Automated Reasoning 2020

Lecture 10: Satisfiability modulo theory (SMT) solvers

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2020-09-10

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 **S**.

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

Definition 10.1

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

Example 10.1

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

Boolean encoder

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

Definition 10.2

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 10.2

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 10.3

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

Example 10.3

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

CDCL($\mathcal{T}$) will proceed by constructing partial models like CDCL.

Reverse encoder

Definition 10.4

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 10.4

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

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

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

Theory propagation

If we have partial assignment m, then we need to check if the theory accepts the assignment.

In other words, we need to know if $\bigwedge e^{-1}(m)$ is sat.

Example 10.5

In last example, we had $e^{-1}(m) = \{\neg(x < 2), y > 0\}$.
We ask if $\bigwedge e^{-1}(m) = \neg(x < 2) \land y > 0$ *is sat. If no, we need to backtrack the assignments.*

We assume that function $\textsc{TheoryDeduction}$ can check satisfiability of $\bigwedge e^{-1}(m)$.

CDCL($\mathcal{T}$)

Algorithm 10.1: CDCL($\mathcal{T}$)(formula G)

$e := \textsc{CreateEncoder}(G)$; $F := e(G)$; $m := \textsc{UnitPropagation}(m, F)$; $dl := 0$; $dstack := \lambda x.0$;

F is Boolean encoding of input G

do

// backtracking

while $m \not\models F$ **do**

  if $dl = 0$ **then return** *unsat*;

  $(C, dl) := \textsc{AnalyzeConflict}(m)$;

  $m.resize(dstack(dl))$; $F := F \cup \{C\}$; m $:= \textsc{UnitPropagation}(m, F)$;

Same as SAT solver CDCL

// Boolean decision

if *F is unassigned under m* **then**

  $dstack(dl) := m.size()$; $dl := dl + 1$; m $:= \textsc{Decide}(m, F)$; m $:= \textsc{UnitPropagation}(m, F)$;

// Theory propagation

if *F is unassigned or sat under m* **then**

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

  if $dl' < dl$ **then** $\{dl = dl'; m.resize(dstack(dl)); \}$;

  $F := F \cup e(Cs)$; m $:= \textsc{UnitPropagation}(m, F)$;

returns a clause set and a decision level

while *F is unassigned under m or* $m \not\models F$ *or* $e^{-1}(m)$ *is unsat*;

return *sat*

Topic 10.1

TheoryDeduction

Theory propagation

THEORYDEDUCTION looks at the atoms assigned so far and checks

- if they are mutually unsatisfiable
- if not, are there other literals from G 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 $L \subseteq e^{-1}(m)$.

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

Example : TheoryDeduction

Example 10.6

If TheoryDeduction(*QF_LRA*)($x > 1 \land x < 0, ...$) *is called, the returned clauses will be*

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

If TheoryDeduction(*QF_LRA*)($x > 1 \land y > 0, ...$) *is called, the returned clauses may be*

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

Assuming $x + y > 0$ occurs in input

Specification of 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 $\ell = \bot$. dl' is the decision level immediately after which the unsatisfiablity occurred (clearly stated shortly).
- if $\bigwedge e^{-1}(m)$ is sat then $dl' = dl$.

Example : CDCL(QF_EUF)

Example 10.7

Consider $F' = (x = y \vee y = z) \wedge (y \neq z \vee z = u) \wedge (z = x)$
$e(F') = (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge x_4$

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 $(Cs, dl') :=$ $\textsc{TheoryDeduction}$(QF_EUF)$(x = y \wedge y \neq z \wedge z = x, ..)$
$Cs = \{x \neq y \vee y = z \vee z \neq x\}, dl' = 0, e(Cs) = \{\neg x_1 \vee x_2 \vee \neg x_4\}$

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 10.1

Complete the run

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

Definition 10.5

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

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 10.2: THEORYDEDUCTION

Input: Set of literals Ls

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

```
foreach ℓ ∈ Ls do
    DP_T.push(ℓ)
if DP_T.checkSat() == unsat then
    // theory conflict
    Ls' := DP_T.unsatCore(); dl' := max{dl'' | ∃ℓ ∈ Ls', i. m[i] = e(ℓ) ∧ dstack(dl'') < i};
    return ({¬⋀Ls'}, dl')
else
    //implied clauses
    Cs := ∅;
    foreach ℓ ∈ Lits(G) do
        DP_T.push(¬ℓ);
        if DP_T.checkSat() == unsat then
            Ls' := DP_T.unsatCore(); Cs := Cs ∪ {¬⋀Ls'};
        DP_T.pop();
    return (Cs,dl)
```

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

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

ℓ is called implied literal and $\neg\ell \in Ls'$

Topic 10.2

Example theory propagation implementation

DP_{EUF}

Decides conjunction of literals with interface

push, pop, checkSat, and unsatCore.

push, checkSat, and pop

- $DP_{EUF}.push$

Algorithm 10.3: $DP_{EUF}.push(t_1 \bowtie t_2)$

```
IncrEUF(t1 ⋈ t2);
```

- $DP_{EUF}.checkSat()$ { **return** $conflictFound$; }

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

Exercise 10.2

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

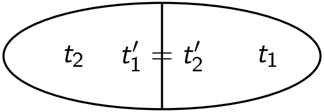
Unsat core

Algorithm 10.4: $DP_{EUF}.unsatCore()$

assume($conflictFound = 1$);
Let $(t_1 \neq t_2)$ be the disequality that was violated;
return $\{t_1 \neq t_2\} \cup getReason(t_1, t_2)$;

Algorithm 10.5: $getReason(t_1, t_2)$

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

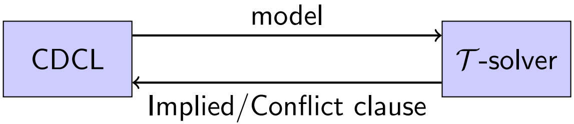


Topic 10.3

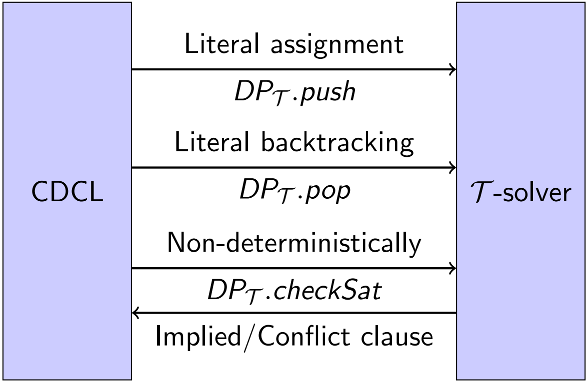
Optimizations

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 10.3

Suggest a scheme for relevant literal management.

Topic 10.4

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 10.5

Problems

Run SMT solvers

Exercise 10.4

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

$$(x > 0 \vee y < 0) \wedge (x + y > 0 \vee x - y < 0)$$

  Give the model generated by the SMT solver.

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

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

  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 10.5

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 your strategies.

End of Lecture 10