SMT Solvers

Maria João Frade

HASLab - INESC TEC
Departamento de Informática, Universidade do Minho

2018/2019

Roadmap

- **SMT solvers**
  - main features;
  - SMT and SAT solvers integration: "eager" vs "lazy" approach;
  - the basic "lazy offline" approach and its enhancements;
  - minimal unsat core: basic ways to comput it;
  - DPLL($\mathcal{T}$) framework.
- **Practice with a SMT solver**

The SMT problem

- The *Satisfiability Modulo Theory (SMT) problem* is a variation of the SAT problem for first-order logic, with the interpretation of symbols constrained by a specific theory (i.e., it is the problem of determining, for a theory $\mathcal{T}$ and given a formula ϕ, whether ϕ is $\mathcal{T}$-satisfiable).
- An *SMT solver* is a tool for deciding satisfiability of a FOL formula with respect to some background theory.
- Common first-order theories SMT solvers reason about:
  - Equality and uninterpreted functions
  - Arithmetics: rationals, integers, reals, difference logic, ...
  - Bit-vectors, arrays, ...
- In practice, one needs to work with a combination of theories.

$$x + 2 = y \to f(read(write(a, x, 3), y - 2)) = f(y - x + 1)$$

Often decision procedures for each theory combine modularly.

SMT solvers

- SMT solvers have gained enormous popularity over the last several years.
- Wide range of applications: software verification, program analysis, test case generation, model checking, scheduling, . . .
- Many existing off-the-shelf SMT solvers:
  - Z3 (Microsoft Research)
  - Yices (SRI, USA)
  - CVC3, CVC4 (NYU & U. Iowa, USA)
  - Alt-Ergo (LRI, France)
  - MathSAT (U Trento, Italy)
  - Barcelogic (UP Catalunya, Spain)
  - Beaver (UC Berkeley, USA)
  - Boolector (FMV, Austria)
  - ...
- SMT solving is active research topic today (see: http://www.smtlib.org)

Solving SMT problems

- For a lot of theories one has (efficient) decision procedures for a limited kind of input problems: **sets (or conjunctions) of literals**.
- In practice, we do not have just sets of literals.
  - We have to deal with: arbitrary Boolean combinations of literals.

How to extend theory solvers to work with arbitrary quantifier-free formulas?

- Naive solution: convert the formula in DNF and check if any of its disjuncts (which are conjunctions of literals) is $\mathcal{T}$-satisfiable.
- In reality, this is completely impractical: DNF conversion can yield exponentially larger formula.
- Current solution: exploit propositional SAT technology

Lifting SAT technology to SMT

How to deal efficiently with boolean complex combinations of atoms in a theory?

- Two main approaches:
  - **Eager approach**
    - translate into an equisatisfiable propositional formula
    - feed it to any SAT solver
  - **Lazy approach**
    - abstract the input formula to a propositional one
    - feed it to a (DPLL-based) SAT solver
    - use a theory decision procedure to refine the formula and guide the SAT solver
- According to many empirical studies, lazy approach performs better than the eager approach.
- We will only focus on the lazy approach.

The "eager" approach

- **Methodology:**
  - Translate into an equisatisfiable propositional formula.
  - Feed it to any SAT solver.
- **Why "eager"?** Search uses all theory information from the beginning.
- **Characteristics:** Sophisticated encodings are needed for each theory.
- **Tools:** UCLID, STP, Boolector, Beaver, Spear, ...

The "lazy" approach

- **Methodology:**
  - Abstract the input formula to a propositional one.
  - Feed it to a (DPLL-based) SAT solver.
  - Use a theory decision procedure to refine the formula and guide the SAT solver.
- **Why "lazy"?** Theory information used lazily when checking $\mathcal{T}$-consistency of propositional models.
- **Characteristics:**
  - SAT solver and theory solver continuously interact.
  - Modular and flexible.
- **Tools:** Z3, Yices, MathSAT, CVC4, Barcelogic, ...

Boolean abstraction

- Define a bijective function prop, called *boolean abstraction function*, that maps each SMT formula to a overapproximate SAT formula.

Given a formula ψ with atoms $\{a_1, \ldots, a_n\}$ and a set of propositional variables $\{P_1, \ldots, P_n\}$ not occurring in ψ,

- The *abstraction mapping*, prop, from formulas over $\{a_1, \ldots, a_n\}$ to propositional formulas over $\{P_1, \ldots, P_n\}$, is defined as the homomorphism induced by $\mathsf{prop}(a_i) = P_i$.
- The inverse prop^{-1} simply replaces propositional variables P_i with their associated atom a_i.

$$\psi : \quad \underbrace{g(a) = c}_{P_1} \wedge (\underbrace{f(g(a)) \neq f(c)}_{\neg P_2} \vee \underbrace{g(a) = d}_{P_3}) \wedge \underbrace{c \neq d}_{\neg P_4}$$

$$\mathsf{prop}(\psi) : \quad P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$$

Boolean abstraction

$$\psi : \quad \underbrace{g(a) = c}_{P_1} \wedge (\underbrace{f(g(a)) \neq f(c)}_{\neg P_2} \vee \underbrace{g(a) = d}_{P_3}) \wedge \underbrace{c \neq d}_{\neg P_4}$$

$$\mathsf{prop}(\psi) : \quad P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$$

- The boolean abstraction constructed this way overapproximates satisfiability of the formula.
  - Even if ψ is not $\mathcal{T}$-satisfiable, $\mathsf{prop}(\psi)$ can be satisfiable.
- However, if boolean abstraction $\mathsf{prop}(\psi)$ is unsatisfiable, then ψ is also unsatisfiable.

Boolean abstraction

For an assignment $\mathcal{A}$ of $\mathsf{prop}(\psi)$, let the set $\Phi(\mathcal{A})$ of first-order literals be defined as follows

$$\Phi(\mathcal{A}) = \{\mathsf{prop}^{-1}(P_i) \mid \mathcal{A}(P_i) = 1\} \cup \{\neg \mathsf{prop}^{-1}(P_i) \mid \mathcal{A}(P_i) = 0\}$$

$$\psi : \quad \underbrace{g(a) = c}_{P_1} \wedge (\underbrace{f(g(a)) \neq f(c)}_{\neg P_2} \vee \underbrace{g(a) = d}_{P_3}) \wedge \underbrace{c \neq d}_{\neg P_4}$$

$$\mathsf{prop}(\psi) : \quad P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$$

- Consider the SAT assignment for $\mathsf{prop}(\psi)$,

$$\mathcal{A} = \{P_1 \mapsto 1, P_2 \mapsto 0, P_4 \mapsto 0\}$$

$\Phi(\mathcal{A}) = \{g(a) = c, f(g(a)) \neq f(c), c \neq d\}$ is not $\mathcal{T}$-satisfiable.

- This is because $\mathcal{T}$-atoms that may be related to each other are abstracted using different boolean variables.

The "lazy" approach (simplest version)

- Given a CNF F, **SAT-Solver**(F) returns a tuple $(r, \mathcal{A})$ where r is SAT if F is satisfiable and UNSAT otherwise, and $\mathcal{A}$ is an assignment that satisfies F if r is SAT.
- Given a set of literals S, **T-Solver**(S) returns a tuple (r, J) where r is SAT if S is $\mathcal{T}$-satisfiable and UNSAT otherwise, and J is a justification if r is UNSAT.
- Given an $\mathcal{T}$-unsatisfiable set of literals S, a *justification* (a.k.a. *unsat core*) for S is any unsatisfiable subset J of S. A justification J is *non-redundant* (or *minimal*) if there is no strict subset J' of J that is also unsatisfiable.

The “lazy” approach (simplest version)

Basic SAT and theory solver integration

SMT-Solver (ψ) {
  $F \leftarrow$ prop(ψ)
  loop {
    $(r, \mathcal{A}) \leftarrow$ **SAT-Solver**(F)
    if $r =$ UNSAT **then return** UNSAT
    $(r, J) \leftarrow$ **T-Solver**$(\Phi(\mathcal{A}))$
    if $r =$ SAT **then return** SAT
    $C \leftarrow \bigvee_{B \in J} \neg$prop$(B)$
    $F \leftarrow F \wedge C$
  }
}

If a valuation $\mathcal{A}$ satisfying F is found, but $\Phi(\mathcal{A})$ is $\mathcal{T}$-unsatisfiable, we add to F a clause C which has the effect of excluding $\mathcal{A}$ when the SAT solver is invoked again in the next iteration. This clause is called a *“theory lemma”* or a *“theory conflict clause”*.

SMT-Solver($g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$)

- $F =$ prop$(\psi) = P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$
- SAT-Solver(F) = SAT, $\mathcal{A} = \{P_1 \mapsto 1, P_2 \mapsto 0, P_4 \mapsto 0\}$
- $\Phi(\mathcal{A}) = \{g(a) = c, f(g(a)) \neq f(c), c \neq d\}$
  T-Solver$(\Phi(\mathcal{A}))$ = UNSAT, $J = \{g(a) = c, f(g(a)) \neq f(c), c \neq d\}$
- $C = \neg P_1 \vee P_2 \vee P_4$

SMT-Solver($g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$)

- $F =$ prop$(\psi) = P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$
- SAT-Solver(F) = SAT, $\mathcal{A} = \{P_1 \mapsto 1, P_2 \mapsto 0, P_4 \mapsto 0\}$
- $\Phi(\mathcal{A}) = \{g(a) = c, f(g(a)) \neq f(c), c \neq d\}$
  T-Solver$(\Phi(\mathcal{A}))$ = UNSAT, $J = \{g(a) = c, f(g(a)) \neq f(c), c \neq d\}$
- $C = \neg P_1 \vee P_2 \vee P_4$

- $F = P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4 \wedge (\neg P_1 \vee P_2 \vee P_4)$
  SAT-Solver(F) = SAT, $\mathcal{A} = \{P_1 \mapsto 1, P_2 \mapsto 1, P_3 \mapsto 1, P_4 \mapsto 0\}$
- $\Phi(\mathcal{A}) = \{g(a) = c, f(g(a)) = f(c), g(a) = d, c \neq d\}$
  T-Solver$(\Phi(\mathcal{A}))$ = UNSAT, $J = \{g(a) = c, f(g(a)) = f(c), g(a) = d, c \neq d\}$
- $C = \neg P_1 \vee \neg P_2 \vee \neg P_3 \vee P_4$

SMT-Solver($g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$)

- $F =$ prop$(\psi) = P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$
- SAT-Solver(F) = SAT, $\mathcal{A} = \{P_1 \mapsto 1, P_2 \mapsto 0, P_4 \mapsto 0\}$
- $\Phi(\mathcal{A}) = \{g(a) = c, f(g(a)) \neq f(c), c \neq d\}$
  T-Solver$(\Phi(\mathcal{A}))$ = UNSAT, $J = \{g(a) = c, f(g(a)) \neq f(c), c \neq d\}$
- $C = \neg P_1 \vee P_2 \vee P_4$

- $F = P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4 \wedge (\neg P_1 \vee P_2 \vee P_4)$
  SAT-Solver(F) = SAT, $\mathcal{A} = \{P_1 \mapsto 1, P_2 \mapsto 1, P_3 \mapsto 1, P_4 \mapsto 0\}$
- $\Phi(\mathcal{A}) = \{g(a) = c, f(g(a)) = f(c), g(a) = d, c \neq d\}$
  T-Solver$(\Phi(\mathcal{A}))$ = UNSAT, $J = \{g(a) = c, f(g(a)) = f(c), g(a) = d, c \neq d\}$
- $C = \neg P_1 \vee \neg P_2 \vee \neg P_3 \vee P_4$

- $F = P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4 \wedge (\neg P_1 \vee P_2 \vee P_4) \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee P_4)$
  SAT-Solver(F) = **UNSAT**

SMT-Solver($x = 3 \wedge (f(x+y) = f(y) \vee y = 2) \wedge x = y$)

- $F = \text{prop}(\psi) = P_1 \wedge (P_2 \vee P_3) \wedge P_4$
- SAT-Solver(F) = SAT, $\mathcal{A} = \{P_1 \mapsto 1, P_2 \mapsto 0, P_3 \mapsto 1, P_4 \mapsto 1\}$
- $\Phi(\mathcal{A}) = \{x = 3, f(x+y) \neq f(y), y = 2, x = y\}$
  T-Solver$(\Phi(\mathcal{A}))$ = UNSAT, $J = \{x = 3, y = 2, x = y\}$
- $C = \neg P_1 \vee \neg P_3 \vee \neg P_4$

SMT-Solver($x = 3 \wedge (f(x+y) = f(y) \vee y = 2) \wedge x = y$)

- $F = \text{prop}(\psi) = P_1 \wedge (P_2 \vee P_3) \wedge P_4$
- SAT-Solver(F) = SAT, $\mathcal{A} = \{P_1 \mapsto 1, P_2 \mapsto 0, P_3 \mapsto 1, P_4 \mapsto 1\}$
- $\Phi(\mathcal{A}) = \{x = 3, f(x+y) \neq f(y), y = 2, x = y\}$
  T-Solver$(\Phi(\mathcal{A}))$ = UNSAT, $J = \{x = 3, y = 2, x = y\}$
- $C = \neg P_1 \vee \neg P_3 \vee \neg P_4$

- $F = P_1 \wedge (P_2 \vee P_3) \wedge P_4 \wedge (\neg P_1 \vee \neg P_3 \vee \neg P_4)$
  SAT-Solver(F) = SAT, $\mathcal{A} = \{P_1 \mapsto 1, P_2 \mapsto 1, P_3 \mapsto 0, P_4 \mapsto 1\}$
- $\Phi(\mathcal{A}) = \{x = 3, f(x+y) = f(y), y \neq 2, x = y\}$
  T-Solver$(\Phi(\mathcal{A}))$ = **SAT**

The "lazy" approach (enhancements)

Several enhancements are possible to increase efficiency of this basic algorithm:

- If $\Phi(\mathcal{A})$ is $\mathcal{T}$-unsatisfiable, identify a small justification (or unsat core) of it and add its negation as a clause.
- Check $\mathcal{T}$-satisfiability of partial assignment $\mathcal{A}$ as it grows.
- If $\Phi(\mathcal{A})$ is $\mathcal{T}$-unsatisfiable, backtrack to some point where the assignment was still $\mathcal{T}$-satisfiable.

Unsat cores

- Given a $\mathcal{T}$-unsatisfiable set of literals S, a justification (a.k.a. unsat core) for S is any unsatisfiable subset J of S.
- So, the easiest justification S is the set S itself.
- However, conflict clauses obtained this way are too weak.
  - Suppose $\Phi(\mathcal{A}) = \{x = 0, x = 3, l_1, l_2, \ldots, l_{50}\}$. This set is unsat.
  - Theory conflict clause $C = \bigvee_{B \in \Phi(\mathcal{A})} \neg\text{prop}(B)$ prevents that exact same assignment. But it doesn't prevent many other bad assignments involving $x = 0$ and $x = 3$.
  - In fact, there are 2^{50} unsat assignments containing $x = 0$ and $x = 3$, but C just prevents one of them!
- Efficiency can be improved if we have a more precise justification. Ideally, a minimal unsat core. This way we block many assignments using just one theory conflict clause.

Computing minimal unsat core

- How to compute a minimal unsat core of a $\mathcal{T}$-unsatisfiable set of literals S?
- A naive approach:
  - take one literal l of S
  - if $S - \{l\}$ is still UNSAT, $S \leftarrow S - \{l\}$
  - repeat this for every literal in S

Compute a minimal unsat core for

$$S = \{x = y, f(x) + z = 5, f(x) \neq f(y), y \leq 3\}$$

- We can do better...

Computing minimal unsat core

- Instead of dropping one literal at a time, drop half the literals, i.e., do binary search.
- Split S into two sets of similar cardinality S_1 and S_2.
- If S_1 is UNSAT, recursively minimize S_1 and return the result.
- Otherwise, if S_2 is UNSAT, recursively minimize S_2 and return the result.
- If neither S_1 nor S_2 are UNSAT
  - let S_1^* be the result of minimizing S_1 assuming unsat core includes S_2;
  - let S_2^* be the result of minimizing S_2 assuming unsat core includes S_1^*;
  - return $S_1^* \cup S_2^*$.
- How to minimize S_1 assuming unsat core includes S_2?
  - Every time we issue sat query for a subset of S_1, also conjoin S_2 because we assume S_2 is part of unsat core.

Compute a minimal unsat core for $S = \{x = y, f(x) + z = 5, f(x) \neq f(y), y \leq 3\}$

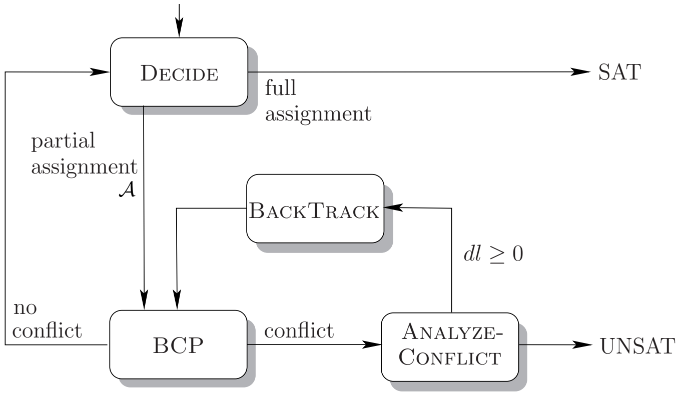
Computing minimal unsat core

- The algorithms just described to compute a minimal unsat core, are using the T-solver as a "blackbox".
  - Independent of the theory; works for any theory.
- Another approach is to augment the T-solver to provide a minimal unsat core.
  - This trategy is potentially much more efficient, because the T-solver can take theory-specific knowledge into account.
  - But not every T-solver provide minimal unsat cores.
- Note that the basic SMT-solver algorithm described above, assumes the T-solver provides an unsat core, but there is no assumption that this core is minimal.

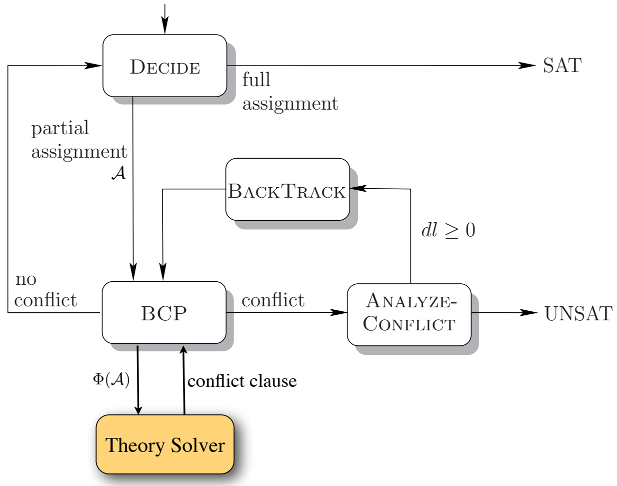
Integration with DPLL

- Lazy SMT solvers are based on the integration of a SAT solver and one (or more) theory solver(s).
- The basic architectural schema described by the SMT-solver algorithm is also called **"lazy offline"** approach, because the SAT solver is re-invoked from scratch each time an assignment is found $\mathcal{T}$-unsatisfiable.
- Some more enhancements are possible if one does not use the SAT solver as a "blackbox".
  - Check $\mathcal{T}$-satisfiability of partial assignment $\mathcal{A}$ as it grows.
  - If $\Phi(\mathcal{A})$ is $\mathcal{T}$-unsatisfiable, backtrack to some point where the assignment was still $\mathcal{T}$-satisfiable.
- To this end we need to integrate the theory solver right into the DPLL algorithm of the SAT solver. This architectural schema is called **"lazy online"** approach.
- Combination of DPLL-based SAT solver and decision procedure for conjunctive $\mathcal{T}$ formula is called *DPLL($\mathcal{T}$) framework*.

DPLL framework for SAT solvers



DPLL($\mathcal{T}$) framework for SMT solvers



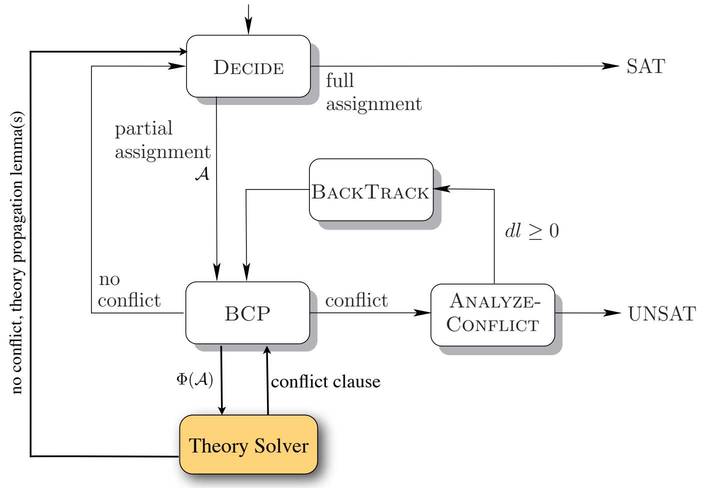
DPLL($\mathcal{T}$) framework

- Suppose SAT solver has made partial assignment $\mathcal{A}$ in Decide step and performed BCP (Boolean Constraints Propagation, i.e. unit propagation).
- If no conflict detected, immediately invoke theory solver.
- Use theory solver to decide if $\Phi(\mathcal{A})$ is $\mathcal{T}$-unsatisfiable.
- If $\Phi(\mathcal{A})$ is $\mathcal{T}$-unsatisfiable, add the negation of its unsat core (the conflict clause) to clause database and continue doing BCP, which will detect conflict.
- As before, Analyze-Conflict decides what level to backtrack to.

DPLL($\mathcal{T}$) framework

- We can go further in the integration of the theory solver into the DPLL algorithm:
  - Theory solver can communicate which literals are implied by current partial assignment.
  - These kinds of clauses implied by theory are called *theory propagation lemmas*.
  - Adding theory propagation lemmas prevents bad assignments to boolean abstraction.

DPLL($\mathcal{T}$) framework



Main benefits of lazy approach

- The theory solver works only with sets of literals.
- Every tool does what it is good at:
  - SAT solver takes care of Boolean information.
  - Theory solver takes care of theory information.
- Modular approach:
  - SAT and theory solvers communicate via a simple API.
  - SMT for a new theory only requires new theory solver.
- Almost all competitive SMT solvers integrate theory solvers use DPLL($\mathcal{T}$) framework.

Solving SMT problems

- The theory solver works only with sets of literals.
- In practice, we need to deal not only with
  - arbitrary Boolean combinations of literals,
  - but also with formulas with quantifiers
- Some more sophisticated SMT solvers are able to handle formulas involving quantifiers. But usually one loses decidability...

Choosing a SMT solver

- Theres are many available SMT solvers:
  - some are targeted to specific theories;
  - many support SMT-LIB format;
  - many provide non-standard features.
- Features to have into account:
  - the efficiency of the solver for the targeted theories;
  - the solver's license;
  - the ways to interface with the solver;
  - the "support" (is it being actively developed?).
- See `http://smtcomp.sourceforge.net`