

Integrating Decision Procedures with Boolean Solving

Silvio Ranise & Christophe Ringeissen

Procédures de décision et vérification de programmes:

Lecture 5

January 6, 2010

Topics

- **GOAL**: **lift** decision procedures for the satisfiability problem of **conjunctions** of (ground) literals to handle **arbitrary Boolean combinations** of (ground) literals
- Key technique: **integrate**
 - ① decision procedure for conjunctions of ground literals **with**
 - ② a Boolean solver to handle the Boolean structure of the formula **via**
 - ③ an **abstract-check-refine** schema
- Systems based on this idea are called **(Lazy) Satisfiability Modulo Theory (SMT) Solvers**

Pre-requisites

- There is one key pre-requisite to this lecture
 - ① **DPLL** algorithm of Lecture 1 (or... how to build a Boolean model of a propositional formula)
- For ease of reference, the following slide contains the essential concept which will be used in the rest of this lecture...

Pre-requisite: the DPLL algorithm

Let S be a set of clauses

$$\textit{Unit Resolution} \quad \frac{S \cup \{L, C \vee \bar{L}\}}{S \cup \{L, C\}} \quad \text{if } \begin{array}{l} \bar{\bar{A}} := A \\ \bar{A} := \neg A \end{array}$$

$$\textit{Unit Subsumption} \quad \frac{S \cup \{L, C \vee L\}}{S \cup \{L\}}$$

$$\textit{Splitting} \quad \frac{S}{S \cup \{A\} \mid S \cup \{\neg A\}} \quad \text{if } A \text{ is an atom occurring in } S$$

Exercise (mandatory): explain how a Boolean model can be extracted from the application of these rules

(**Hint**: think of the derivation trees and collect the various unit clauses...)

Index

- 1 Satisfiability Modulo Theory: introduction
 - Conflict sets and conflict clauses
- 2 SMT Solvers: an algorithmic view
 - SMT solver: a refinement of the basic algorithm
- 3 Bibliography on SMT solving

What about Arbitrary Boolean Combinations?

- Let ϕ be an arbitrary Boolean combinations of atoms in a given theory T with decidable satisfiability problem
 - How can we check the T -satisfiability of ϕ ?
 - ▷ convert ϕ into DNF $\bigvee_i \Phi_i$ and check each disjunct Φ_i for T -satisfiability
 - **Main problem**: conversion to DNF may result in a blow-up in space of the resulting formula
 - Unfortunately, the problem is very hard since it subsumes the propositional satisfiability problem which is NP
 - ▷ we are interested in developing efficient solutions (in practice)
- ➡ engineering rather than logic

Satisfiability Modulo Theory (SMT) Systems

<http://combination.cs.uiowa.edu/smtlib>

- Class of systems based on a the following idea: **integrate**
 - ▷ a Boolean solver (to handle the propositional structure of the formula)
 - ▷ a satisfiability procedure for the theory T
- All **lazy** SMT solvers the systems are based on an **abstract-check-refine** process
- There also exist **eager** SMT solvers that reduce satisfiability modulo theory to a pure Boolean satisfiability problem upfront and then use a SAT solver

Lazy SMT: Generic System Architecture

- Based on an **abstract-check-refine** process:
 - ▷ the input formula ϕ is **abstracted** to a propositional formula ϕ^p
 - ▷ the Boolean solver enumerates the propositional assignments $\beta_1^p, \dots, \beta_n^p$ of ϕ^p
 - ▷ each conjunction β_i of T -literals is **checked** for T -satisfiability
 - ▷ in case of T -unsatisfiability of β_i , **refine** the abstract propositional formula

Observations on SMT (I)

- Use an **off-the-shelf** Boolean solver to leverage its advantages, e.g.

- ▷ SAT solvers can scale up
- ▷ they **return just one propositional satisfying assignment**
- ▷ they can accept incrementally new clauses...

Scenario: let β^p be a propositional assignment of ϕ^p such that β ($\beta^p = \text{abs}(\beta)$) is T -unsatisfiable

How can we modify ϕ^p so that it return a new satisfying assignment distinct from β^p ?

$$\phi^p := \phi^p \wedge \neg\beta^p$$

Observations on SMT (II)

- Did we really get a practically efficient system?
 - ▷ Well... no! Since we enumerate all possible satisfying assignments of ϕ^P which may be exponentially many in the number of propositional literals in ϕ or, equivalently, we have still to check exponentially many the T -satisfiability of a conjunction of literals
 - ▷ So, even though the satisfiability procedure for T is very fast, it is invoked an exponential number of times and this may result in unacceptable degradation of the performances of the system
- How can we **improve this situation in practice**? In theory, this exponential blow-up is unavoidable!

Observations on SMT (II)

- To overcome this problem in practice, all SMT systems are based on the same technique which is a refinement of the technique to modify ϕ^p so that the Boolean solver return a new satisfying assignment distinct from β^p , namely:

$$\phi^p := \phi^p \wedge \neg\beta^p$$

- We assume that the satisfiability procedure for T has a more complex interface: besides returning sat/unsat, it also returns a **subset** π of the input set β of the literals which is still T -unsatisfiable so that we can perform the following

$$\phi^p := \phi^p \wedge \neg\pi^p$$

- ▷ Why should this be beneficial in practice ?

Observations on SMT (II)

- Let us consider the following (unsatisfiable) conjunction of literals in the theory of uninterpreted function symbols:

$$\beta := \ell_1 \wedge \cdots \wedge f(x) \neq f(y) \wedge x = y \wedge \cdots \wedge \ell_n$$

where each ℓ_i is an equality for $i = 1, \dots, n$

▷ If we use the naïve approach of enumerating all possible assignment, we will consider 2^{n-2} conjunctions of literals (corresponding to 2^{n-2} Boolean assignments) whose T -unsatisfiability is the responsibility of the same set $\{x = y, f(x) \neq f(y)\}$ of only two literals

▷ Instead, if we add the (propositional abstraction of the) negation of $\{x = y, f(x) \neq f(y)\}$ to ϕ^p , we will prune the search space of the Boolean solver of all the 2^{n-2} propositional assignments in **one shot!**

Observations on SMT (II)

- If $n = 100$ (which is a small/medium number for practical applications), then

we avoid invoking 2^{98} , i.e. around $3.2 \cdot 10^{29}$, prop. assignments!

- In practice, this **situation is quite common**: unsatisfiability of large conjunction of literals usually depends on a small sub-set of the input literals
- So, this technique gives spectacular results in practice and makes SMT system capable of tackling very large Boolean combinations of atoms in a given theory
 - ▷ the T -unsatisfiable subset π of the input literals is called **conflict set** and $\neg\pi$ is called a **conflict clause**

How can we obtain conflict sets from sat. proc?

- Naïve approach: by **guessing**
 - ▷ after detecting the unsatisfiability of a set β of literals...
 - ▷ pick a literal ℓ in β and check the unsatisfiability of $\beta \setminus \{\ell\}$: if it is still unsatisfiable, then $\beta := \beta \setminus \{\ell\}$; otherwise pick another literal
 - ▷ stop when all literals have been considered and return β as the conflict set
- **Pros**: very little effort in implementation
- **Cons**: not very efficient since each Boolean assignment, there are n calls to the satisfiability procedure (this is a linear blow-up but it may be significant for applications especially if the satisfiability procedure is quadratic or cubic)

About the “quality” of conflict set?

- The smaller (in **cardinality**) the conflict set, the higher the number of propositional assignments are eliminated
 - ▷ This requires to consider all subsets of a given set: an exponential number **⇒** not practical!
- So, we focus on **minimal** conflict set: a conflict set is minimal if all its (strict) subsets are satisfiable
 - ▷ Unfortunately, even for simple theories such as the theory of uninterpreted function symbol, the problem is NP-hard!
- In practice, we are already happy with **approximations of minimal conflict sets**

SMT System: main algorithm

```
function Bool +  $T$  ( $\phi$ : quantifier-free formula)  
   $\phi^p \leftarrow nf(fol2prop(\phi))$   
  while Bool-satisfiable( $\phi^p$ ) do  
     $\beta^p \leftarrow pick\_assign(\phi^p)$   
     $(\rho, \pi) \leftarrow T\text{-satisfiable}(prop2fol(\beta^p))$   
    if  $\rho = sat$  then return sat  
     $\phi^p \leftarrow \phi^p \wedge \neg fol2prop(\pi)$   
  end while  
  return unsat  
end
```

SMT System: main algorithm (remarks)

- **Propositional abstraction:** $fol2prop(\phi)$ returns the propositional abstraction ϕ^p of ϕ and $fol2prop$ is its inverse
 - ▷ $fol2prop(\phi)$ maps each atom in ϕ to a propositional letter, then it is extended homomorphically over the Boolean structure of ϕ
 - ▷ nf computes a normal form (e.g. CNF) of the input formula so that the Boolean solver is capable of handling it (SAT solvers can handle only formulae in CNF)

Computation of CNF

Let $\Phi = (a_1 \wedge \dots \wedge a_m) \vee (b_1 \wedge \dots \wedge b_n)$

Equivalent CNF:

$$\bigwedge_{i=1}^m \bigwedge_{j=1}^n (a_i \vee b_j)$$

Equisatisfiable CNF:

$$(X \vee Y) \wedge (X \Leftrightarrow a_1 \wedge \dots \wedge a_m) \wedge (Y \Leftrightarrow b_1 \wedge \dots \wedge b_n)$$

where $(X \Leftrightarrow a_1 \wedge \dots \wedge a_m) \wedge (Y \Leftrightarrow b_1 \wedge \dots \wedge b_n)$ can be represented as a conjunction of clauses (Exercise).

SMT System: main algorithm (remarks)

- The **Boolean solver** is supposed to provide two interface functionalities:
 - ▷ *Bool-satisfiable* which checks whether the input formula is propositionally satisfiable or not
 - ▷ *pick_assign* returns a satisfying assignment of the input formula
- It is usually also assumed that it is possible to **incrementally add** a clause to those already in the solver
 - ▷ Notice that this requirement is not crucial: SAT solvers are so fast that even restarting them from scratch each time a conflict clause is added is not dramatic in term of performances!

SMT System: main algorithm (remarks)

- The **satisfiability procedure** for a theory T is supposed to have the following interface: T -*satisfiable*(β) returns the pair (ρ, π) where
 - ▷ ρ is sat when β is T -satisfiable (π is left unspecified in this case)
 - ▷ ρ is unsat when β is T -unsatisfiable and $\pi \subseteq \beta$ is T -unsatisfiable

SMT System: main algorithm (remarks)

- The main loop of the algorithm searches a propositional assignment whose refinement is T -satisfiable
 - ▷ If such an assignment is found, then the **satisfiability** of the input formula is returned
 - ▷ If all assignments have been found unsatisfiable, then the **unsatisfiability** of the input formula is reported

SMT System: correctness of the main algorithm

- **Lemma** [Return on sat] ϕ is T -satisfiable iff there exists an assignment β^p of ϕ^p such that
 - ▷ β^p (propositionally) satisfies ϕ^p
 - ▷ β is T -satisfiable
- **Lemma** [Loop invariant] If β^p is a set of literals satisfying ϕ^p and β is T -unsatisfiable, then
 - ▷ ϕ is T -satisfiable iff $\phi \wedge \neg\beta$ is

SMT System: correctness of the main algorithm

- **Lemma** [Loop variant] If β^P is a set of literals satisfying ϕ^P and β is T -unsatisfiable, then
 - ▷ the number of propositional assignments of $\phi^P \wedge \neg\beta^P$ are strictly less than those of ϕ^P
- **Theorem** [Correctness] The function $\text{Bool} + T$ is a decision procedure for checking the satisfiability of arbitrary Boolean combination of ground atoms

SMT System: refining the main algorithm

- **Observation:** SAT solvers are capable of incrementally building a satisfying assignment...
 - ▷ it would be interesting to incrementally check the T -satisfiability of the corresponding literals while the assignment is built
 - ▷ in this way, we will discover unsatisfiability early and prune the search space of the Boolean solver with a “better” conflict set
- The idea is that before a split on a certain propositional letter (a non-deterministic step), it is wise to check whether the satisfying assignment built so far is satisfiable: if it is not, there is no point to continue and create a branching point!
 - ▷ the conflict set returned will contain literals higher-up in the search tree and this prunes substantially the search space of the SAT solver

SMT System: refining the main algorithm

```
function Bool +  $T$  ( $\phi$ : quantifier-free formula)  
   $\phi^p \leftarrow nf(fol2prop(\phi))$   
  while Bool-satisfiable( $\phi^p$ ) do  
     $\beta^p \leftarrow \emptyset$   
    repeat  
       $\beta^p \leftarrow extend\_partial\_assign(\phi^p, \beta^p)$   
       $(\rho, \pi) \leftarrow T\text{-satisfiable}(prop2fol(\beta^p))$   
      if ( $\rho = \text{sat} \wedge |\beta^p| = |Atoms(\phi)|$ ) then return sat  
    until  $\rho = \text{unsat}$   
     $\phi^p \leftarrow \phi^p \wedge \neg fol2prop(\pi)$   
  end while  
  return unsat  
end
```

SMT System: refining the main algorithm

- The new function $extend_partial_assign(\phi^p, \beta^p)$ monotonically extends the input (partial) assignment
 - ▷ a possible implementation of this is to eagerly perform all the computations of the DPLL algorithm which do not create branching point and then return the resulting assignment so that its refinement can be checked for T -satisfiability
- Notice that we can only report the satisfiability of the input formula when a **total** assignment is found T -satisfiable
- The inner loop of the new version of the algorithm must terminate since $|Atoms(\phi)| - |\beta^p|$ is monotonically decreasing and $|Atoms(\phi)| - |\beta^p| \geq 0$

Suggested reading

- Overview on SMT solvers (both eager and lazy approaches)
S. Ranise and C. Tinelli. “*Satisfiability Modulo Theories.*” In TRENDS and CONTROVERSIES—IEEE Magazine on Intelligent Systems, Volume 21, Number 6, pages 71–81, November/December 2006. (Pre-print available at <http://www.loria.fr/~ranise/pubs/IEEE-pp.pdf>)
- In-depth discussion about lazy SMT solvers
R. Sebastiani. “*Lazy Satisfiability Modulo Theories.*” In Journal on Satisfiability, Boolean Modeling and Computation, JSAT. Vol. 3, 2007. Pag 141–224, © IOS Press. (Available at http://jsat.ewi.tudelft.nl/content/volume3/JSAT3_9_Sebastiani.pdf)
C. Barrett, R. Sebastiani, S. Seshia and C. Tinelli. Chapter on “*Satisfiability Modulo Theories.*” In A. Biere, H. van Maaren and T. Walsh editors, Handbook on Satisfiability. IOS Press, February 2009. (Pre-print available at <http://www.cs.uiowa.edu/~tinelli/html/publications.html>)