Automated Reasoning 2018

Lecture 7: Satisfiability modulo theory (SMT) solvers

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2018-08-07

CDCL($\mathcal{T}$)

CDCL solves(i.e. checks satisfiability) quantifier-free propositional formulas

CDCL($\mathcal{T}$) solves quantifier-free formulas in theory $\mathcal{T}$,

- separates the boolean and theory reasoning,
- proceeds like CDCL, and
- needs support of a $\mathcal{T}$-solver $DP_{\mathcal{T}}$, i.e., a decision procedure for conjunction of literals of $\mathcal{T}$

The tools that are build using CDCL($\mathcal{T}$) are called satisfiablity modulo theory solvers (SMT solvers)

CDCL($\mathcal{T}$) - some notation

Let $\mathcal{T}$ be a first-order-logic theory with signature $\mathbf{S}$.

We assume input formulas are from $\mathcal{T}$, quantifier-free, and in CNF.

Definition 7.1

For a quantifier-free $\mathcal{T}$ formula F, let $atoms(F)$ denote the set of atoms appearing in F.

Example 7.1

- $f(x) \approx g(h(x,y))$ *is a formula in QF_EUF.*
- $x > 0 \vee y + x \approx 3.5z$ *is a formula in QF_LRA.*

Free variables vs. constants

If we have a quantifier-free formula and are interested in satisfiability. Then, we may assume that all variables in the formula are existentially quantified.

If we apply skolemization on such a formula then we obtain a formula with fresh constants and no variables.

> In quantifier-free satisfiability checking, variables and constants play the same role.

Boolean encoder

For a formula F, let boolean encoder e be a partial map from $atoms(F)$ to fresh boolean variables.

Definition 7.2

For a formula F, let $e(F)$ denote the term obtained by replacing each atom a by $e(a)$ if $e(a)$ is defined.

Example 7.2

Let $F = x < 2 \vee (y > 0 \vee x \geq 2)$
and $e = \{x < 2 \mapsto x_1, y > 0 \mapsto x_2\}$
$e(F) = x_1 \vee (x_2 \vee \neg x_1)$

Partial model

Definition 7.3

For a boolean encoder e, a partial model m is an ordered partial map from range(e) to $\mathcal{B}$.

Example 7.3

partial models $\{x \mapsto 0, y \mapsto 1\}$ *and* $\{y \mapsto 1, x \mapsto 0\}$ *are not same.*

Definition 7.4

For a partial model m of e, let

$e^{-1}(m) \triangleq \{e^{-1}(x)|x \mapsto 1 \in m\} \cup \{\neg e^{-1}(x)|x \mapsto 0 \in m\}$

Example 7.4

Let $e = \{x < 2 \mapsto x_1, y > 0 \mapsto x_2\}$ *and* $m = \{x_1 \mapsto 0, x_2 \mapsto 1\}$.

$e^{-1} = \{x_1 \mapsto x < 2, x_2 \mapsto y > 0\}$

$e^{-1}(m) = \{\neg(x < 2), y > 0\}$

CDCL($\mathcal{T}$)

Algorithm 7.1: CDCL($\mathcal{T}$)(formula F')

```
e := CreateEncoder(F') ;
F := e(F'); m := UnitPropagation(m, F); dl := 0; dstack := λx.0;
do
    // backtracking
    while m ⊭ F do
        if dl = 0 then return unsat;
        (C, dl) := AnalyzeConflict(m) ;                    // clause learning
        m.resize(dstack(dl)); F := F ∪ {C}; m := UnitPropagation(m, F);
    // Boolean decision
    if m is partial then
        dstack(dl) := m.size();
        dl := dl + 1; m := Decide(m, F); m := UnitPropagation(m, F) ;
    // Theory propagation
    if m ⊨ F then
        (Cs, dl') := TheoryDeduction(T)(⋀ e^{-1}(m), m, dstack, dl);
        if dl' < dl then {dl = dl'; m.resize(dstack(dl)); } ;
        F := F ∪ e(Cs); m := UnitPropagation(m, F);
while m is partial or m ⊭ F;
return sat
```

stands for decision level

dstack records history for backtracking

returns a clause set and a decision level

Theory propagation

THEORYDEDUCTION looks at the atoms assigned so far and checks

- if they are mutually unsatisfiable
- if not, are there other literals from F that are implied by the current assignment

Any implementation must comply with the following goals

- Correctness: boolean model is consistent with $\mathcal{T}$
- Termination: unsat partial models are never repeated

TheoryDeduction

TheoryDeduction solves conjunction of literals and returns a set of clauses and a decision level.

$$(Cs, dl') := \textsc{TheoryDeduction}(\mathcal{T})(\bigwedge e^{-1}(m), m, dstack, dl)$$

Cs may contain the clauses of the form

$$(\bigwedge L) \Rightarrow \ell$$

where $\ell \in lits(F') \cup \{\bot\}$ and $L \subseteq e^{-1}(m)$.

Example 7.5

If $\textsc{TheoryDeduction}(QF\_LRA)(x > 1 \wedge x < 0, ...)$ *is called, the returned clauses will be*

$$Cs := \{(x > 1 \wedge x < 0 \Rightarrow \bot)\}.$$

If $\textsc{TheoryDeduction}(QF\_LRA)(x > 1 \wedge y > 0, ...)$ *is called, the returned clauses will be*

$$Cs := \{(x > 1 \wedge y > 0 \Rightarrow x + y > 0), ...\}.$$

Assuming $x + y > 0$ occurs in input

Commentary: The RHS need not be a single literal. However, in most theories the single literal is a good practical choice.

Requirement form TheoryDeduction

The output of TheoryDeduction must satisfy the following conditions

- If $\bigwedge e^{-1}(m)$ is unsat in $\mathcal{T}$ then Cs must contain a clause with $\ell = \bot$.
- if $\bigwedge e^{-1}(m)$ is sat then $dl' = dl$.
  Otherwise, dl' is the decision level immediately after which the unsatisfiablity occurred (clearly stated shortly).

Example : CDCL(QF_EUF)

Example 7.6

Consider $F' = (x \approx y \lor y \approx z) \land (y \not\approx z \lor z \approx u) \land (z \approx x)$

$e(F') = (x_1 \lor x_2) \land (\neg x_2 \lor x_3) \land x_4$

After $F := e(F'); m := \textsc{UnitPropagation}(m, F)$

$m = \{x_4 \mapsto 1\}$

After $m := \textsc{Decide}(m, F);$

$m = \{x_4 \mapsto 1, x_2 \mapsto 0\}$

After $m := \textsc{UnitPropagation}(m, F)$

$m = \{x_4 \mapsto 1, x_2 \mapsto 0, x_1 \mapsto 1\}$

Example : CDCL(QF_EUF) II

After $(Cs, dl') :=$ $\textsc{TheoryDeduction}(\text{QF\_EUF})(x \approx y \land y \not\approx z \land z \approx x, ..)$
$Cs = \{x \not\approx y \lor y \approx z \lor z \not\approx x\}, dl' = 0, e(Cs) = \{\neg x_1 \lor x_2 \lor \neg x_4\}$

After $F := F \cup e(Cs)); m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1, x_2 \mapsto 0, x_1 \mapsto 1\}$ $\leftarrow$ conflict with learned clause

Exercise 7.1
Complete the run

Topic 7.1

Implementation of TheoryDeduction

Theory propagation implementation - incremental solver

Theory propagation is implemented using incremental theory solvers.

Incremental solver $DP_{\mathcal{T}}$ for theory $\mathcal{T}$

- takes input constraints as a sequence of literals,
- maintains a data structure that defines the solver state and satisfiability of constraints seen so far.

Theory solver $DP_{\mathcal{T}}$ interface

A theory solver must provide the following interface.

- push(ℓ) - adds literal ℓ in "constraint store"
- pop() - removes last pushed literal from the store
- checkSat() - checks satisfiability of current store
- unsatCore() - returns the set of literals that caused unsatisfiablity

Commentary: We assume that push and pop call checkSat() at the end of their execution. Therefore, explicit calls to checkSat() are not necessary. However, practical tools allow users to choose the policy of calling checkSat() - lazy vs. eager

Theory propagation implementation

Algorithm 7.2: THEORYDEDUCTION

Input: Set of literals Ls

Read only input: m partial model, $dstack$ decision depths, dl current decision level

foreach $\ell \in Ls$ **do**
  $DP_{\mathcal{T}}.push(\ell)$

if $DP_{\mathcal{T}}.checkSat() == unsat$ **then**
  // theory conflict
  $Ls' := DP_{\mathcal{T}}.unsatCore()$; // minimize clause
  $dl' := \max\{dl'' | \exists \ell \in Ls', i.\ m[i] = e(\ell) \wedge dstack(dl'') < i\}$;
  return $(\{\neg \bigwedge Ls'\}, dl')$

else
  //implied clauses
  $Cs := \emptyset$;
  foreach $\ell \in Lits(F')$ **do**
    $DP_{\mathcal{T}}.push(\neg\ell)$;
    if $DP_{\mathcal{T}}.checkSat() == unsat$ **then**
      $Ls' := DP_{\mathcal{T}}.unsatCore()$; // ℓ is called implied literal and $\neg\ell \in Ls'$
      $Cs := Cs \cup \{\neg \bigwedge Ls'\}$;
    $DP_{\mathcal{T}}.pop()$;
  return (Cs, dl)

$Ls' = Ls$ will also be correct. But, inefficient.

dl' is the latest decision after which all literals in Ls' became true.

Topic 7.2

Example theory propagation implementation

Theory of Equality and function symbols (EUF)

EUF syntax: quantifier-free first order formulas with signature $\mathbf{S} = (\mathbf{F}, \emptyset)$, i.e., countably many function symbols and no predicates.

The theory axioms include

1. $\forall x.\ x \approx x$
2. $\forall x, y.\ x \approx y \Rightarrow y \approx x$
3. $\forall x, y, z.\ x \approx y \land y \approx z \Rightarrow x \approx z$
4. for each $f/n \in \mathbf{F}$,

$$\forall x_1, .., x_n, y_1, .., y_n.\ x_1 \approx y_1 \land .. \land x_n \approx y_n \Rightarrow f(x_1, .., x_n) \approx f(y_1, .., y_n)$$

Since the axioms are valid in FOL with equality, the theory is sometimes referred as the base theory.

Note: Predicates can be easily added if desired

DP_{EUF}

Decides conjunction of literals with interface

push, pop, checkSat, and unsatCore.

$DP_{EUF}.push$

General idea: maintain equivalence classes among terms

Algorithm 7.3: $DP_{EUF}.push(t_1 \bowtie t_2)$

globals: set of terms $Ts := \emptyset$, set of pairs of classes $DisEq := \emptyset$, bool $conflictFound := 0$

$Ts := Ts \cup subTerms(t_1) \cup subTerms(t_2)$;

$C_1 := getClass(t_1)$; $C_2 := getClass(t_2)$; // if t_i is seen first time, create new class

if $\bowtie$ = "$\approx$" **then**

- **if** $C_1 = C_2$ **then** **return** ;
- **if** $(C_1, C_2) \in DisEq$ **then** { $conflictFound := 1$; **return**; } ;
- $C := mergeClasses(C_1, C_2)$; $parent(C) := (C_1, C_2, t_1 \approx t_2)$;
- $DisEq := DisEq[C_1 \mapsto C, C_2 \mapsto C]$

else

- // $\bowtie$ = "$\not\approx$"
- $DisEq := DisEq \cup (C_1, C_2)$;
- **if** $C_1 = C_2$ **then** $conflictFound := 1$; **return** ;

foreach $f(r_1, \ldots, r_n), f(s_1, \ldots, s_n) \in Ts \wedge \forall i \in 1..n.\ \exists C.\ r_i, s_i \in C$ **do**

- $DP_{EUF}.push(f(r_1, \ldots, r_n) \approx f(s_1, \ldots, s_n))$;

Example: push

Example 7.7

Consider input $f(f(x)) \not\approx x \wedge f(x) \approx x$

- $DP_{EUF}.push(f(f(x)) \not\approx x)$
  - *term set* $Ts = \{x, f(x), f(f(x))\}$
  - *classes* $C_1 = \{f(f(x))\}$, *and* $C_2 = \{x\}$
  - $DisEq = \{(C_1, C_2)\}$

- $DP_{EUF}.push(f(x) \approx x)$
  - *classes* $C_1 = \{f(f(x))\}$, $C_2 = \{x\}$, *and* $C_3 = \{f(x)\}$
  - $C_4 = mergeClasses(C_2, C_3)$: *classes* $C_1 = \{f(f(x))\}$, $C_4 = \{f(x), x\}$
  - $DisEq = \{(C_1, C_4)\}$
  - *Apply congruence on function* f *and terms of* C_4
    - *Triggers recursive call* $DP_{EUF}.push(f(f(x)) \approx f(x))$

- $DP_{EUF}.push(f(f(x)) \approx f(x))$
  - *Since* $(C_1, C_4) \in DisEq$, $conflictFound = 1$ *and exit*

checkSat and pop

- $DP_{EUF}.checkSat()$ { **return** *conflictFound*; }
- $DP_{EUF}.pop()$ is implemented by recording the time stamp of pushes and undoing all the mergers happened in after last push.

Exercise 7.2

Write pseudo code for $DP_{EUF}.pop()$

Unsat core

Definition 7.5

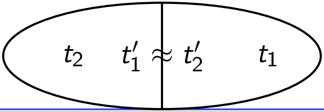
An unsat core of Σ is a subset (preferably minimal) of Σ that is unsat.

Algorithm 7.4: $DP_{EUF}.unsatCore()$

assume($conflictFound = 1$);
Let $(t_1 \not\approx t_2)$ be the disequality that was violated;
return $\{t_1 \not\approx t_2\} \cup getReason(t_1, t_2)$;

Algorithm 7.5: $getReason(t_1, t_2)$

Let $(t'_1 \approx t'_2)$ be the merge operation that placed t_1 and t_2 in same class;
if $t'_1 = f(s_1, .., s_k) \approx f(u_1, ...u_k) = t'_2$ *was derived due to congruence* **then**
  $reason := \bigcup_i getReason(s_i, u_i)$
else
  $reason := \{t'_1 \approx t'_2\}$
return $getReason(t_1, t'_1) \cup reason \cup getReason(t'_2, t_2)$

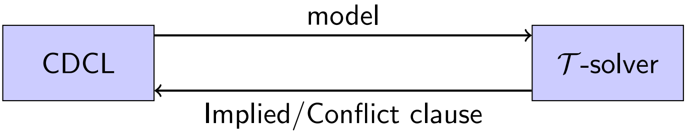


Topic 7.3

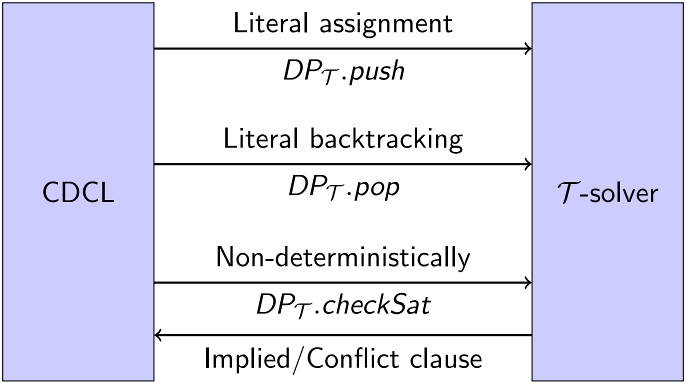
Optimizations

Incremantal theory propagation

- Earlier $CDCL(\mathcal{T})$



- Fine-grained interaction with theory



Theory propagation strategies

- Exhaustive or Eager :
  Cs contains all possible clauses
- Minimal or Lazy :
  Cs only contains the clause that refutes current m
- Somewhat Lazy :
  Cs contains only easy to deduce clauses

Implied literals without implied clauses

Bottleneck: There may be too many implied clauses.

Observation: Very few of the implied clauses are useful, i.e., contribute in early detection of conflict.

Optimization: apply implied literals, without adding implied clauses.

Optimization overhead: If an implied model is used in conflict then recompute the implied clause for the implication graph analysis.

Relevancy

Bottleneck: All the assigned literals are sent to the theory solver.

Observation: However, *CDCL* only needs to send those literals to the solver that make unique clauses satisfiable.

Optimization:

- Each clause chooses one literal that makes it sat under current model.
- Those clause that are not sat under current model do nothing.
- If a literal is not chosen by any clause then it is not passed on to $\mathcal{T}$-solver.

Patented: US8140459 by Z3 guys(the original idea is more general than stated here)

Optimization overhead: Relevant literal management

Exercise 7.3

Suggest a scheme for relevant literal management.

Effect of optimizations

Only experiments can tell if these are good ideas!

Topic 7.4

SMT Solvers

Rise of SMT solvers

- In early 2000s, stable SMT solvers started appearing. e.g., Yiecs
- SMT competition(SMT-comp) became a driving force in their ever increasing efficiency
- Formal methods community quickly realized their potential
- Z3, one of the leading SMT solver, alone has about 3000+ citations (375 per year)(June 2016)

Leading tools

The following are some of the leading SMT solvers

- Z3
- CVC4
- MathSAT
- Boolector

Topic 7.5

Problems

Run SMT solvers

Exercise 7.4

- *Find a satisfying assignment of the following formula using SMT solver*

$$(x > 0 \lor y < 0) \land (x + y > 0 \lor x - y < 0)$$

  Give the model generated by the SMT solver.

- *Prove the following formula is valid using SMT solver*

$$(x > y \land y > z) \Rightarrow x > z$$

  Give the proof generated by the SMT solver.

Please do not simply submit the output. Please write the answers in the mathematical notation.

Knapsack problem

Exercise 7.5

Write a program for solving the knapsack problem that requires filling a knapsack with stuff with maximum value. For more information look at the following.

`https://en.wikipedia.org/wiki/Knapsack_problem`

The output of the program should be the number of solutions that have value more than 95% of the best value.

Download Z3 from the following webpage:
https://github.com/Z3Prover/z3

We need a tool to feed random inputs to your tool. Write a tool that generates random instances, similar to what was provided last time.

Evaluate the performance on reasonably sized problems. You also need to design the evaluation strategy. Evaluation plots and a small text to describe

End of Lecture 7