathcal{M}_k)$, such that $|classes(R_{\mathcal{M}_k})| \leq |classes(R_{\delta(\mathcal{M}_k)})|$.

Outline

References

- R. Nieuwenhuis and A. Oliveras: Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)
- R. Bruttomesso et al: A Lazy and Layered SMT(BV) Solver for Hard Industrial Verification Problems
- S. Lahiri and S. Qadeer: Back to the Future: Revisiting Precise Program Verification using SMT Solvers
- L. de Moura et al: A Tutorial on Satisfiability Modulo Theories
- L. de Moura and Nikolaj Bjørner: Model-Based Theory Combination