Combination and Augmentation Methods for Satisfiability Modulo Theories

Cesare Tinelli

`tinelli@cs.uiowa.edu`

The University of Iowa

Credits

- ***Slides inspired by previous work and presentations by:***

  Silvio Ghilardi, Sava Krstic, Albert Oliveras, Harald Ruess, Roberto Sebastiani, Natarajan Shankar, Ashish Tiwari, Calogero Zarba, and others.

- ***Special thanks to:***
  - Clark Barrett and Albert Oliveras (for contributing some of the material) and
  - Ed Clarke (for the invitation).

Propositional Satisfiability: SAT

- Deciding the satisfiability of a propositional formula is a well-studied and important problem.
- Theoretical interest: first established NP-Complete problem, phase transition, . . .
- Practical interest: applications to scheduling, planning, logic synthesis, verification, . . .
  - Development of algorithms and enhancements.
  - Implementation of extremely efficient tools.
  - Solvers based on the DPLL procedure have been the most successful so far.

Satisfiability Modulo Theories

- Any SAT solver can be used to decide the satisfiability of ground (i.e., variable-free) first-order formulas.

Satisfiability Modulo Theories

- Any SAT solver can be used to decide the satisfiability of ground (i.e., variable-free) first-order formulas.
- Often, however, one is interested in the satisfiability of certain ground formulas in a theory:

Satisfiability Modulo Theories

- Any SAT solver can be used to decide the satisfiability of ground (i.e., variable-free) first-order formulas.
- Often, however, one is interested in the satisfiability of certain ground formulas in a theory:
  - Pipelined microprocessors: theory of equality, atoms like $f(g(a, b), c) = g(c, a)$.

Satisfiability Modulo Theories

- Any SAT solver can be used to decide the satisfiability of ground (i.e., variable-free) first-order formulas.
- Often, however, one is interested in the satisfiability of certain ground formulas in a theory:
  - Pipelined microprocessors: theory of equality, atoms like $f(g(a,b),c) = g(c,a)$.
  - Timed automata, planning: theory of integers/reals, atoms like $x - y < 2$.

Satisfiability Modulo Theories

- Any SAT solver can be used to decide the satisfiability of ground (i.e., variable-free) first-order formulas.
- Often, however, one is interested in the satisfiability of certain ground formulas in a theory:
  - Pipelined microprocessors: theory of equality, atoms like $f(g(a,b),c) = g(c,a)$.
  - Timed automata, planning: theory of integers/reals, atoms like $x - y < 2$.
  - Software verification/model checking: combination of theories, atoms like $5 + car(a+2) = cdr(a[j]+1)$.

Satisfiability Modulo Theories

- Any SAT solver can be used to decide the satisfiability of ground (i.e., variable-free) first-order formulas.
- Often, however, one is interested in the satisfiability of certain ground formulas in a theory:
  - Pipelined microprocessors: theory of equality, atoms like $f(g(a,b),c) = g(c,a)$.
  - Timed automata, planning: theory of integers/reals, atoms like $x - y < 2$.
  - Software verification/model checking: combination of theories, atoms like $5 + car(a+2) = cdr(a[j]+1)$.
- We refer to this general problem as (ground) Satisfiability Modulo Theories, or SMT.

Satisfiability Modulo Theories

Given a logical theory T and a formula φ, the SMT problem consists of deciding whether there exists a model $\mathcal{A}$ of T that satisfies φ.

Satisfiability Modulo Theories

Given a logical theory T and a formula φ, the SMT problem consists of deciding whether there exists a model $\mathcal{A}$ of T that satisfies φ.

Some theories of interest in SMT

- Equality with "Uninterpreted Function Symbols"
- Arithmetic (Real and Integer)
- Arrays
- Bit-vectors
- Sets
- Inductive Datatypes

Solving SMT Problems

Fact: Many theories of interest have (efficient) decision procedures for sets of literals.

Solving SMT Problems

Fact: Many theories of interest have (efficient) decision procedures for sets of literals.

Problem: In practice, we need to deal with

1. arbitrary Boolean combinations of literals, and
2. literals over more than one theory.

Solving SMT Problems

Fact: Many theories of interest have (efficient) decision procedures for sets of literals.

Problem: In practice, we need to deal with

1. arbitrary Boolean combinations of literals, and
2. literals over more than one theory.

This talk concerns general methods to address Point 1 and 2.

Structure of this Talk

Part I:

From sets of ground literals to arbitrary ground formulas.

Part II:

From a single theory T to multiple theories $T_1, \ldots, T_n$.

Part I

From sets of ground literals to arbitrary ground formulas

Satisfiability Modulo a Theory T

- Note: The T-satisfiability of ground formulas is decidable iff the T-satisfiability of sets of literals is decidable.

- Problem: In practice, dealing with Boolean combinations of literals is as hard as in the propositional case.

- Current solution: Exploit propositional satisfiability technology.

Lifting SAT Technology to SMT

Lifting SAT Technology to SMT

- Eager approach [CBMC, UCLID, . . .]:
  - translate into an equisatisfiable propositional formula,
  - feed it to any SAT solver.

Lifting SAT Technology to SMT

- Eager approach [CBMC, UCLID, . . .]:
  - translate into an equisatisfiable propositional formula,
  - feed it to any SAT solver.
- Lazy approach [Barcelogic, CVC*, ICS, MathSAT, Verifun, Yices, Zap, . . .]:
  - abstract the input formula into a propositional one,
  - feed it to a DPLL-based SAT solver,
  - use a theory decision procedure to refine the formula,
  - use the decision procedure to guide the search of DPLL solver.

Lifting SAT Technology to SMT

- Eager approach [CBMC, UCLID, . . .]:
  - translate into an equisatisfiable propositional formula,
  - feed it to any SAT solver.

- Lazy approach [Barcelogic, CVC*, ICS, MathSAT, Verifun, Yices, Zap, . . .]:
  - abstract the input formula into a propositional one,
  - feed it to a DPLL-based SAT solver,
  - use a theory decision procedure to refine the formula,
  - use the decision procedure to guide the search of DPLL solver.

- This talk will focus on the lazy approach.

The Original DPLL Procedure [DLL62]

- Tries to build incrementally a satisfying truth assignment M for a CNF formula F.
- M is grown by
  - deducing the truth value of a literal from M and F, or
  - guessing a truth value.
- If a wrong guess for a literal leads to an inconsistency, the procedure backtracks and tries the opposite value.

An Abstract Framework for DPLL [NOT06]

- Many variants and enhancements of the DPLL procedure exist.
- We can model DPLL and its enhancements abstractly and declaratively as transition systems.
- A transition system is a binary relation over states, induced by a set of conditional transition rules.

Advantages of Abstract Framework

An abstract framework helps:

- Skip over implementation details and unimportant control aspects.
- Reason formally about DPLL-based solvers for SAT and for SMT.
- Model modern features such as non-chronological bactracking, lemma learning or restarts.
- Describe different strategies and prove their correctness.

An Abstract Framework for DPLL

Our states:

$$fail \quad \text{or} \quad M \parallel F$$

where F is a CNF formula, a set of clauses,
and
M is a sequence of annotated literals
denoting a partial truth assignment.

An Abstract Framework for DPLL

Our states:

$$fail \quad \text{or} \quad M \parallel F$$

Initial state:

- $\emptyset \parallel F$, where F is to be checked for satisfiability.

Expected final states:

- $fail$, if F is unsatisfiable
- $M \parallel G$, where G is logically equivalent to F and M satisfies G, otherwise.

Transition Rules for the Original DPLL

Extending the assignment:

Propagate

$$M \parallel F, C \vee l \quad \rightarrow \quad M\; l \parallel F, C \vee l \quad \textbf{if} \begin{cases} M \models_p \neg C, \\ l \text{ is undefined in } M \end{cases}$$

Notation: $\models_p$ is propositional entailment

Transition Rules for the Original DPLL

Extending the assignment:

Propagate

$$M \parallel F, C \vee l \quad \to \quad M\, l \parallel F, C \vee l \quad \textbf{if} \begin{cases} M \models_p \neg C, \\ l \text{ is undefined in } M \end{cases}$$

Notation: $\models_p$ is propositional entailment

Decide

$$M \parallel F \quad \to \quad M\, l^{\bullet} \parallel F \quad \textbf{if} \begin{cases} l \text{ or } \bar{l} \text{ occurs in } F, \\ l \text{ is undefined in } M \end{cases}$$

Notation: $l^{\bullet}$ annotates l as a decision literal

Transition Rules for the Original DPLL

Repairing the assignment:

Fail

$$M \parallel F, C \quad \longrightarrow \quad \mathit{fail} \quad \textbf{if} \begin{cases} M \models_p \neg C, \\ M \text{ contains no decision literals} \end{cases}$$

Transition Rules for the Original DPLL

Repairing the assignment:

Fail

$$M \parallel F, C \quad \longrightarrow \quad \mathit{fail} \quad \textbf{if} \begin{cases} M \models_p \neg C, \\ M \text{ contains no decision literals} \end{cases}$$

Backtrack

$$M\, l^{\bullet}\, N \parallel F, C \quad \longrightarrow \quad M\, \bar{l} \parallel F, C \quad \textbf{if} \begin{cases} M\, l^{\bullet}\, N \models_p \neg C, \\ l \text{ last decision literal} \end{cases}$$

From Backtracking to Backjumping

Backtrack

$$M\, l^{\bullet}\, N \parallel F, C \quad \longrightarrow \quad M\, \bar{l} \parallel F, C \quad \textbf{if} \quad \begin{cases} M\, l^{\bullet}\, N \models_p \neg C, \\ l \text{ last decision literal} \end{cases}$$

From Backtracking to Backjumping

Backtrack

$$M\, l^{\bullet}\, N \parallel F, C \quad \longrightarrow \quad M\, \bar{l} \parallel F, C \quad \textbf{if} \begin{cases} M\, l^{\bullet}\, N \models_p \neg C, \\ l \text{ last decision literal} \end{cases}$$

Backjump

$$M\, l^{\bullet}\, N \parallel F, C \quad \longrightarrow \quad M\, k \parallel F, C \quad \textbf{if} \begin{cases} 1.\ M\, l^{\bullet}\, N \models_p \neg C, \\ 2.\ \text{for some clause } D \vee k : \\ \quad F, C \models_p D \vee k, \\ \quad M \models_p \neg D, \\ \quad k \text{ is undefined in } M, \\ \quad k \text{ or } \overline{k} \text{ occurs in} \\ \quad M\, l^{\bullet}\, N \parallel F, C \end{cases}$$

From Backtracking to Backjumping

Backtrack

$$M\, l^\bullet\, N \parallel F, C \quad \longrightarrow \quad M\, \bar{l} \parallel F, C \quad \textbf{if} \begin{cases} M\, l^\bullet\, N \models_p \neg C, \\ l \text{ last decision literal} \end{cases}$$

Backjump

$$M\, l^\bullet\, N \parallel F, C \quad \longrightarrow \quad M\, k \parallel F, C \quad \textbf{if} \begin{cases} 1.\ M\, l^\bullet\, N \models_p \neg C, \\ 2.\ \text{for some clause } D \vee k: \\ \quad F, C \models_p D \vee k, \\ \quad M \models_p \neg D, \\ \quad k \text{ is undefined in } M, \\ \quad k \text{ or } \overline{k} \text{ occurs in} \\ \quad M\, l^\bullet\, N \parallel F, C \end{cases}$$

Note: If (1) holds, clauses like $D \vee k$ are computable from C.

Basic DPLL System

At the core, current DPLL-based SAT solvers are implementations of the transition system:

Basic DPLL

- Propagate
- Decide
- Fail
- Backjump

Enhancements to Basic DPLL

Enhancements to Basic DPLL

Learn

$$M \parallel F \quad \longrightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } F, \\ F \models_p C \end{cases}$$

Enhancements to Basic DPLL

Learn

$$M \parallel F \quad\longrightarrow\quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } F, \\ F \models_p C \end{cases}$$

Forget

$$M \parallel F, C \quad\longrightarrow\quad M \parallel F \quad \textbf{if}\ F \models_p C$$

Enhancements to Basic DPLL

Learn

$$M \parallel F \quad \longrightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } F, \\ F \models_p C \end{cases}$$

Forget

$$M \parallel F, C \quad \longrightarrow \quad M \parallel F \quad \textbf{if } F \models_p C$$

Usually, C is a clause identified during conflict analysis.

Enhancements to Basic DPLL

Learn

$$M \parallel F \quad \longrightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } F, \\ F \models_p C \end{cases}$$

Forget

$$M \parallel F, C \quad \longrightarrow \quad M \parallel F \quad \textbf{if } F \models_p C$$

Restart

$$M \parallel F \quad \longrightarrow \quad \emptyset \parallel F \quad \textbf{if} \ldots \textit{you want to}$$

Enhancements to Basic DPLL

Learn

$$M \parallel F \quad \longrightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } F, \\ F \models_p C \end{cases}$$

Forget

$$M \parallel F, C \quad \longrightarrow \quad M \parallel F \quad \textbf{if } F \models_p C$$

Restart

$$M \parallel F \quad \longrightarrow \quad \emptyset \parallel F \quad \textbf{if } \ldots \textit{you want to}$$

The DPLL system =

$\{$Propagate, Decide, Fail, Backjump, Learn, Forget, Restart$\}$

The DPLL System – Correctness

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.

The DPLL System – Correctness

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.

Proposition (Soundness) For every execution $\emptyset \parallel F \Longrightarrow \cdots \Longrightarrow M \parallel G$ with $M \parallel G$ irreducible wrt. Basic DPLL, $M \models_p F$.

The DPLL System – Correctness

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.

Proposition (Soundness) For every execution $\emptyset \parallel F \Longrightarrow \cdots \Longrightarrow M \parallel G$ with $M \parallel G$ irreducible wrt. Basic DPLL, $M \models_p F$.

Proposition (Completeness) If F is unsatisfiable, for every execution $\emptyset \parallel F \Longrightarrow \cdots \Longrightarrow S$ with S irreducible wrt. Basic DPLL, $S = \mathit{fail}$.

(For simplicity the statements above are not entirely accurate. See [NOT06] for details.)

(Very) Lazy Approach for SMT – Example

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

Theory T: EUF

(Very) Lazy Approach for SMT – Example

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

Simplest setting:

- Off-line SAT solver
- Non-incremental T-solver

(Very) Lazy Approach for SMT – Example

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

- Send $\{1,\, \overline{2} \vee 3,\, \overline{4}\}$ to SAT solver.

(Very) Lazy Approach for SMT – Example

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

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.
- SAT solver returns model $\{1,\ \overline{2},\ \overline{4}\}$.
  Theory solver finds $\{1,\ \overline{2}\}$ T-unsatisfiable.

(Very) Lazy Approach for SMT – Example

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

- Send $\{1, \overline{2} \vee 3, \overline{4}\}$ to SAT solver.
- SAT solver returns model $\{1, \overline{2}, \overline{4}\}$.
  Theory solver finds $\{1, \overline{2}\}$ T-unsatisfiable.
- Send $\{1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2\}$ to SAT solver.

(Very) Lazy Approach for SMT – Example

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

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.
- SAT solver returns model $\{1,\ \overline{2},\ \overline{4}\}$.
  Theory solver finds $\{1,\ \overline{2}\}$ T-unsatisfiable.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2\}$ to SAT solver.
- SAT solver returns model $\{1,\ 2,\ 3,\ \overline{4}\}$.
  Theory solver finds $\{1,\ 3,\ \overline{4}\}$ T-unsatisfiable.

(Very) Lazy Approach for SMT – Example

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

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.
- SAT solver returns model $\{1,\ \overline{2},\ \overline{4}\}$.
  Theory solver finds $\{1,\ \overline{2}\}$ T-unsatisfiable.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2\}$ to SAT solver.
- SAT solver returns model $\{1,\ 2,\ 3,\ \overline{4}\}$.
  Theory solver finds $\{1,\ 3,\ \overline{4}\}$ T-unsatisfiable.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4\}$ to SAT solver.

(Very) Lazy Approach for SMT – Example

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

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.
- SAT solver returns model $\{1,\ \overline{2},\ \overline{4}\}$.
  Theory solver finds $\{1,\ \overline{2}\}$ T-unsatisfiable.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2\}$ to SAT solver.
- SAT solver returns model $\{1,\ 2,\ 3,\ \overline{4}\}$.
  Theory solver finds $\{1,\ 3,\ \overline{4}\}$ T-unsatisfiable.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4\}$ to SAT solver.
- SAT solver finds $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4\}$ unsatisfiable.

Modeling the Lazy Approach

This naive combination can be greatly improved with an on-line DPLL engine and an incremental T-solver.

Modeling the Lazy Approach

This naive combination can be greatly improved with an on-line DPLL engine and an incremental T-solver.

Both the naive and the more sophisticated integrations can be modeled in Abstract DPLL with the following rules:

- Propagate, Decide, Fail, Restart

  (as in the propositional case) and

- T-Backjump, T-Learn, T-Forget

Modeling the Lazy Approach

This naive combination can be greatly improved with an on-line DPLL engine and an incremental T-solver.

Both the naive and the more sophisticated integrations can be modeled in Abstract DPLL with the following rules:

- Propagate, Decide, Fail, Restart

  (as in the propositional case) and

- T-Backjump, T-Learn, T-Forget

Note: The first component of a state $M \parallel F$ is still a truth assignment, but now for ground, first-order literals.

Modeling the Lazy Approach

T-Backjump

$$M\, l^{\bullet}\, N \parallel F, C \quad \longrightarrow \quad M\, k \parallel F, C \quad \textbf{if} \begin{cases} 1.\ M\, l^{\bullet}\, N \models_p \neg C, \\ 2.\ \text{for some clause } D \vee k\text{:} \\ \quad F, C \models_T D \vee k, \\ \quad M \models_p \neg D, \\ \quad k \text{ is undefined in } M, \\ \quad k \text{ or } \overline{k} \text{ occurs in} \\ \quad M\, l^{\bullet}\, N \parallel F, C \end{cases}$$

Only change: $\models_T$ instead of $\models_p$

Not.: $F \models_T G$ iff every model of T that satisfies F satisfies G.

Modeling the Lazy Approach

T-Backjump

$$M\, l^\bullet\, N \parallel F, C \quad \longrightarrow \quad M\, k \parallel F, C \quad \textbf{if} \begin{cases} 1.\ M\, l^\bullet\, N \models_p \neg C, \\ 2.\ \text{for some clause } D \vee k\text{:} \\ \quad F, C \models_T D \vee k, \\ \quad M \models_p \neg D, \\ \quad k \text{ is undefined in } M, \\ \quad k \text{ or } \overline{k} \text{ occurs in} \\ \quad M\, l^\bullet\, N \parallel F, C \end{cases}$$

T-Learn

$$M \parallel F \quad \longrightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } M \parallel F, \\ F \models_T C \end{cases}$$

T-Forget

$$M \parallel F, C \quad \longrightarrow \quad M \parallel F \quad \textbf{if } F \models_T C$$

Modeling the Lazy Approach

The naive interaction between SAT solver and theory solver in the previous example can be modeled with the following

Refinement of T-Learn

$$M \parallel F \quad \longrightarrow \quad M \parallel F, \overline{l_1} \vee \ldots \vee \overline{l_n} \quad \textbf{if} \begin{cases} l_1 \cdots l_n \subseteq M \\ l_1 \wedge \cdots \wedge l_n \vdash_T \perp \end{cases}$$

with Restart applied right after each application of this T-Learn.

Modeling the Lazy Approach

The naive interaction between SAT solver and theory solver in the previous example can be modeled with the following

Refinement of T-Learn

$$M \parallel F \;\; \longrightarrow \;\; M \parallel F, \overline{l_1} \vee \ldots \vee \overline{l_n} \quad \textbf{if} \begin{cases} l_1 \cdots l_n \subseteq M \\ l_1 \wedge \cdots \wedge l_n \vdash_T \bot \end{cases}$$

with Restart applied right after each application of this T-Learn.

However, note that

- The learned blocking clause is false in M, hence either Backjump or Fail applies.
- T-Learn can be applied as early as possible, i.e., with $M = N\, l_n$.

(Very) Lazy Theory Approach - Example

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

(Very) Lazy Theory Approach - Example

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

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow^{*} \quad (\text{Propagate}) \\
1\ \overline{4} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow \quad (\text{Decide}) \\
1\ \overline{4}\ \ \overline{2}^{\bullet} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow \quad (T\text{-Learn}) \\
1\ \overline{4}\ \ \overline{2}^{\bullet} & \| & 1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 & \Longrightarrow \quad (\text{Restart}) \\
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 & \Longrightarrow^{*} \quad (\text{Propagate}) \\
1\ \overline{4}\ 2\ 3 & \| & 1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 & \Longrightarrow \quad (T\text{-Learn}) \\
1\ \overline{4}\ 2\ 3 & \| & 1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 & \Longrightarrow \quad (\text{Restart}) \\
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 & \Longrightarrow^{*} \quad (\text{Propagate}) \\
1\ \overline{4}\ 2\ 3 & \| & 1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 & \Longrightarrow \quad (\text{Fail}) \\
 & fail & &
\end{array}$$

Lazy Theory Approach - Example

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

More advanced setting:

- On-line SAT solver
- Incremental T-solver

Lazy Theory Approach - Example

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

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow^{*} & \text{(Propagate)} \\
1\ \overline{4} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & \text{(Decide)} \\
1\ \overline{4}\ \ \overline{2}^{\bullet} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (T\text{-Learn}) \\
1\ \overline{4}\ \ \overline{2}^{\bullet} & \| & 1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 & \Longrightarrow & (\text{Backjump with } \overline{1} \vee 2) \\
1\ \overline{4}\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 & \Longrightarrow & \text{(Propagate)} \\
1\ \overline{4}\ 2\ 3 & \| & 1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 & \Longrightarrow & (T\text{-Learn}) \\
1\ \overline{4}\ 2\ 3 & \| & 1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 & \Longrightarrow & \text{(Fail)} \\
 & & fail & &
\end{array}$$

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 as appropriate Fail or (Backjump + T-Learn of backjump clause).
2. If M is T-unsatisfiable, apply T-Learn of blocking clause and go to 1.
3. Apply Propagate.
4. Apply Decide.

Theory Propagation

With the previous rules, the T-solver is used just to validate the choices of the DPLL engine.

Theory Propagation

With the previous rules, the T-solver is used just to validate the choices of the DPLL engine.

With the new rule below, it can also be used to direct the engine's search.

T-Propagate

$$M \parallel F \quad \rightarrow \quad M\ l \parallel F \quad \textbf{if} \begin{cases} M \models_T l \\ l \text{ or } \bar{l} \text{ occurs in } F \\ l \text{ is undefined in } M \end{cases}$$

Theory Propagation - Example

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

$$\emptyset \;\; \| \quad 1,\ \overline{2} \vee 3,\ \overline{4}$$

Theory Propagation - Example

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

$$\begin{array}{lll} \emptyset \parallel \quad 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{Propagate}) \\ 1 \parallel \quad 1,\ \overline{2} \vee 3,\ \overline{4} & & \end{array}$$

Theory Propagation - Example

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

$$\begin{array}{rcl}
\emptyset \;\| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow \quad (\text{Propagate}) \\
1 \;\| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow \quad (T\text{-Propagate}) \\
1\ 2 \;\| & 1,\ \overline{2} \vee 3,\ \overline{4} &
\end{array}$$

Theory Propagation - Example

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

$$\begin{array}{rcllcl}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{Propagate}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (T\text{-Propagate}) \\
1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{Propagate}) \\
1\ 2\ 3 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & &
\end{array}$$

Theory Propagation - Example

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

$$\begin{array}{rclcl}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{Propagate}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (T\text{-Propagate}) \\
1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{Propagate}) \\
1\ 2\ 3 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (T\text{-Propagate}) \\
1\ 2\ 3\ 4 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & &
\end{array}$$

Theory Propagation - Example

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

$$\begin{array}{rcllcl}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{Propagate}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (T\text{-Propagate}) \\
1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{Propagate}) \\
1\ 2\ 3 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (T\text{-Propagate}) \\
1\ 2\ 3\ 4 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{Fail}) \\
 & fail & & & &
\end{array}$$

Theory Propagation

- With exhaustive theory propagation every assignment M is T-satisfiable (since $M\ l$ is T-unsatisfiable iff $M \models_T \bar{l}$).
- For some theories, e.g., difference logic, this approach is extremely effective.
- For some others, e.g., the theory of equality, it is too expensive to detect all T-consequences.
- If T-Propagate is not applied exhaustively, T-Learn is needed to repair T-unsatisfiable assignments.

From Complete to Incomplete Theory Solvers

- Abstract DPLL Modulo Theories is based on the availability of a T-solver for determining T-entailment ($\models_T$).
- At the very least, the T-solver must be refutationally sound:

  never calling a T-satisfiable set M of literals T-unsatisfiable,

- Ideally, it should also be refutationally complete:

  always able to recognize a T-unsatisfiable set M of literals as such.

- For certain theories, it is advantageous to relax the refutational completeness requirement.

Case Splitting

- For certain theories, determining that M is T-unsatisfiable requires reasoning by cases.

Case Splitting

- For certain theories, determining that M is T-unsatisfiable requires reasoning by cases.

Example: T = the theory of arrays.

$$M = \{\, \underbrace{r(w(a,i,x),j) \neq x}_{1},\ \underbrace{r(w(a,i,x),j) \neq r(a,j)}_{2} \,\}$$

Case Splitting

- For certain theories, determining that M is T-unsatisfiable requires reasoning by cases.

Example: T = the theory of arrays.

$$M = \{\underbrace{r(w(a,i,x),j) \neq x}_{1},\ \underbrace{r(w(a,i,x),j) \neq r(a,j)}_{2}\}$$

$i = j$) Then, $r(w(a,i,x),j) = x$. Contradiction with 1.

Case Splitting

- For certain theories, determining that M is T-unsatisfiable requires reasoning by cases.

Example: T = the theory of arrays.

$$M = \{\underbrace{r(w(a,i,x),j) \neq x}_{1},\ \underbrace{r(w(a,i,x),j) \neq r(a,j)}_{2}\}$$

$i = j$) Then, $r(w(a,i,x),j) = x$. Contradiction with 1.

$i \neq j$) Then, $r(w(a,i,x),j) = r(a,j)$. Contradiction with 2.

Case Splitting

- For certain theories, determining that M is T-unsatisfiable requires reasoning by cases.

Example: T = the theory of arrays.

$$M = \{\underbrace{r(w(a,i,x),j) \neq x}_{1},\ \underbrace{r(w(a,i,x),j) \neq r(a,j)}_{2}\}$$

$i = j$) Then, $r(w(a,i,x),j) = x$. Contradiction with 1.

$i \neq j$) Then, $r(w(a,i,x),j) = r(a,j)$. Contradiction with 2.

Conclusion: M is T-unsatisfiable.

Case Splitting

- For certain theories, determining that M is T-unsatisfiable requires reasoning by cases.

Case Splitting

- For certain theories, determining that M is T-unsatisfiable requires reasoning by cases.

- A complete T-solver does that with (internal) case splitting and backtracking mechanisms.

Case Splitting

- For certain theories, determining that M is T-unsatisfiable requires reasoning by cases.
- A complete T-solver does that with (internal) case splitting and backtracking mechanisms.
- An alternative approach is to lift case splitting and backtracking from the T-solver to the DPLL engine.

Case Splitting

- For certain theories, determining that M is T-unsatisfiable requires reasoning by cases.
- A complete T-solver does that with (internal) case splitting and backtracking mechanisms.
- An alternative approach is to lift case splitting and backtracking from the T-solver to the DPLL engine.
- Basic idea: Code the case split as a set of clauses and send them as needed to the engine so it can split on them.

Splitting on Demand [BNOT06]

Basic idea: Code the case split as a set of clauses and send them as needed to the engine so it can split on them.

Possible benefits:

- All case-splitting is coordinated by the DPLL engine
- Only have to implement case-splitting infrastructure in one place
- DPLL heuristics are not sabotaged by internal theory splitting

Splitting on Demand [BNOT06]

Basic idea: Code the case split as a set of clauses and send them as needed to the engine so it can split on them.

Splitting on Demand [BNOT06]

Basic idea: Code the case split as a set of clauses and send them as needed to the engine so it can split on them.

Basic Scenario:

$$M = \{\ldots,\ s = \underbrace{r(w(a,i,t),j)}_{s'},\ \ldots,\}$$

Splitting on Demand [BNOT06]

Basic idea: Code the case split as a set of clauses and send them as needed to the engine so it can split on them.

Basic Scenario:

$$M = \{\ldots,\ s = \underbrace{r(w(a,i,t),j)}_{s'},\ \ldots,\}$$

DPLL Engine: “Is M T-unsatisfiable?”

Splitting on Demand [BNOT06]

Basic idea: Code the case split as a set of clauses and send them as needed to the engine so it can split on them.

Basic Scenario:

$$M = \{\ldots,\ s = \underbrace{r(w(a,i,t),j)}_{s'},\ \ldots,\}$$

DPLL Engine: "Is M T-unsatisfiable?"

T**-solver:** "I do not know yet, but it will help me if you split on these theory lemmas:

$$s = s' \wedge i = j \rightarrow s = t, \quad s = s' \wedge i \neq j \rightarrow s = r(a,j) \text{ "}$$

Splitting on Demand in Abstract DPLL

How do we extend ADPLL Modulo Theories to handle such theory case-splits?

Splitting on Demand in Abstract DPLL

How do we extend ADPLL Modulo Theories to handle such theory case-splits?

Recall the T-Learn rule:

$$M \parallel F \quad \Longrightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } M \parallel F \\ F \models_T C \end{cases}$$

Splitting on Demand in Abstract DPLL

How do we extend ADPLL Modulo Theories to handle such theory case-splits?

Recall the T-Learn rule:

$$M \parallel F \quad \Longrightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } M \parallel F \\ F \models_T C \end{cases}$$

This rule allows a theory solver to send clauses to the DPLL engine as long as their atoms occur in $M \parallel F$.

Splitting on Demand in Abstract DPLL

How do we extend ADPLL Modulo Theories to handle such theory case-splits?

Recall the T-Learn rule:

$$M \parallel F \quad \Longrightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } M \parallel F \\ F \models_T C \end{cases}$$

This rule allows a theory solver to send clauses to the DPLL engine as long as their atoms occur in $M \parallel F$.

We wish to relax this requirement to allow additional atoms, possibly even containing new terms.

Extending Abstract DPLL Modulo Theories

Given an initial state $\emptyset \parallel F$, let $\mathcal{L}$ be an extended set of literals computed from F (see [BNOT06] for a formal definition of $\mathcal{L}$).

Extending Abstract DPLL Modulo Theories

Given an initial state $\emptyset \parallel F$, let $\mathcal{L}$ be an extended set of literals computed from F (see [BNOT06] for a formal definition of $\mathcal{L}$).

Clauses sent to the DPLL engine will be allowed to use any literal in $\mathcal{L}$.

Extending Abstract DPLL Modulo Theories

Given an initial state $\emptyset \parallel F$, let $\mathcal{L}$ be an extended set of literals computed from F (see [BNOT06] for a formal definition of $\mathcal{L}$).

Clauses sent to the DPLL engine will be allowed to use any literal in $\mathcal{L}$.

We extend the T-Learn rule as follows:

Extended T-Learn

$$M \parallel F \quad \Longrightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ are in } \mathcal{L} \\ F \models_T \exists \overline{v}.\, C \\ \overline{v} = \text{ "new" variables in } C \end{cases}$$

Extending Abstract DPLL Modulo Theories

Given an initial state $\emptyset \parallel F$, let $\mathcal{L}$ be an extended set of literals computed from F (see [BNOT06] for a formal definition of $\mathcal{L}$).

Clauses sent to the DPLL engine will be allowed to use any literal in $\mathcal{L}$.

We extend the T-Learn rule as follows:

Extended T-Learn

$$M \parallel F \quad \Longrightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ are in } \mathcal{L} \\ F \models_T \exists\, \overline{v}.\, C \\ \overline{v} = \text{ "new" variables in } C \end{cases}$$

Note: The set $\mathcal{L}$ never actually needs to be computed.

Extending Abstract DPLL Modulo Theories

Given an initial state $\emptyset \parallel F$, let $\mathcal{L}$ be an extended set of literals computed from F (see [BNOT06] for a formal definition of $\mathcal{L}$).

Clauses sent to the DPLL engine will be allowed to use any literal in $\mathcal{L}$.

We extend the T-Learn rule as follows:

Extended T-Learn

$$M \parallel F \quad \Longrightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ are in } \mathcal{L} \\ F \models_T \exists \overline{v}.\, C \\ \overline{v} = \text{ "new" variables in } C \end{cases}$$

Fact: For many theories with a theory solver, there exists an appropriate finite $\mathcal{L}$ for every input F.

Extending Abstract DPLL Modulo Theories

Now we can relax the requirement on the theory solver:

In the state $M \parallel G$, if $M \models G$, the theory solver must either

- decide whether $M \models_T \bot$ or
- generate a new clause by T-Learn containing at least one literal of $\mathcal{L}$ undefined in M.

Extending Abstract DPLL Modulo Theories

Now we can relax the requirement on the theory solver:

In the state $M \parallel G$, if $M \models G$, the theory solver must either

- decide whether $M \models_T \bot$ or
- generate a new clause by T-Learn containing at least one literal of $\mathcal{L}$ undefined in M.

Note: the T-solver is required to decide $M \models_T \bot$ only if all literals in $\mathcal{L}$ are defined in M.

Extending Abstract DPLL Modulo Theories

Now we can relax the requirement on the theory solver:

In the state $M \parallel G$, if $M \models G$, the theory solver must either

- decide whether $M \models_T \bot$ or
- generate a new clause by T-Learn containing at least one literal of $\mathcal{L}$ undefined in M.

Note: the T-solver is required to decide $M \models_T \bot$ only if all literals in $\mathcal{L}$ are defined in M.

Extending Abstract DPLL Modulo Theories

Now we can relax the requirement on the theory solver:

In the state $M \parallel G$, if $M \models G$, the theory solver must either

- decide whether $M \models_T \bot$ or
- generate a new clause by T-Learn containing at least one literal of $\mathcal{L}$ undefined in M.

Note: the T-solver is required to decide $M \models_T \bot$ only if all literals in $\mathcal{L}$ are defined in M.

In practice, tdetermine if $M \models_T \bot$ the T-solver only needs a small subset of $\mathcal{L}$ to be defined in M.

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F \qquad \Longrightarrow \qquad$ Propagate

$x = \{y\}, x = y \cup z \parallel F$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F$	$\Longrightarrow$	Propagate
$x = \{y\}, x = y \cup z \parallel F$	$\Longrightarrow$	Decide
$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet} \parallel F$		

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F$	$\Longrightarrow$	Propagate
$x = \{y\}, x = y \cup z \parallel F$	$\Longrightarrow$	Decide
$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet} \parallel F$	$\Longrightarrow$	Propagate
$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F$		

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F$	$\Longrightarrow$	Propagate
$x = \{y\}, x = y \cup z \parallel F$	$\Longrightarrow$	Decide
$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet} \parallel F$	$\Longrightarrow$	Propagate
$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F$	$\Longrightarrow$	T-Learn

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F \quad \Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z \parallel F \quad \Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet} \parallel F \quad \Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F \quad \Longrightarrow$ T-Learn

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ Decide

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \\ w \in x^{\bullet} \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F \quad \Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z \parallel F \quad \Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet} \parallel F \quad \Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F \quad \Longrightarrow$ T-Learn

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ Decide

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \\ w \in x^{\bullet} \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

$\Longrightarrow$ Propagate

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \\ w \in x^{\bullet}, w \notin z \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F$ $\Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z \parallel F$ $\Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet} \parallel F$ $\Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F$ $\Longrightarrow$ T-Learn

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z$
$w \in x^{\bullet}$ $\parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z$
$w \in x^{\bullet}, w \notin z$ $\parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

T-solver: $w \in y \cup z$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F \quad \Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z \parallel F \quad \Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet} \parallel F \quad \Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F \quad \Longrightarrow$ T-Learn

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ Decide

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \\ w \in x^{\bullet} \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

$\Longrightarrow$ Propagate

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \\ w \in x^{\bullet}, w \notin z \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

T-solver: $w \in y \cup z \quad \ldots \quad w \in y$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F \quad \Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z \parallel F \quad \Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet} \parallel F \quad \Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F \quad \Longrightarrow$ T-Learn

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ Decide

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \\ w \in x^{\bullet} \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

$\Longrightarrow$ Propagate

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \\ w \in x^{\bullet}, w \notin z \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

T-solver: $w \in y \cup z \quad \ldots \quad w \in y \quad \ldots \quad w \in \emptyset$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F \quad \Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z \parallel F \quad \Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet} \parallel F \quad \Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F \quad \Longrightarrow$ T-Learn

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z$
$w \in x^{\bullet}$ $\parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z$
$w \in x^{\bullet}, w \notin z$ $\parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

T-solver: $w \in y \cup z \quad \ldots \quad w \in y \quad \ldots \quad w \in \emptyset \quad \ldots \quad \perp$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F \quad \Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z \parallel F \quad \Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet} \parallel F \quad \Longrightarrow$ Propagate

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F \quad \Longrightarrow$ T-Learn

$x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ Decide

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \\ w \in x^{\bullet} \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

$\Longrightarrow$ Propagate

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\bullet}, x \neq z \\ w \in x^{\bullet}, w \notin z \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

T-solver: $w \in y \cup z \quad \ldots \quad w \in y \quad \ldots \quad w \in \emptyset \quad \ldots \quad \bot$

$\Longrightarrow$ T-Learn

$\ldots$

Correctness Results

Given the new rules, previous correctness results can be easily extended.

- Soundness: The new T-Learn rule maintains satisfiability of the clause set.

Correctness Results

Given the new rules, previous correctness results can be easily extended.

- Soundness: The new T-Learn rule maintains satisfiability of the clause set.
- Completeness: As long as the theory solver can decide $M \models_T \bot$ when all literals in $\mathcal{L}$ are determined, the system is still complete.

Correctness Results

Given the new rules, previous correctness results can be easily extended.

- Soundness: The new T-Learn rule maintains satisfiability of the clause set.
- Completeness: As long as the theory solver can decide $M \models_T \perp$ when all literals in $\mathcal{L}$ are determined, the system is still complete.
- Termination: The system terminates under the same conditions as the original system. Roughly:
  - Any lemma is (re)learned only finitely many times
  - Restart is applied with increased periodicity

Correctness Results

Given the new rules, previous correctness results can be easily extended.

- Soundness: The new T-Learn rule maintains satisfiability of the clause set.
- Completeness: As long as the theory solver can decide $M \models_T \bot$ when all literals in $\mathcal{L}$ are determined, the system is still complete.
- Termination: The system terminates under the same conditions as the original system. Roughly:
  - Any lemma is (re)learned only finitely many times
  - Restart is applied with increased periodicity

  The first condition can always be satisfied as $\mathcal{L}$ is finite.

Part II

From a single theory T to multiple theories $T_1, \ldots, T_n$.

The Combined Satisfiability Problem

For $i = 1, 2$,

- let T_i a first-order theory of signature Σ_i and
- let $\mathcal{L}^{\Sigma_i}$ be a class of Σ_i-formulas

such that the T_i-satisfiability problem for $\mathcal{L}^{\Sigma_i}$ is decidable.

The Combined Satisfiability Problem

For $i = 1, 2$,

- let T_i a first-order theory of signature Σ_i and
- let $\mathcal{L}^{\Sigma_i}$ be a class of Σ_i-formulas

such that the T_i-satisfiability problem for $\mathcal{L}^{\Sigma_i}$ is decidable.

Combination methods apply to languages $\mathcal{L}^{\Sigma_1 \cup \Sigma_2}$ that are effectively purifiable for T_1 and T_2,

The Combined Satisfiability Problem

For $i = 1, 2$,

- let T_i a first-order theory of signature Σ_i and
- let $\mathcal{L}^{\Sigma_i}$ be a class of Σ_i-formulas

such that the T_i-satisfiability problem for $\mathcal{L}^{\Sigma_i}$ is decidable.

Combination methods apply to languages $\mathcal{L}^{\Sigma_1 \cup \Sigma_2}$ that are effectively purifiable for T_1 and T_2, i.e., such that

the $(T_1 \cup T_2)$-satisfiability of a formula $\varphi \in \mathcal{L}^{\Sigma_1 \cup \Sigma_2}$
is effectively reducible to
the $(T_1 \cup T_2)$-satisfiability of formulas of the form $\varphi_1 \wedge \varphi_2$ with $\varphi_i \in \mathcal{L}^{\Sigma_i}$ for $i = 1, 2$.

An Effectively Purifiable Language

The language of conjunctions of literals is effectively purifiable for any T_1 and T_2.

An Effectively Purifiable Language

The language of conjunctions of literals is effectively purifiable for any T_1 and T_2.

Let φ be a conjunction of $(\Sigma_1 \cup \Sigma_2)$-literals.

1. Apply to completion to φ (modulo AC of $\wedge$) the following term abstraction rule:

$$\frac{L[t] \wedge \psi}{L[x] \wedge x \approx t \wedge \psi} \quad \text{if} \quad \begin{array}{l} x \text{ is a fresh variable and} \\ t \text{ is an alien subterm of } L \end{array}$$

2. Group the Σ_1-literals in φ_1 and the rest in φ_2.

An Effectively Purifiable Language

The language of conjunctions of literals is effectively purifiable for any T_1 and T_2.

Let φ be a conjunction of $(\Sigma_1 \cup \Sigma_2)$-literals.

1. Apply to completion to φ (modulo AC of $\wedge$) the following term abstraction rule:

$$\frac{L[t] \wedge \psi}{L[x] \wedge x \approx t \wedge \psi} \quad \text{if} \quad \begin{array}{l} x \text{ is a fresh variable and} \\ t \text{ is an alien subterm of } L \end{array}$$

2. Group the Σ_1-literals in φ_1 and the rest in φ_2.

Proposition For every $(\Sigma_1 \cup \Sigma_2)$-structure $\mathcal{A}$, φ is satisfiable in $\mathcal{A}$ iff $\varphi_1 \wedge \varphi_2$ is satisfiable in $\mathcal{A}$.

Combined Satisfiability of Pure Literals

From now on, wlog we consider only

combined satisfiability problems of the form

$$\varphi_1 \wedge \varphi_2$$

where each φ_i is a Σ_i-formula.

Combined Satisfiability of Pure Literals

From now on, wlog we consider only

combined satisfiability problems of the form

$$\varphi_1 \wedge \varphi_2$$

where each φ_i is a Σ_i-formula.

Observation: Such problems are really just interpolation problems.

Combined Satisfiability as Interpolation

For $i = 1, 2$, let T_i-be a Σ_i-theory and $\varphi_i(\mathbf{x}_i)$ a Σ_i-formula.

$\varphi_1 \wedge \varphi_2$ is $(T_1 \cup T_2)$-unsatisfiable

Combined Satisfiability as Interpolation

For $i = 1, 2$, let T_i-be a Σ_i-theory and $\varphi_i(\mathbf{x}_i)$ a Σ_i-formula.

$\varphi_1 \wedge \varphi_2$ is $(T_1 \cup T_2)$-unsatisfiable

iff

$$(T_1, \varphi_1), (T_2, \varphi_2) \models \perp$$

Combined Satisfiability as Interpolation

For $i = 1, 2$, let T_i-be a Σ_i-theory and $\varphi_i(\mathbf{x}_i)$ a Σ_i-formula.

$\varphi_1 \wedge \varphi_2$ is $(T_1 \cup T_2)$-unsatisfiable

iff

$$(T_1, \varphi_1), (T_2, \varphi_2) \models \bot$$

iff (by Craig's interpolation lemma)

there is a $(\Sigma_1 \cap \Sigma_2)$-formula $\varphi(\mathbf{x})$ with $\mathbf{x} = \mathbf{x}_1 \cap \mathbf{x}_2$ s.t.

$$T_1, \varphi_1 \models \varphi \quad \text{and} \quad T_2, \varphi_2, \varphi \models \bot$$

Combined Satisfiability as Interpolation

For $i = 1, 2$, let T_i-be a Σ_i-theory and $\varphi_i(\mathbf{x}_i)$ a Σ_i-formula.

$\varphi_1 \wedge \varphi_2$ is $(T_1 \cup T_2)$-unsatisfiable

iff

$$(T_1, \varphi_1), (T_2, \varphi_2) \models \perp$$

iff (by Craig's interpolation lemma)

there is a $(\Sigma_1 \cap \Sigma_2)$-formula $\varphi(\mathbf{x})$ with $\mathbf{x} = \mathbf{x}_1 \cap \mathbf{x}_2$ s.t.

$$T_1, \varphi_1 \models \varphi \quad \text{and} \quad T_2, \varphi_2, \varphi \models \perp$$

The problem then is "just" computing the interpolant φ.

Combined Satisfiability as Interpolation

For $i = 1, 2$, let T_i-be a Σ_i-theory and $\varphi_i(\mathbf{x}_i)$ a Σ_i-formula.

$\varphi_1 \wedge \varphi_2$ is $(T_1 \cup T_2)$-unsatisfiable

iff

$$(T_1, \varphi_1), (T_2, \varphi_2) \models \bot$$

iff (by Craig's interpolation lemma)

there is a $(\Sigma_1 \cap \Sigma_2)$-formula $\varphi(\mathbf{x})$ with $\mathbf{x} = \mathbf{x}_1 \cap \mathbf{x}_2$ s.t.

$T_1, \varphi_1 \models \varphi$ and $T_2, \varphi_2, \varphi \models \bot$

Unfortunately, Craig's lemma provides no information on

- what φ looks like or
- how to compute φ without an explicit proof that $T_1, T_2, \varphi_1, \varphi_2 \models \bot$.

Combined Satisfiability as Interpolation

For $i = 1, 2$, let T_i-be a Σ_i-theory and $\varphi_i(\mathbf{x}_i)$ a Σ_i-formula.

$\varphi_1 \wedge \varphi_2$ is $(T_1 \cup T_2)$-unsatisfiable

iff

$$(T_1, \varphi_1), (T_2, \varphi_2) \models \bot$$

iff (by Craig's interpolation lemma)

there is a $(\Sigma_1 \cap \Sigma_2)$-formula $\varphi(\mathbf{x})$ with $\mathbf{x} = \mathbf{x}_1 \cap \mathbf{x}_2$ s.t.

$T_1, \varphi_1 \models \varphi$ and $T_2, \varphi_2, \varphi \models \bot$

All existing combination methods are in essence ways to compute φ, possibly incrementally, in finite time.

The Combined Satisfiability Problem for QFFs

For $i = 1, 2$,

- let T_i a first-order theory of signature Σ_i and
- let P_i be a procedure that decides the T_i-satisfiability problem for quantifier-free Σ_i-formulas.

The Combined Satisfiability Problem for QFFs

For $i = 1, 2$,

- let T_i a first-order theory of signature Σ_i and
- let P_i be a procedure that decides the T_i-satisfiability problem for quantifier-free Σ_i-formulas.

How to decide the $(T_1 \cup T_2)$-satisfiability problem for quantifier-free $(\Sigma_1 \cup \Sigma_2)$-formulas using P_1 and P_2 modularly?

The Combination Problem for QFFs

- Problem most people mean when talking about combining decision procedures.
- Problem with the largest impact and most practical uses so far.
- Most common settings:
  - T_1 and T_2 are signature-disjoint.
- Basic combination method for the problem due to Greg Nelson and Derek Oppen [NO79].

The Nelson-Oppen Method

- For $i = 1, 2$, let T_i a first-order theory of signature Σ_i.
- Let $T = T_1 \cup T_2$.
- Let C be a set of free constants (i.e., not in $\Sigma_1 \cup \Sigma_2$).

The Nelson-Oppen Method

- For $i = 1, 2$, let T_i a first-order theory of signature Σ_i.
- Let $T = T_1 \cup T_2$.
- Let C be a set of free constants (i.e., not in $\Sigma_1 \cup \Sigma_2$).

We consider only input problems of the form

$$\Gamma_1 \cup \Gamma_2$$

where each Γ_i is a finite set of ground $\Sigma_i(C)$-literals.

The Nelson-Oppen Method

No loss of generality in considering ground $\Sigma_i(C)$-literals as:

The Nelson-Oppen Method

No loss of generality in considering ground $\Sigma_i(C)$-literals as:

1. for each $\varphi(\mathbf{x}) \in \mathrm{QF}(\Sigma_1 \cup \Sigma_2, X)$,

   $\varphi(\mathbf{x})$ is T-sat iff $\varphi(\mathbf{c})$ is T-sat for some $\mathbf{c}$ in C

The Nelson-Oppen Method

No loss of generality in considering ground $\Sigma_i(C)$-literals as:

1. for each $\varphi(\mathbf{x}) \in \mathrm{QF}(\Sigma_1 \cup \Sigma_2, X)$,

   $\varphi(\mathbf{x})$ is T-sat iff $\varphi(\mathbf{c})$ is T-sat for some $\mathbf{c}$ in C

2. for each ground $\varphi(\mathbf{c})$,

   $\varphi(\mathbf{c})$ is T-sat iff one disjunct ψ of $\varphi(\mathbf{c})$'s DNF is T-sat

The Nelson-Oppen Method

No loss of generality in considering ground $\Sigma_i(C)$-literals as:

1. for each $\varphi(\mathbf{x}) \in \mathrm{QF}(\Sigma_1 \cup \Sigma_2, X)$,
   $\varphi(\mathbf{x})$ is T-sat iff $\varphi(\mathbf{c})$ is T-sat for some $\mathbf{c}$ in C
2. for each ground $\varphi(\mathbf{c})$,
   $\varphi(\mathbf{c})$ is T-sat iff one disjunct ψ of $\varphi(\mathbf{c})$'s DNF is T-sat
3. for each conjunction ψ of literals,
   ψ is T-sat iff its separate form $\psi_1 \wedge \psi_2$ is T-sat

The Nelson-Oppen Method

No loss of generality in considering ground $\Sigma_i(C)$-literals as:

1. for each $\varphi(\mathbf{x}) \in \mathrm{QF}(\Sigma_1 \cup \Sigma_2, X)$,

   $\varphi(\mathbf{x})$ is T-sat iff $\varphi(\mathbf{c})$ is T-sat for some $\mathbf{c}$ in C

2. for each ground $\varphi(\mathbf{c})$,

   $\varphi(\mathbf{c})$ is T-sat iff one disjunct ψ of $\varphi(\mathbf{c})$'s DNF is T-sat

3. for each conjunction ψ of literals,

   ψ is T-sat iff its separate form $\psi_1 \wedge \psi_2$ is T-sat

4. for each conjunction $\psi_1 \wedge \psi_2$ of literals,

   $\psi_1 \wedge \psi_2$ is T-sat iff $\Gamma_1 \cup \Gamma_2$ is T-sat where each Γ_i is the set of literals in ψ_i.

The Nelson-Oppen Method

Barebone, non-deterministic, non-incremental version [Opp80, Rin96, TH96]:

The Nelson-Oppen Method

Barebone, non-deterministic, non-incremental version [Opp80, Rin96, TH96]:

Input:	$\Gamma_1 \cup \Gamma_2$ with Γ_i a finite set of ground $\Sigma_i(C)$-literals.
Output:	**sat** or **unsat**.

The Nelson-Oppen Method

Barebone, non-deterministic, non-incremental version [Opp80, Rin96, TH96]:

Input: $\Gamma_1 \cup \Gamma_2$ with Γ_i a finite set of ground $\Sigma_i(C)$-literals.
Output: **sat** or **unsat**.

1. Guess an arrangement Δ, that is:
   - Choose any equivalence relation R on the constants from C shared by Γ_1 and Γ_2.
   - Let $\Delta = \{c \approx d \mid cRd\} \cup \{c \not\approx d \mid$ not $cRd\}$

The Nelson-Oppen Method

Barebone, non-deterministic, non-incremental version [Opp80, Rin96, TH96]:

Input: $\Gamma_1 \cup \Gamma_2$ with Γ_i a finite set of ground $\Sigma_i(C)$-literals.
Output: **sat** or **unsat**.

1. Guess an arrangement Δ, that is:
   - Choose any equivalence relation R on the constants from C shared by Γ_1 and Γ_2.
   - Let $\Delta = \{c \approx d \mid cRd\} \cup \{c \not\approx d \mid \text{not } cRd\}$
2. If $\Gamma_i \cup \Delta$ is T_i-unsatisfiable for $i = 1$ or $i = 2$, return **unsat**

The Nelson-Oppen Method

Barebone, non-deterministic, non-incremental version [Opp80, Rin96, TH96]:

Input: $\Gamma_1 \cup \Gamma_2$ with Γ_i a finite set of ground $\Sigma_i(C)$-literals.
Output: **sat** or **unsat**.

1. Guess an arrangement Δ, that is:
   - Choose any equivalence relation R on the constants from C shared by Γ_1 and Γ_2.
   - Let $\Delta = \{c \approx d \mid cRd\} \cup \{c \not\approx d \mid \text{not } cRd\}$
2. If $\Gamma_i \cup \Delta$ is T_i-unsatisfiable for $i = 1$ or $i = 2$, return **unsat**
3. Otherwise, return **sat**

Total Correctness of the NO Method

The method is always terminating because there is only a finite number of arrangements to guess.

Total Correctness of the NO Method

The method is always terminating because there is only a finite number of arrangements to guess.

When

- $\Sigma_1 \cap \Sigma_2 = \emptyset$ and
- T_1 and T_2 are stably infinite,

the method is sound and complete.

Soundness:

If the answer is **unsat** for every arrangement,
then the input is $(T_1 \cup T_2)$-unsatisfiable.

Completeness:

If the input is $(T_1 \cup T_2)$-is unsatisfiable,
then the answer is **unsat** for every arrangement.

Stably Infinite Theories

A Σ-theory T is stably infinite iff every quantifier-free T-satisfiable formula is satisfiable in an infinite model of T.

Stably Infinite Theories

A Σ-theory T is stably infinite iff every quantifier-free T-satisfiable formula is satisfiable in an infinite model of T.

Many *interesting* theories are stably infinite:

- Theories of an infinite structure.
- Complete theories with an infinite model.
- Convex theories with no trivial models (see later).

Stably Infinite Theories

A Σ-theory T is stably infinite iff every quantifier-free T-satisfiable formula is satisfiable in an infinite model of T.

Many *interesting* theories are stably infinite:

- Theories of an infinite structure.
- Complete theories with an infinite model.
- Convex theories with no trivial models (see later).

But others are not stably infinite:

- Theories of a finite structure.
- Theories with models of bounded cardinality.
- Some equational/Horn theories.

The NO Calculus

Declarative, non-deterministic, incremental version of the NO method

The NO Calculus

Declarative, non-deterministic, incremental version of the NO method

Let C_0 be the free constants shared by the initial Γ_1^0 and Γ_2^0.

The NO Calculus

Declarative, non-deterministic, incremental version of the NO method

Let C_0 be the free constants shared by the initial Γ_1^0 and Γ_2^0.

Apply these rules exhaustively, starting with the triple $\Gamma_1^0;\ \emptyset;\ \Gamma_2^0$:

The NO Calculus

Declarative, non-deterministic, incremental version of the NO method

Let C_0 be the free constants shared by the initial Γ_1^0 and Γ_2^0.

Apply these rules exhaustively, starting with the triple $\Gamma_1^0;\ \emptyset;\ \Gamma_2^0$:

$$\frac{\Gamma_1;\ \Delta;\ \Gamma_2}{\bot} \quad \text{if } \Gamma_i, \Delta \models_{T_i} \bot \text{ for } i = 1 \text{ or } i = 2$$

$$\frac{\Gamma_1;\ \Delta;\ \Gamma_2}{\Gamma_1;\ \Delta, c \approx d;\ \Gamma_2 \qquad\qquad \Gamma_1;\ \Delta, c \not\approx d;\ \Gamma_2} \quad \text{if } \begin{cases} c, d \in C_0, \\ c \approx d \notin \Delta, \\ c \not\approx d \notin \Delta \end{cases}$$

Correctness of the NO Calculus

Some terminology:

- A derivation tree in the NO calculus is a tree such that
  - every node is either a triple $\Gamma;\ \Delta;\ \Gamma$ or $\perp$
  - a node N is a child of a M only if it is a direct consequence of M.
- A derivation tree for $\Gamma_1;\ \Delta;\ \Gamma_2$ is a derivation tree with root $\Gamma_1;\ \Delta;\ \Gamma_2$.
- A refutation tree is a derivation tree all of whose leaves are $\perp$.

Correctness of the NO Calculus

The NO calculus is sound, complete and terminating whenever T_1 and T_2 are stably infinite and signature-disjoint.

Termination:

Every derivation tree in NO is finite.

Soundness and Completeness:

$\Gamma_1 \cup \Gamma_2$ is $(T_1 \cup T_2)$-unsatisfiable
iff
$\Gamma_1;\ \emptyset;\ \Gamma_2$ has a refutation tree in NO.

The d-NO Calculus

Declarative, (more) deterministic, incremental version of the NO method (more faithful to the original [NO79])

The d-NO Calculus

Declarative, (more) deterministic, incremental version of the NO method (more faithful to the original [NO79])

Apply these rules exhaustively, starting with $\Gamma_1^0;\ \emptyset;\ \Gamma_2^0$:

$$\frac{\Gamma_1;\ \Delta;\ \Gamma_2}{\bot} \quad \text{if } \Gamma_i, \Delta \models_{T_i} \bot \text{ for } i = 1 \text{ or } i = 2$$

$$\frac{\Gamma_1;\ \Delta;\ \Gamma_2}{\Gamma_1;\ \Delta, c_1 \approx d_1;\ \Gamma_2 \qquad \cdots \qquad \Gamma_1;\ \Delta, c_n \approx d_n;\ \Gamma_2} \quad \text{if } (*)$$

$$(*) = \begin{cases} n \geq 1,\ c_1, \ldots, c_n, d_1, \ldots, d_n \in C_0, \\ i \in \{1, 2\},\ J = \{1, \ldots, n\}, \\ \Gamma_i, \Delta \models_{T_i} \bigvee_{j \in J} c_j \approx d_j \\ \Gamma_i, \Delta \not\models_{T_i} \bigvee_{j \in J'} c_j \approx d_j \text{ for any } J' \subsetneq J \end{cases}$$

The d-NO Calculus and Convex Theories

The d-NO calculus becomes really deterministic when T_1 and T_2 are convex.

Then, every refutation tree consists of a single branch.

The d-NO Calculus and Convex Theories

The d-NO calculus becomes really deterministic when T_1 and T_2 are convex.

Then, every refutation tree consists of a single branch.

A Σ-theory T is convex iff

for all finite sets Γ of Σ-literals and
for all non-empty disjunctions $\bigvee_{i \in I} x_i \approx y_i$ of variables,

$$\Gamma \models_T \bigvee_{i \in I} x_i \approx y_i \text{ iff } \Gamma \models_T x_i \approx y_i \text{ for some } i \in I.$$

The d-NO Calculus and Convex Theories

The d-NO calculus becomes really deterministic when T_1 and T_2 are convex.

Then, every refutation tree consists of a single branch.

A Σ-theory T is convex iff

for all finite sets Γ of Σ-literals and
for all non-empty disjunctions $\bigvee_{i \in I} x_i \approx y_i$ of variables,

$$\Gamma \models_T \bigvee_{i \in I} x_i \approx y_i \ \text{ iff } \ \Gamma \models_T x_i \approx y_i \text{ for some } i \in I.$$

Useful fact: Every convex theory T with no trivial models (i.e., such that $T \models \exists x, y. x \not\approx y$) is stably infinite [BDS02b].

The d-NO Calculus and Convex Theories

Many interesting theories are convex (not immediate to show):

- All Horn theories—this includes all (conditional) equational theories.
- Some non-Horn theories, like linear rational arithmetic.

The d-NO Calculus and Convex Theories

Many interesting theories are convex (not immediate to show):

- All Horn theories—this includes all (conditional) equational theories.
- Some non-Horn theories, like linear rational arithmetic.

But many more are not convex:

- All theories of a finite structure.
- Non-linear rational arithmetic.
- Linear integer arithmetic.
- The theory of arrays.

Extending Nelson-Oppen

The main requirements of the method:

- The disjointness of Σ_1 and Σ_2 and
- the stable infiniteness of T_1 and T_2

are only sufficient conditions for its correctness.

Can they be relaxed?

Extending Nelson-Oppen

The main requirements of the method:

- The disjointness of Σ_1 and Σ_2 and
- the stable infiniteness of T_1 and T_2

are only sufficient conditions for its correctness.

Can they be relaxed?

Relaxing either of them turns out to be rather hard.

Only a few results in this direction, all very recent, and most of them mainly of academic interest for now.

Extending NO: Non-Stably Infinite Theories

The only existing results (we are aware of) are about

- combining arbitrary theories with the theory of equality (aka the empty theory, EUF, . . .) [Gan02],
- about combining arbitrary theories with shiny or polite theories [TZ05, RRZ05]
- combining universal theories [Zar04].

The results in [TZ05, RRZ05] subsume those in [Gan02] but are not comparable to those in [Zar04].

The results in [Zar04] also lift the disjointness restriction.

Extending Nelson-Oppen: Non-Disjoint Theories

Three main approaches, respectively described in: [TR03], [Ghi04], and [Zar04].

All of them need to extend the constraint sharing mechanism beyond (dis)equalities of shared constants.

None of them is more general than the others.

[TR03] and [Ghi04] are rather technical and beyond the scope of this talk.

[Zar04] is very general but yields weaker results both in theory (only semi-decidability) and in practice (too much to guess).

Abstract DPLL Modulo Multiple Theories

Let $T_1, \ldots, T_n$ be distinct theories with respective theory solvers $S_1, \ldots, S_n$.

How can we reason over all of them with Abstract DPLL?

Abstract DPLL Modulo Multiple Theories

Let $T_1, \ldots, T_n$ be distinct theories with respective theory solvers $S_1, \ldots, S_n$.

How can we reason over all of them with Abstract DPLL?

Quick Solution:

1. Combine $S_1, \ldots, S_n$ with Nelson-Oppen into a T-solver for $T = T_1 \cup \cdots \cup T_n$.
2. Use Abstract DPLL Modulo T.

Abstract DPLL Modulo Multiple Theories

Let $T_1, \ldots, T_n$ be distinct theories with respective theory solvers $S_1, \ldots, S_n$.

How can we reason over all of them with Abstract DPLL?

Better Solution [Bar02, Tin04, BBC$^+$05, BNOT06]:

1. Lift Nelson-Oppen to the DPLL level.
2. Use Abstract DPLL Modulo $T_1, \ldots, T_n$.

Abstract DPLL Modulo Multiple Theories

Preliminaries

- Let $n = 2$, for simplicity.
- Let T_i be of signature Σ_i for $i = 1, 2$, with $\Sigma_1 \cap \Sigma_2 = \emptyset$.
- Let C be a set of free constants.
- Assume wlog that each input literal has signature $\Sigma_1(C)$ or $\Sigma_2(C)$ (no mixed literals).
- Let $M^i = \{\Sigma_i(C)$-literals of $M\}$.
- Let $se(M) = \{c \approx d \mid c, d$ occur in C, M^1 and $M^2\}$ (*shared equalities*).

Abstract DPLL – Rules for Multiple Theories

Propagate (unchanged)

Fail (unchanged)

T-Backjump (unchanged, with $T = T_1 \cup T_2$)

Decide

$$M \parallel F \quad \longrightarrow \quad M\, l^{\bullet} \parallel F \quad \textbf{if} \quad \begin{cases} l \text{ or } \bar{l} \text{ occurs in } M \parallel F \text{ or in } se(M), \\ l \text{ is undefined in } M \end{cases}$$

Only change: decide on (undefined) shared equalities as well.

Abstract DPLL – Rules for Multiple Theories

Refined T-Learn

$$M \parallel F \quad \to \quad M \parallel F, \overline{l_1} \vee \ldots \vee \overline{l_n} \quad \textbf{if} \begin{cases} i \in \{1,2\} \\ l_j \text{ or } \bar{l}_j \text{ in } M^i \text{ or } se(M) \\ l_1 \wedge \cdots \wedge l_n \models_{T_i} \perp \end{cases}$$

T-Propagate

$$M \parallel F \quad \to \quad M\, l \parallel F \quad \textbf{if} \begin{cases} i \in \{1,2\} \\ M^i \models_{T_i} l \\ l \text{ or } \bar{l} \text{ occurs in } M \parallel F \text{ or } se(M) \\ l \text{ is undefined in } M \end{cases}$$

Changes: (i) reason locally in T_i, (ii) theory propagate shared equalities as well.

References

[ABC+02] Gilles Audemard, Piergiorgio Bertoli, Alessandro Cimatti, Artur Korniłowicz, and Roberto Sebastiani. A SAT-based approach for solving formulas over boolean and linear mathematical propositions. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 195–210. Springer, 2002

[ACG00] Alessandro Armando, Claudio Castellini, and Enrico Giunchiglia. SAT-based procedures for temporal reasoning. In S. Biundo and M. Fox, editors, *Proceedings of the 5th European Conference on Planning (Durham, UK)*, volume 1809 of *Lecture Notes in Computer Science*, pages 97–108. Springer, 2000

[Bar02] Clark W. Barrett. *Checking Validity of Quantifier-Free Formulas in Combinations of First-Order Theories*. PhD dissertation, Department of Computer Science, Stanford University, Stanford, CA, Sep 2002

[BBC+05] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi Junttila, Silvio Ranise, Roberto Sebastiani, and Peter van Rossu. Efficient satisfiability modulo theories via delayed theory combination. In K.Etessami and S. Rajamani, editors, *Proceedings of the 17th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science. Springer, 2005. (To appear)

References

[BCLZ04] Thomas Ball, Byron Cook, Shuvendu K. Lahiri, and Lintao Zhang. Zapato: Automatic theorem proving for predicate abstraction refinement. In R. Alur and D. Peled, editors, *Proceedings of the 16th International Conference on Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 457–461. Springer, 2004

[BDS02a] Clark W. Barrett, David L. Dill, and Aaron Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In J. C. Godskesen, editor, *Proceedings of the International Conference on Computer-Aided Verification*, Lecture Notes in Computer Science, 2002

[BDS02b] Clark W. Barrett, David L. Dill, and Aaron Stump. A generalization of Shostak's method for combining decision procedures. In A. Armando, editor, *Proceedings of the 4th International Workshop on Frontiers of Combining Systems, FroCoS'2002 (Santa Margherita Ligure, Italy)*, volume 2309 of *Lecture Notes in Computer Science*, pages 132–147, apr 2002

[BLS02] Randal E. Bryant, Shuvendu K. Lahiri, and Sanjit A. Seshia. Deciding CLU logic formulas via boolean and pseudo-boolean encodings. In *Proc. Intl. Workshop on Constraints in Formal Verification*, 2002

References

[BNOT06] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Splitting on demand in sat modulo theories. In M. Hermann and A. Voronkov, editors, *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'06), Phnom Penh, Cambodia*, volume 4246 of *Lecture Notes in Computer Science*, pages 512–526. Springer, 2006

[BT02] Franz Baader and Cesare Tinelli. Deciding the word problem in the union of equational theories. *Information and Computation*, 178(2):346–390, December 2002

[BT03] Peter Baumgartner and Cesare Tinelli. The model evolution calculus. In F. Baader, editor, *Proceedings of the 19th International Conference on Automated Deduction, CADE-19 (Miami, Florida, USA)*, number 2741 in Lecture Notes in Artificial Intelligence, pages 350–364. Springer, 2003

[CKSY04] Edmund Clarke, Daniel Kroening, Natasha Sharygina, and Karen Yorav. Predicate abstraction of ANSI–C programs using SAT. *Formal Methods in System Design (FMSD)*, 25:105–127, September–November 2004

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962

References

[dMR02] Leonardo de Moura and Harald Rueß. Lemmas on demand for satisfiability solvers. In *Proc. of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT'02)*, May 2002

[FJOS03] Cormac Flanagan, Rajeev Joshi, Xinming Ou, and James B. Saxe. Theorem proving using lazy proof explication. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2003

[Gan02] Harald Ganzinger. Shostak light. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, volume 2392 of *Lecture Notes in Computer Science*, pages 332–346. Springer-Verlag, jul 2002

[Ghi04] Silvio Ghilardi. Model theoretic methods in combined constraint satisfiability. *Journal of Automated Reasoning*, 3(3–4):221–249, 2004

[GHN$^+$04] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast decision procedures. In R. Alur and D. Peled, editors, *Proceedings of the 16th International Conference on Computer Aided Verification, CAV'04 (Boston, Massachusetts)*, volume 3114 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2004

References

[NO79] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, October 1979

[NO05] Robert Nieuwenhuis and Albert Oliveras. DPLL(T) with exhaustive theory propagation and its application to difference logic. In K. Etessami and S. Rajamani, editors, *Proceedings of 17th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science. Springer, 2005. (To appear)

[NOT05] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Abstract DPLL and abstract DPLL modulo theories. In F. Baader and A. Voronkov, editors, *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'04), Montevideo, Uruguay*, volume 3452 of *Lecture Notes in Artificial Intelligence*, pages 36–50. Springer, 2005

[Opp80] Derek C. Oppen. Complexity, convexity and combinations of theories. *Theoretical Computer Science*, 12:291–302, 1980

[Rin96] Christophe Ringeissen. Cooperation of decision procedures for the satisfiability problem. In F. Baader and K.U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 121–140. Kluwer Academic Publishers, March 1996

References

[RRZ05] Silvio Ranise, Christophe Ringeissen, and Calogero G. Zarba. Combining data structures with nonstably infinite theories using many-sorted logic. In B. Gramlich, editor, *Proceedings of the Workshop on Frontiers of Combining Systems*, Lecture Notes in Computer Science. Springer, 2005. (To appear.)

[SLB03] Sanjit A. Seshia, Shuvendu K. Lahiri, and Randal E. Bryant. A hybrid SAT-based decision procedure for separation logic with uninterpreted functions. In *Proc. 40th Design Automation Conference*, pages 425–430. ACM Press, 2003

[TH96] Cesare Tinelli and Mehdi T. Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In F. Baader and K.U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop (Munich, Germany)*, Applied Logic, pages 103–120. Kluwer Academic Publishers, March 1996

[Tin02] Cesare Tinelli. A DPLL-based calculus for ground satisfiability modulo theories. In Giovambattista Ianni and Sergio Flesca, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (Cosenza, Italy)*, volume 2424 of *Lecture Notes in Artificial Intelligence*. Springer, 2002

[Tin04] Cesare Tinelli. The DPLL($T_1, \ldots, T_n$): modeling DPLL-based checkers for satisfiability modulo multiple theories. (Unpublished), 2004

References

[TR03] Cesare Tinelli and Christophe Ringeissen. Unions of non-disjoint theories and combinations of satisfiability procedures. *Theoretical Computer Science*, 290(1):291–353, January 2003

[TZ04] Cesare Tinelli and Calogero Zarba. Combining decision procedures for sorted theories. In J. Alferes and J. Leite, editors, *Proceedings of the 9th European Conference on Logic in Artificial Intelligence (JELIA'04), Lisbon, Portugal*, volume 3229 of *Lecture Notes in Artificial Intelligence*, pages 641–653. Springer, 2004

[TZ05] Cesare Tinelli and Calogero Zarba. Combining nonstably infinite theories. *Journal of Automated Reasoning*, 34(3):209–238, April 2005

[Zar04] Calogero G. Zarba. C-tableaux. Technical Report RR-5229, INRIA, 2004