CS257: Introduction to Automated Reasoning

DPLL(T): Combining T-Solvers with SAT

Stanford University

CENTAUR

Theory of Uninterpreted Functions: $\mathcal{T}_{EUF}$

Given a signature Σ with equalities, the most unrestricted theory would include the class of all Σ-models.

This family of theories parameterized by the signature, is known as the theory of **Equality with Uninterpreted Functions (EUF)** or the **empty theory**, since it imposes no restrictions on its models.

QF_UF (conjunctions of $\mathcal{T}_{EUF}$-literals) can be decided with **congruence closure** procedure.

Example: $(f(a) = a) \wedge (g(a) \neq f(a))$

Note: For simplicity, assume we only consider equality over 1 sort.

Congruence Closure: Definitions

Consider a set S and a binary relation R.

R is an **equivalence relation** iff it is reflexive, symmetric, and transitive.

An equivalence relation R is a **congruence relation** iff for every n-ary function f,
$\forall x_1, \ldots x_n. \forall y_1, \ldots, y_n.((\bigwedge_{i=1}^{n} R(x_i, y_i)) \rightarrow R(f(x_1, \ldots, x_n), f(y_1, \ldots, y_n)))$.

Is $=$ an congruence relation?

Congruence Closure: Definitions

Given a relation R, its **equivalence closure** R^E is the smallest relation that

- contains R;
- is a equivalent relation.

Given a relation R, its **congruence closure** R^C is the smallest relation that

- contains R;
- is a congruence relation.

Congruence Closure: Algorithm

Given a Σ-formula α, define its **subterm set** S_α as the set that contains precisely the subterms of α that do not contain equality symbols.

Example: $\alpha := f(f(a)) = a \wedge f(f(f(a))) = a \wedge g(a) \neq g(f(a))$

$S_\alpha := \{a, f(a), f(f(a)), f(f(f(a))), g(a), g(f(a))\}$

High-level idea:

1. Partition the literals into a set of equalities E and a set of inequalities D
2. Construct the congruence closure of E over S_α
3. Unsatisfiable iff there exists $t_1 \neq t_2 \in D$ and $(t_1, t_2) \in E^C$

Congruence Closure: Algorithm

$$\alpha := f(f(a)) = a \land f(f(f(a))) = a \land g(a) \neq g(f(a))$$

$$S_\alpha := \{a, f(a), f(f(a)), f(f(f(a))), g(a), g(f(a))\}$$

Step 1: place each subterm of α into its own congruence class:

$$\{a\},\ \{f(a)\},\ \{f(f(a))\},\ \{f(f(f(a)))\},\ \{g(a)\},\ \{g(f(a))\}$$

Congruence Closure: Algorithm

Step 2: For each positive literal $t_1 = t_2$ in α

- **merge** the congruence classes for t_1 and t_2
- **propagate** the resulting congruences

$$\alpha := f(f(a)) = a \land f(f(f(a))) = a \land g(a) \neq g(f(a))$$

$$\{a, f(a), f(f(a)), f(f(f(a)))\},\ \{g(a), g(f(a))\}$$

Congruence Closure: Algorithm

$$\alpha := f(f(a)) = a \land f(f(f(a))) = a \land g(a) \neq g(f(a))$$

$$\{a, f(a), f(f(a)), f(f(f(a)))\}, \{g(a), g(f(a))\}$$

Step 3: α is $\mathcal{T}_{EUF}$-unsatisfiable, iff α has a negative literal $t_1 \neq t_2$, where t_1 and t_2 are in the same congruence class.

Note: This Algorithm can be implemented efficiently with a union-find data structure (CC. Chap. 9.1-9.3).

Congruence Closure: still an active research problem

Downey, et al. “Variations on the common subexpressions problem”, 1980.

Nieuwenhuis and Oliveras, “Proof-Producing Congruence Closure”, 2005.

Willsey, et al. “egg: Fast and extensible equality saturation”, 2021.

What if we have disjunctions?

The congruence closure checks the satisfiability of conjunctions of $\mathcal{T}_{EUF}$-literals.

What about

$$g(a) = c \quad \wedge \quad (f(g(a)) \neq f(c) \vee g(a) = d) \quad \wedge \quad c \neq d$$

Theorem: The T-satisfiability of quantifier-free formulas is decidable iff the T-satisfiability of conjunctions/sets of literals is decidable.

Convert the formula to DNF and check if any of its disjuncts is T-satisfiable. Very inefficient!

A better solution: exploit propositional satisfiability technology

Lifting SAT Technology to SMT

Two main approaches:

1. "Eager"
   - translate into an equisatisfiable propositional formula
   - feed it to any SAT solver
2. "Lazy"
   - abstract the input formula to a propositional one
   - feed it to a (CDCL-based) SAT solver
   - use a theory decision procedure to refine the formula and guide the SAT solver
   - Notable systems: **Bitwuzla**, **cvc5**, **MathSAT**, **Yices**, **Z3**

Lazy Approach for SMT

Given a quantifier-free Σ-formula ϕ, for each atomic formula α in ϕ, we associate a unique propositional variable $e(\alpha)$.

The **Boolean skeleton** of a formula ϕ is a propositional logic formula, where each atomic formula α in ϕ is replaced with $e(\alpha)$.

Example:

$$\phi := (x < 0 \vee (x + y < 1 \wedge \neg(x < 0))) \rightarrow y < 0$$

Let $e(x < 0) = p_1$, $e(x + y < 1) = p_2$, $e(y < 0) = p_3$

What is the Boolean skeleton of ϕ?

(Very) Lazy Approach for SMT – Example

$$g(a) = c \quad \wedge \quad f(g(a)) \neq f(c) \vee g(a) = d \quad \wedge \quad c \neq d$$

Simplest setting:

- Off-line SAT solver
- Non-incremental **theory solver** for conjunctions of equalities and disequalities
- Theory atoms (e.g., $g(a) = c$) abstracted to propositional atoms (e.g., 1)

(Very) Lazy Approach for SMT – Example

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\neg 2} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\neg 4}$$

- Send $\{1,\ \neg 2 \vee 3,\ \neg 4\}$ to SAT solver.
- SAT solver returns model $\{1,\ \neg 2,\ \neg 4\}$.
  Theory solver finds (concretization of) $\{1,\ \neg 2,\ \neg 4\}$ unsat.
- Send $\{1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4\}$ to SAT solver.
- SAT solver returns model $\{1,\ 3,\ \neg 4\}$.
  Theory solver finds $\{1,\ 3,\ \neg 4\}$ unsat.
- Send $\{1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2,\ \neg 1 \vee \neg 3 \vee 4\}$ to SAT solver.
- SAT solver finds $\{1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4,\ \neg 1 \vee \neg 3 \vee 4\}$ unsat.
  Done: the original formula is unsatisfiable in T_{EUF}.

Eager Approach for SMT – Example

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

Step 1: eliminate all function applications (Ackermann's encoding)

- introduce a constant symbol f_x to replace function application $f(x)$;
- for each pair of introduced variables f_x, f_y, add the formula $x = y \rightarrow f_x = f_y$.

$$f(b) \Rightarrow f_b \qquad f(a) \Rightarrow f_a$$

$$(f_b = a \vee f_a \neq a) \wedge (a = b \rightarrow f_a = f_b)$$

Now, atomic formulas are equalities between constants/variables

Eager Approach for SMT – Example

Rename f_b as c and f_a as d:

$$(f_b = a \lor f_a \neq a) \land (a = b \rightarrow f_a = f_b)$$

becomes

$$(c = a \lor d \neq a) \land (a = b \rightarrow d = c)$$

Step 2: eliminate all equalities.

- replace each pair of constants x, y with a unique propositional variable $p_{x,y}$
- add facts about reflexivity, symmetry, transitivity

$$\begin{gathered}(p_{c,a} \lor \neg p_{d,a}) \land (p_{a,b} \rightarrow p_{d,c}) \\ \land\, p_{a,a} \land p_{b,b} \land p_{c,c} \land p_{d,d} \\ \land\, (p_{a,b} \leftrightarrow p_{b,a}) \land (p_{a,c} \leftrightarrow p_{c,a}) \land (p_{a,d} \leftrightarrow p_{d,a}) \ldots \\ \land\, ((p_{a,b} \land p_{b,c}) \rightarrow p_{a,c}) \land ((p_{a,c} \land p_{c,d}) \rightarrow p_{a,d}) \ldots\end{gathered}$$

The resulting propositional formula is equi-satisfiable with the original T_{EUF}-formula.

Note: not all the transitivity cases are needed.

Discussion

"Eager"

- translate into an equisatisfiable propositional formula
- feed it to any SAT solver

"Lazy"

- abstract the input formula to a propositional one
- feed it to a (CDCL-based) SAT solver
- use a theory decision procedure to refine the formula and guide the SAT solver

What are the pros and cons of the eager approach and the lazy approach?

Submit your answers to

`https://pollev.com/andreww095`

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- Check T-satisfiability only of full propositional model ~~Check T-satisfiability only of full propositional model~~
- Check T-satisfiability of partial assignment M as it grows
- If M is T-unsatisfiable, add $\neg M$ as a clause ~~If M is T-unsatisfiable, add $\neg M$ as a clause~~
- If M is T-unsatisfiable, identify a T-unsatisfiable subset M_0 of M and add $\neg M_0$ as a clause
- If M is T-unsatisfiable, add clause and restart ~~If M is T-unsatisfiable, add clause and restart~~
- If M is T-unsatisfiable, backtrack to some point where the assignment was still T-satisfiable

Lazy Approach – Main Benefits

- Every tool does what it is good at:
  - SAT solver takes care of Boolean information
  - Theory solver takes care of theory information
- The theory solver works only with conjunctions of literals
- Modular approach:
  - SAT and theory solvers communicate via a simple API
  - SMT for a new theory only requires new theory solver
  - An off-the-shelf SAT solver can be embedded in a lazy SMT system with a few new lines of code

An Abstract Framework for Lazy SMT

Several variants and enhancements of lazy SMT solvers exist

They can be modeled with an abstract calculus.

Review: CDCL

States: $\texttt{Fail}$ or $\langle M, \Delta, C \rangle$

Initial state:

- $\langle (), \Delta_0, \texttt{no} \rangle$, where Δ_0 is to be checked for satisfiability

Expected final states:

- $\texttt{Fail}$ if Δ_0 is **unsatisfiable**
- $\langle M, G, \texttt{no} \rangle$ otherwise, where
  - G is **equivalent** to Δ_0 and
  - M **satisfies** G

Review: CDCL Rules

$$\frac{l \in \texttt{Lits}(\Delta) \qquad l, \neg l \notin M}{M := M \bullet l}\ (\textbf{Decide}) \qquad \frac{\Delta \models C \qquad C \notin \Delta}{\Delta := \Delta \cup \{C\}}\ (\textbf{Learn})$$

$$\frac{C \neq \texttt{no} \qquad \bullet \notin M}{\texttt{Fail}}\ (\textbf{Fail}) \qquad \frac{C = \texttt{no} \qquad \Delta = \Delta' \cup \{C\} \qquad \Delta' \models C}{\Delta := \Delta'}\ (\textbf{Forget})$$

$$\frac{\{l_1, \cdots, l_n, l\} \in \Delta \qquad \neg l_1, \cdots, \neg l_n \in M \qquad l, \neg l \notin M}{M := M\ l}\ (\textbf{Propagate})$$

$$\frac{}{M := M^{[0]} \qquad C := \texttt{no}}\ (\textbf{Restart})$$

Conflict, **Explain**

$$\frac{C = \{l_1, \cdots, l_n, l\} \qquad \texttt{lev}(\neg l_1), \ldots, \texttt{lev}(\neg l_n) \leq\ i < \texttt{lev}(\neg l)}{C := \texttt{no} \quad M := M^{[i]}\, l}\ (\textbf{Backjump})$$

We are going to extend this abstract framework to lazy SMT

From SAT to SMT

Same states and transitions but

- Δ contains quantifier-free clauses in some theory T
- CDCL Rules operates on the Boolean skeleton of Δ (assume a mapping from theory literal to propositional literal
- M is a sequence of theory literals (i.e., atomic formulas or their negations) and decision points
- the CDCL system is augmented with rules

  T-**Conflict**, T-**Propagate**

SMT-level Rules

Fix a theory T

At SAT level:

$$\frac{C = \mathtt{no} \qquad \{l_1, \cdots l_n\} \in \Delta \qquad \neg l_1, \ldots, \neg l_n \in M}{C := \{l_1, \cdots, l_n\}} \text{ (**Conflict**)}$$

At SMT level:

$$\frac{\mathrm{C} = \mathtt{no} \quad l_1 \wedge \ldots \wedge l_n \vDash_T \bot \quad l_1, \ldots, l_n \in \mathrm{M}}{\mathrm{C} := \{\neg l_1, \cdots, \neg l_n\}} \text{ (}T\text{-**Conflict**)}$$

If the conjunction of a set of literals in the trail are unsatisfiable modulo T, the negation of the set of literals consistutes a conflict clause.

SMT-level Rules

At SAT level:

$$\frac{\{l_1, \cdots, l_n, l\} \in \Delta \quad \neg l_1, \cdots, \neg l_n \in M \quad l, \neg l \notin M}{M := M\ l}\ (\textbf{Propagate})$$

At SMT level:

$$\frac{l \in \texttt{Lits}(\Delta) \quad \mathrm{M} \models_T l \quad l, \neg l \notin \mathrm{M}}{\mathrm{M} := \mathrm{M}\ l}\ (T\textbf{-Propagate})$$

If the current partial assignment logically entails some literal l in T, extend the trail with l.

SMT-level Rules

At SAT level:

$$\frac{C = \{l\} \cup D \quad \{l_1, \cdots, l_n, \neg l\} \in \Delta \quad \neg l_1, \ldots \neg l_n, \neg l \in M \quad \neg l_1, \ldots, \neg l_n \prec_M \neg l}{C := \{l_1, \cdots, l_n\} \cup D} \; (\textbf{Explain})$$

At SMT level:

$$\frac{\mathrm{C} = \{l\} \cup D \quad \neg l_1 \wedge \ldots \wedge \neg l_n \models_T \neg l \quad \neg l_1, \ldots, \neg l_n \prec_{\mathrm{M}} \neg l}{\mathrm{C} := \{l_1, \cdots, l_n\} \cup D} \; (T\textbf{-Explain})$$

There is a literal l in the conflict clause, and $\neg l$ is logically entailed by some literals assigned before it. We can derive a new conflict clause by performing a resolution.

Modeling the Very Lazy Theory Approach

T-**Conflict** is enough to model the naive integration of SAT solvers and theory solvers seen in the earlier EUF example

$$\underbrace{g(a) = c}_{1} \;\wedge\; \underbrace{f(g(a)) \neq f(c)}_{\neg 2} \vee \underbrace{g(a) = d}_{3} \;\wedge\; \underbrace{c \neq d}_{\neg 4}$$

M	Δ	C	rule
	$1,\ \neg 2 \vee 3,\ \neg 4$	no	
$1\ \neg 4$	$1,\ \neg 2 \vee 3,\ \neg 4$	no	by **Propagate**$^+$
$1\ \neg 4 \bullet \neg 2$	$1,\ \neg 2 \vee 3,\ \neg 4$	no	by **Decide**
$1\ \neg 4 \bullet \neg 2$	$1,\ \neg 2 \vee 3,\ \neg 4$	$\neg 1 \vee 2 \vee 4$	by T-**Conflict**
$1\ \neg 4 \bullet \neg 2$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4$	$\neg 1 \vee 2 \vee 4$	by **Learn**
$1\ \neg 4$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4$	no	by **Restart**
$1\ \neg 4\ 2\ 3$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4$	no	by **Propagate**$^+$
$1\ \neg 4\ 2\ 3$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4,\ \neg 1 \vee \neg 3 \vee 4$	$\neg 1 \vee \neg 3 \vee 4$	by T-**Conflict**, **Learn**
Fail			by **Fail**

A Better Lazy Approach

The very lazy approach can be improved considerably with

- An **on-line** SAT engine,
  which can accept new input clauses on the fly

- an **incremental and explicating** T-solver,
  which can:
  1. check the T-satisfiability of M as it is extended and
  2. identify a small T-unsatisfiable subset of M once M becomes T-unsatisfiable

A Better Lazy Approach

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\neg 2} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\neg 4}$$

M	Δ	C	rule
	$1,\ \neg 2 \vee 3,\ \neg 4$	no	
$1\ \neg 4$	$1,\ \neg 2 \vee 3,\ \neg 4$	no	by **Propagate**$^+$
$1\ \neg 4 \bullet \neg 2$	$1,\ \neg 2 \vee 3,\ \neg 4$	no	by **Decide**
$1\ \neg 4 \bullet \neg 2$	$1,\ \neg 2 \vee 3,\ \neg 4$	$\neg 1 \vee 2$	by ***T*-Conflict**
$1\ \neg 4\ 2$	$1,\ \neg 2 \vee 3,\ \neg 4$	no	by **Backjump**
$1\ \neg 4\ 2\ 3$	$1,\ \neg 2 \vee 3,\ \neg 4$	no	by **Propagate**
$1\ \neg 4\ 2\ 3$	$1,\ \neg 2 \vee 3,\ \neg 4$	$\neg 1 \vee \neg 3 \vee 4$	by ***T*-Conflict**
Fail			by **Fail**

Lazy Approach – Strategies

Ignoring **Restart** (for simplicity), a common strategy is to apply the rules using the following priorities:

1. If a clause is falsified by the current assignment M, apply **Conflict**
2. If M is T-unsatisfiable, apply **T-Conflict**
3. Apply **Fail** or **Explain**+**Learn**+**Backjump** as appropriate
4. Apply **Propagate**
5. Apply **Decide**

Note: Depending on the cost of checking the T-satisfiability of M, Step (2) can be applied with lower frequency or priority

Theory Propagation

With T-**Conflict** as the only theory rule, the theory solver is used just to validate the choices of the SAT engine

With T-**Propagate** and T-**Explain**, it can also be used to guide the engine's search

$$\frac{l \in \mathtt{Lits}(\Delta) \quad \mathrm{M} \models_T l \quad l, \neg l \notin \mathrm{M}}{\mathrm{M} := \mathrm{M}\ l}\ (T\text{-}\textbf{Propagate})$$

$$\frac{\mathrm{C} = \{l\} \cup D \quad \neg l_1 \wedge \ldots \wedge \neg l_n \models_T \neg l \quad \neg l_1, \ldots, \neg l_n \prec_{\mathrm{M}} \neg l}{\mathrm{C} := \{l_1, \cdots, l_n\} \cup D}\ (T\text{-}\textbf{Explain})$$

Theory Propagation Example

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\neg 2} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\neg 4}$$

M	Δ	C	rule
	$1,\ \neg 2 \vee 3,\ \neg 4$	no	
$1\ \neg 4$	$1,\ \neg 2 \vee 3,\ \neg 4$	no	by **Propagate**$^{+}$
$1\ \neg 4\ 2$	$1,\ \neg 2 \vee 3,\ \neg 4$	no	by T-**Propagate** $(1 \models_T 2)$
$1\ \neg 4\ 2\ \neg 3$	$1,\ \neg 2 \vee 3,\ \neg 4$	no	by T-**Propagate** $(1,\ \neg 4 \models_T \neg 3)$
$1\ \neg 4\ 2\ \neg 3$	$1,\ \neg 2 \vee 3,\ \neg 4$	$\neg 2 \vee 3$	by **Conflict**
Fail			by **Fail**

Note: T-propagation eliminates search altogether in this case
no applications of **Decide** are needed

Modeling Modern Lazy SMT Solvers

At the core, current lazy SMT solvers are implementations of the transition system with rules:

(1) **Propagate**, **Decide**, **Conflict**, **Explain**, **Backjump**, **Fail**

(2) T**-Conflict**, T**-Propagate**, T**-Explain**

(3) **Learn**, **Forget**, **Restart**

Basic DPLL Modulo Theories $\overset{\text{def}}{=}$ (1) + (2)

DPLL Modulo Theories $\overset{\text{def}}{=}$ (1) + (2) + (3)

Correctness

Updated terminology:

Irreducible state: state to which no Basic DPLL modulo Theories rules apply

Execution: sequence of transitions allowed by the rules and starting with $M = \emptyset$ and $C = \texttt{no}$

Exhausted execution: execution ending in an irreducible state

Proposition: (Termination) Every execution in which

(a) **Learn**/**Forget** are applied only finitely many times and

(b) **Restart** is applied with increased periodicity

is finite.

Lemma: Every exhausted execution ends with either $C = \texttt{no}$ or $\texttt{Fail}$.

Proposition (Soundness) For every exhausted execution starting with $\Delta = \Delta_0$ and ending with $\texttt{Fail}$, the clause set Δ_0 is T-unsatisfiable.

DPLL(T) Architecture

The approach formalized so far can be implemented with a simple architecture named DPLL(T)

$$\text{DPLL}(T) = \text{DPLL}(X) \text{ engine} + T\text{-solver}$$

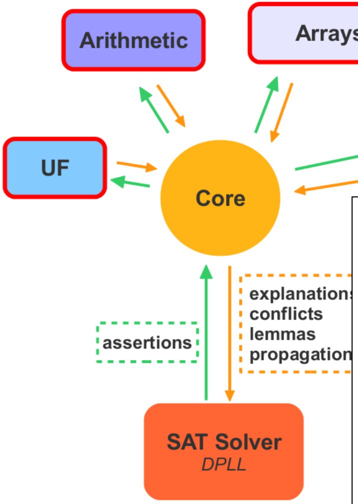
DPLL(X):

- Very similar to a SAT solver, enumerates Boolean models
- Not allowed: pure literal
- Required: incremental addition of clauses
- Desirable: partial model detection

T-solver:

- Checks the T-satisfiability of conjunctions of literals
- Computes theory propagations
- Produces explanations of T-unsatisfiability/propagation
- Must be incremental and backtrackable

SMT Solvers



Theory Solvers

- Decide T-satisfiability of a conjunction of theory literals
- Incremental
- Backtrackable
- Conflict Generation
- Theory Propagation