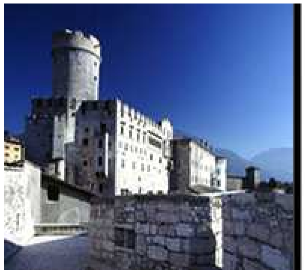
Foundations of Lazy SMT and DPLL(T)

Cesare Tinelli

The University of Iowa



Second International

SAT / SMT Summer School

June 12th - June 15th, 2012 | Trento | Italy

Acknowledgments: Many thanks to Albert Oliveras for contributing some of the material used in these slides.

Disclamer: The literature on SMT and its applications is already vast. The bibliographic references provided here are just a sample. Apologies to all authors whose work is not cited.

Introduction

Historically, automated reasoning $\equiv$ uniform proof-search procedures for First Order Logic

Limited success: is FOL the best compromise between expressivity and efficiency?

More recent trend [Sha02] focuses on:

- addressing mostly (expressive enough) decidable fragments of a certain logic
- incorporating domain-specific reasoning, e.g on:
  - arithmetic reasoning
  - equality
  - data structures (arrays, lists, stacks, ...)

Introduction

Examples of this trend:

SAT: propositional formalization, Boolean reasoning

- $+$ high degree of efficiency
- $-$ expressive (all NP-complete problems) but involved encodings

SMT: first-order formalization, Boolean $+$ domain-specific reasoning

- $+$ improves expressivity and scalability
- $-$ some (but acceptable) loss of efficiency

Introduction

Examples of this trend:

SAT: propositional formalization, Boolean reasoning

- $+$ high degree of efficiency
- $-$ expressive (all NP-complete problems) but involved encodings

SMT: first-order formalization, Boolean $+$ domain-specific reasoning

- $+$ improves expressivity and scalability
- $-$ some (but acceptable) loss of efficiency

This lecture: overview of SMT formal foundations

The SMT Problem

Some problems are more naturally expressed in logics other than propositional logic, e.g:

- Software verification needs reasoning about equality, arithmetic, data structures, ...

SMT is about deciding the satisfiability of a (usually quantifier-free) FOL formula with respect to some *background theory*

- Example (Equality with Uninterpreted Functions):

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

Wide range of applications: Extended Static Checking [FLL$^+$02], Predicate abstraction [LNO06], Model checking [AMP06, HT08], Scheduling [BNO$^+$08b], Test generation [TdH08], ...

Theories of Interest: EUF

Equality ($=$) with Uninterpreted Functions [NO80, BD94, NO07]

Typically used to abstract unsupported constructs, e.g.:

- non-linear multiplication in arithmetic
- ALUs in circuits

Example: The formula

$$a * (|b| + c) = d \;\; \wedge \;\; b * (|a| + c) \neq d \;\; \wedge \;\; a = b$$

is unsatisfiable, but no arithmetic reasoning is needed

If we abstract it to

$$mul(a, add(abs(b), c)) = d \;\; \wedge \;\; mul(b, add(abs(a), c)) \neq d \;\; \wedge \;\; a = b$$

it is still unsatisfiable

Theories of Interest: Arithmetic(s)

Very useful, for obvious reasons

Restricted fragments (over the reals or the integers) support more efficient methods:

- Bounds: $x \bowtie k$ with $\bowtie \in \{<, >, \leq, \geq, =\}$ [BBC+05a]
- Difference logic: $x - y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$ [NO05, WIGG05, CM06]
- UTVPI: $\pm x \pm y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$ [LM05]
- Linear arithmetic, e.g: $2x - 3y + 4z \leq 5$ [DdM06]
- Non-linear arithmetic, e.g: $2xy + 4xz^2 - 5y \leq 10$ [BLNM+09, ZM10]

Theories of Interest: Arrays

Used in software verification and hardware verification (for memories) [SBDL01, BNO+08a, dMB09]

Two interpreted function symbols read and write

Axiomatized by:

- $\forall a\, \forall i\, \forall v\ \mathrm{read}(\mathrm{write}(a,i,v),i) = v$
- $\forall a\, \forall i\, \forall j\, \forall v\ i \neq j \to \mathrm{read}(\mathrm{write}(a,i,v),j) = \mathrm{read}(a,j)$

Sometimes also with *extensionality*:

- $\forall a\, \forall b\ (\forall i\, \mathrm{read}(a,i) = \mathrm{read}(b,i)\ \to a = b)$

Is the following set of literals satisfiable in this theory?

$$\mathrm{write}(a,i,x) \neq b,\ \mathrm{read}(b,i) = y,\ \mathrm{read}(\mathrm{write}(b,i,x),j) = y,\ a = b,\ i = j$$

Theories of Interest: Bit vectors

Useful both in hardware and software verification [BCF+07, BB09]

Universe consists of (fixed-sized) vectors of bits

Different types of operations:

- *String-like*: concat, extract, ...
- *Logical*: bit-wise not, or, and, ...
- *Arithmetic*: add, subtract, multiply, ...
- *Comparison*: $<, >$, ...

Is this formula satisfiable over bit vectors of size 3?

$$a[0:1] \neq b[0:1] \ \wedge \ (a \mid b) = c \ \wedge \ c[0] = 0 \ \wedge \ a[1] + b[1] = 0$$

Combinations of Theories

In practice, theories are seldom used in isolation

E.g., software verifications may need a combination of arrays, arithmetic, bit vectors, data types, ...

Formulas of the following form usually arise:

$$i = j + 2 \;\; \wedge \;\; a = \mathrm{write}(b, i+1, 4) \;\; \wedge$$
$$(\mathrm{read}(a, j+3) = 2 \;\vee\; f(i-1) \neq f(j+1))$$

Often decision procedures for each theory combine modularly [NO79, TH96, BBC+05b]

Solving SMT Problems

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

Solving SMT Problems

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

Problem: In practice, we need to deal with

1. arbitrary Boolean combinations of literals
2. literals over more than one theory
3. formulas with quantifiers

Solving SMT Problems

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

Problem: In practice, we need to deal with

1. arbitrary Boolean combinations of literals
2. literals over more than one theory
3. formulas with quantifiers

This lecture focuses more on general methods to address (1), mostly, and (2)

More details on (2) and (3) will be given in later lectures today

Structure of this Lecture

Introduction

Part I

From sets of literals to arbitrary quantifier-free formulas

Part II

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

Part I

From sets of literals to arbitrary quantifier-free formulas

Satisfiability Modulo a Theory T

Def. A formula is *(un)satisfiable in* a theory T, or T*-(un)satisfiable*, if there is a (no) model of T that satisfies it

Satisfiability Modulo a Theory T

Def. A formula is *(un)satisfiable in* a theory T, or T-*(un)satisfiable*, if there is a (no) model of T that satisfies it

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

Satisfiability Modulo a Theory T

Def. A formula is *(un)satisfiable in* a theory T, or T*-(un)satisfiable*, if there is a (no) model of T that satisfies it

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

(Convert the formula in DNF and check if any of its disjuncts is T-sat)

Satisfiability Modulo a Theory T

Def. A formula is *(un)satisfiable in* a theory T, or T*-(un)satisfiable*, if there is a (no) model of T that satisfies it

Note: The T-satisfiability of quantifier-free formulas is decidable iff the T-satisfiability of conjunctions/sets of literals is decidable
(Convert the formula in DNF and check if any of its disjuncts is T-sat)

Problem: In practice, dealing with Boolean combinations of literals is as hard as in propositional logic

Satisfiability Modulo a Theory T

Def. A formula is *(un)satisfiable in* a theory T, or T-*(un)satisfiable*, if there is a (no) model of T that satisfies it

Note: The T-satisfiability of quantifier-free formulas is decidable iff the T-satisfiability of conjunctions/sets of literals is decidable
(Convert the formula in DNF and check if any of its disjuncts is T-sat)

Problem: In practice, dealing with Boolean combinations of literals is as hard as in propositional logic

Current solution: Exploit propositional satisfiability technology

Lifting SAT Technology to SMT

Two main approaches:

Lifting SAT Technology to SMT

Two main approaches:

1. "Eager" [PRSS99, SSB02, SLB03, BGV01, BV02]
    - translate into an equisatisfiable propositional formula
    - feed it to any SAT solver

Notable systems: UCLID

Lifting SAT Technology to SMT

Two main approaches:

2. "Lazy" [ACG00, dMR02, BDS02, ABC$^+$02]
   - abstract the input formula to a propositional one
   - feed it to a (DPLL-based) SAT solver
   - use a theory decision procedure to refine the formula and guide the SAT solver

Notable systems: Barcelogic, Boolector, CVC3, MathSAT, Yices, Z3, ...

Lifting SAT Technology to SMT

Two main approaches:

2. “Lazy” [ACG00, dMR02, BDS02, ABC^{+}02]
   - abstract the input formula to a propositional one
   - feed it to a (DPLL-based) SAT solver
   - use a theory decision procedure to refine the formula and guide the SAT solver

Notable systems: Barcelogic, Boolector, CVC3, MathSAT, Yices, Z3, . . .

This talk will focus on the lazy approach

(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: Equality with Uninterpreted Functions

(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

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

- 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 (concretization of) $\{1,\ \overline{2},\ \overline{4}\}$ unsat.

(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 (concretization of) $\{1,\ \overline{2},\ \overline{4}\}$ unsat.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4\}$ to SAT solver.

(Very) Lazy Approach for SMT – Example

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

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.
- SAT solver returns model $\{1,\ \overline{2},\ \overline{4}\}$.
  Theory solver finds (concretization of) $\{1,\ \overline{2},\ \overline{4}\}$ unsat.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4\}$ to SAT solver.
- SAT solver returns model $\{1,\ 3,\ \overline{4}\}$.
  Theory solver finds $\{1,\ 3,\ \overline{4}\}$ unsat.

(Very) Lazy Approach for SMT – Example

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

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.
- SAT solver returns model $\{1,\ \overline{2},\ \overline{4}\}$.
  Theory solver finds (concretization of) $\{1,\ \overline{2},\ \overline{4}\}$ unsat.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4\}$ to SAT solver.
- SAT solver returns model $\{1,\ 3,\ \overline{4}\}$.
  Theory solver finds $\{1,\ 3,\ \overline{4}\}$ unsat.
- 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

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

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

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- Check T-satisfiability only of full propositional model

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- ~~Check T-satisfiability only of full propositional model~~
- Check T-satisfiability of partial assignment M as it grows

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- ~~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

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- ~~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, identify a T-unsatisfiable subset M_0 of M and add $\neg M_0$ as a clause

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- ~~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, 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

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- ~~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, 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, bactrack 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

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

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 [GHN+04]
  - 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 few new lines of code (tens)

An Abstract Framework for Lazy SMT

Several variants and enhancements of lazy SMT solvers exist

They can be modeled abstractly and declaratively as *transition systems*

A transition system is a binary relation over states, induced by a set of conditional transition rules

The framework can be first developed for SAT and then extended to lazy SMT [NOT06, KG07]

Advantages of Abstract Framework

An abstract framework helps one:

- skip over implementation details and unimportant control aspects
- reason formally about solvers for SAT and SMT
- model advanced features such as non-chronological bactracking, lemma learning, theory propagation, ...
- describe different strategies and prove their correctness
- compare different systems at a higher level
- get new insights for further enhancements

Advantages of Abstract Framework

An abstract framework helps one:

- skip over implementation details and unimportant control aspects
- reason formally about solvers for SAT and SMT
- model advanced features such as non-chronological bactracking, lemma learning, theory propagation, ...
- describe different strategies and prove their correctness
- compare different systems at a higher level
- get new insights for further enhancements

The one described next is a re-elaboration of those in [NOT06, KG07]

The Original DPLL Procedure

- Modern SAT solvers are based on the DPLL procedure [DP60, DLL62]
- DPLL 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

States:

$$\text{fail} \quad \text{or} \quad \langle M, F \rangle$$

where

- M is a sequence of literals and *decision points* $\bullet$ denoting a partial truth *assignment*
- F is a set of clauses denoting a CNF *formula*

Def. If $M = M_0 \bullet M_1 \bullet \cdots \bullet M_n$ where each M_i contains no decision points

- M_i is *decision level* i of M
- $M^{[i]} \stackrel{\text{def}}{=} M_0 \bullet \cdots \bullet M_i$

An Abstract Framework for DPLL

States:

fail or $\langle M, F \rangle$

Initial state:

- $\langle (), F_0 \rangle$, where F_0 is to be checked for satisfiability

Expected final states:

- fail if F_0 is unsatisfiable
- $\langle M, G \rangle$ otherwise, where
    - G is equivalent to F_0 and
    - M satisfies G

Transition Rules: Notation

States treated like records:

- M denotes the truth assignment component of current state
- F denotes the formula component of current state

Transition rules in *guarded assignment form* [KG07]

$$\frac{p_1 \quad \cdots \quad p_n}{[\mathsf{M} := e_1] \quad [\mathsf{F} := e_2]}$$

updating M, F or both when premises $p_1, \ldots, p_n$ all hold

NB: When convenient, will treat M as the set of its literals

Transition Rules for the Original DPLL

Extending the assignment

Propagate $$\frac{l_1 \vee \cdots \vee l_n \vee l \in \mathsf{F} \quad \bar{l}_1, \ldots, \bar{l}_n \in \mathsf{M} \quad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\ l}$$

Not. Clauses are treated modulo ACI of $\vee$

Transition Rules for the Original DPLL

Extending the assignment

Propagate $$\frac{l_1 \vee \cdots \vee l_n \vee l \in \mathsf{F} \quad \bar{l}_1, \ldots, \bar{l}_n \in \mathsf{M} \quad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\ l}$$

Not. Clauses are treated modulo ACI of $\vee$

Decide $$\frac{l \in \mathrm{Lit}(\mathsf{F}) \quad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M} \bullet l}$$

Not. $\mathrm{Lit}(F) \stackrel{\text{def}}{=} \{l \mid l \text{ literal of } F\} \cup \{\bar{l} \mid l \text{ literal of } F\}$

Transition Rules for the Original DPLL

Repairing the assignment

Fail $$\frac{l_1 \vee \cdots \vee l_n \in \mathsf{F} \quad \bar{l}_1, \ldots, \bar{l}_n \in \mathsf{M} \quad \bullet \notin \mathsf{M}}{\mathsf{fail}}$$

Transition Rules for the Original DPLL

Repairing the assignment

Fail $$\frac{l_1 \vee \cdots \vee l_n \in \mathsf{F} \quad \bar{l}_1, \ldots, \bar{l}_n \in \mathsf{M} \quad \bullet \notin \mathsf{M}}{\mathsf{fail}}$$

Backtrack

$$\frac{l_1 \vee \cdots \vee l_n \in \mathsf{F} \quad \bar{l}_1, \ldots, \bar{l}_n \in \mathsf{M} \quad \mathsf{M} = M \bullet l\ N \quad \bullet \notin N}{\mathsf{M} := M\ \bar{l}}$$

NB: Last premise of **Backtrack** enforces chronological backtracking

From DPLL to CDCL Solvers (1)

To model conflict-driven backjumping and learning, add to states a third component C whose value is either no or a *conflict clause*

From DPLL to CDCL Solvers (1)

To model conflict-driven backjumping and learning, add to states a third component C whose value is either no or a *conflict clause*

States: fail or $\langle M, F, C\rangle$

Initial state:

- $\langle (), F_0, \text{no}\rangle$, where F_0 is to be checked for satisfiability

Expected final states:

- fail if F_0 is unsatisfiable
- $\langle M, G, \text{no}\rangle$ otherwise, where
  - G is equivalent to F_0 and
  - M satisfies G

From DPLL to CDCL Solvers (2)

Replace **Backtrack** with

From DPLL to CDCL Solvers (2)

Replace **Backtrack** with

Conflict $$\frac{\mathsf{C} = \mathsf{no} \quad l_1 \vee \cdots \vee l_n \in \mathsf{F} \quad \bar{l}_1, \ldots, \bar{l}_n \in \mathsf{M}}{\mathsf{C} := l_1 \vee \cdots \vee l_n}$$

Explain $$\frac{\mathsf{C} = l \vee D \quad l_1 \vee \cdots \vee l_n \vee \bar{l} \in \mathsf{F} \quad \bar{l}_1, \ldots, \bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := l_1 \vee \cdots \vee l_n \vee D}$$

Backjump

$$\frac{\mathsf{C} = l_1 \vee \cdots \vee l_n \vee l \quad \mathsf{lev}\ \bar{l}_1, \ldots, \mathsf{lev}\ \bar{l}_n \leq \ i < \mathsf{lev}\ \bar{l}}{\mathsf{C} := \mathsf{no} \quad \mathsf{M} := \mathsf{M}^{[i]}\ l}$$

Not. $l \prec_{\mathsf{M}} l'$ if l occurs before l' in M

$\mathsf{lev}\ l = i$ iff l occurs in decision level i of M

From DPLL to CDCL Solvers (2)

Replace **Backtrack** with

Conflict $\dfrac{\mathsf{C} = \mathsf{no} \quad l_1 \vee \cdots \vee l_n \in \mathsf{F} \quad \bar{l}_1, \ldots, \bar{l}_n \in \mathsf{M}}{\mathsf{C} := l_1 \vee \cdots \vee l_n}$

Explain $\dfrac{\mathsf{C} = l \vee D \quad l_1 \vee \cdots \vee l_n \vee \bar{l} \in \mathsf{F} \quad \bar{l}_1, \ldots, \bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := l_1 \vee \cdots \vee l_n \vee D}$

Backjump

$$\dfrac{\mathsf{C} = l_1 \vee \cdots \vee l_n \vee l \quad \mathsf{lev}\ \bar{l}_1, \ldots, \mathsf{lev}\ \bar{l}_n \leq \ i < \mathsf{lev}\ \bar{l}}{\mathsf{C} := \mathsf{no} \quad \mathsf{M} := \mathsf{M}^{[i]}\ l}$$

Maintain invariant: $\mathsf{F} \models_{\mathrm{p}} \mathsf{C}$ and $\mathsf{M} \models_{\mathrm{p}} \neg\mathsf{C}$ when $\mathsf{C} \neq \mathsf{no}$

Not. $\models_{\mathrm{p}}$ denotes propositional entailment

From DPLL to CDCL Solvers (3)

Modify **Fail** to

Fail $\dfrac{C \neq no \quad \bullet \notin M}{fail}$

Execution Example

$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$

M	F	C	rule
	F	no	//

Execution Example

$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$

M	F	C	rule
	F	no	//
1	F	no	by **Propagate**

Execution Example

$$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$$

M	F	C	rule
	F	no	//
1	F	no	by **Propagate**
$1\ 2$	F	no	by **Propagate**

Execution Example

$$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$$

M	F	C	rule
	F	no	//
1	F	no	by **Propagate**
$1\ 2$	F	no	by **Propagate**
$1\ 2 \bullet 3$	F	no	by **Decide**

Execution Example

$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$

M	F	C	rule
	F	no	//
1	F	no	by **Propagate**
$1\ 2$	F	no	by **Propagate**
$1\ 2 \bullet 3$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4$	F	no	by **Propagate**

Execution Example

$$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$$

M	F	C	rule
	F	no	//
1	F	no	by **Propagate**
$1\ 2$	F	no	by **Propagate**
$1\ 2 \bullet 3$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5$	F	no	by **Decide**

Execution Example

$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$

M	F	C	rule
	F	no	//
1	F	no	by **Propagate**
$1\ 2$	F	no	by **Propagate**
$1\ 2 \bullet 3$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}$	F	no	by **Propagate**

Execution Example

$$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$$

M	F	C	rule
	F	no	//
1	F	no	by **Propagate**
$1\ 2$	F	no	by **Propagate**
$1\ 2 \bullet 3$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	no	by **Propagate**

Execution Example

$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$

M	F	C	rule
	F	no	$//$
1	F	no	by **Propagate**
$1\ 2$	F	no	by **Propagate**
$1\ 2 \bullet 3$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	$\overline{2} \vee \overline{5} \vee 6 \vee \overline{7}$	by **Conflict**

Execution Example

$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$

M	F	C	rule
	F	no	//
1	F	no	by **Propagate**
$1\ 2$	F	no	by **Propagate**
$1\ 2 \bullet 3$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	$\overline{2} \vee \overline{5} \vee 6 \vee \overline{7}$	by **Conflict**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	$1 \vee \overline{2} \vee \overline{5} \vee 6$	by **Explain** with $\overline{1} \vee \overline{5} \vee 7$

Execution Example

$$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$$

M	F	C	rule
	F	no	//
1	F	no	by **Propagate**
$1\ 2$	F	no	by **Propagate**
$1\ 2 \bullet 3$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	$\overline{2} \vee \overline{5} \vee 6 \vee \overline{7}$	by **Conflict**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	$1 \vee \overline{2} \vee \overline{5} \vee 6$	by **Explain** with $\overline{1} \vee \overline{5} \vee 7$
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	$1 \vee \overline{2} \vee \overline{5}$	by **Explain** with $\overline{5} \vee \overline{6}$

Execution Example

$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$

M	F	C	rule
	F	no	//
1	F	no	by **Propagate**
$1\ 2$	F	no	by **Propagate**
$1\ 2 \bullet 3$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	$\overline{2} \vee \overline{5} \vee 6 \vee \overline{7}$	by **Conflict**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	$1 \vee \overline{2} \vee \overline{5} \vee 6$	by **Explain** with $\overline{1} \vee \overline{5} \vee 7$
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	$1 \vee \overline{2} \vee \overline{5}$	by **Explain** with $\overline{5} \vee \overline{6}$
$1\ 2\ \overline{5}$	F	no	by **Backjump**

Execution Example

$$F := \{1,\ \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ \overline{1} \vee \overline{5} \vee 7,\ \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$$

M	F	C	rule
	F	no	//
1	F	no	by **Propagate**
$1\ 2$	F	no	by **Propagate**
$1\ 2 \bullet 3$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5$	F	no	by **Decide**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	no	by **Propagate**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	$\overline{2} \vee \overline{5} \vee 6 \vee \overline{7}$	by **Conflict**
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	$1 \vee \overline{2} \vee \overline{5} \vee 6$	by **Explain** with $\overline{1} \vee \overline{5} \vee 7$
$1\ 2 \bullet 3\ 4 \bullet 5\ \overline{6}\ 7$	F	$1 \vee \overline{2} \vee \overline{5}$	by **Explain** with $\overline{5} \vee \overline{6}$
$1\ 2\ \overline{5}$	F	no	by **Backjump**
$1\ 2\ \overline{5} \bullet 3$	F	no	by **Decide**
$\ldots$			

From DPLL to CDCL Solvers (4)

Also add

Learn $\dfrac{\mathsf{F} \models_{\mathrm{p}} C \quad C \notin \mathsf{F}}{\mathsf{F} := \mathsf{F} \cup \{C\}}$

Forget $\dfrac{\mathsf{C} = \mathsf{no} \quad \mathsf{F} = G \cup \{C\} \quad G \models_{\mathrm{p}} C}{\mathsf{F} := G}$

Restart $\dfrac{}{\mathsf{M} := \mathsf{M}^{[0]} \quad \mathsf{C} := \mathsf{no}}$

NB: Learn can be applied to any clause stored in C when $\mathsf{C} \neq \mathsf{no}$

Modeling Modern SAT Solvers

At the core, current CDCL SAT solvers are implementations of the transition system with rules

Propagate, **Decide**,

Conflict, **Explain**, **Backjump**,

Learn, **Forget**, **Restart**

Modeling Modern SAT Solvers

At the core, current CDCL SAT solvers are implementations of the transition system with rules

Propagate, **Decide**,

Conflict, **Explain**, **Backjump**,

Learn, **Forget**, **Restart**

Basic DPLL $\stackrel{\text{def}}{=}$

{ **Propagate**, **Decide**, **Conflict**, **Explain**, **Backjump** }

DPLL $\stackrel{\text{def}}{=}$ Basic DPLL + { **Learn**, **Forget**, **Restart** }

The Basic DPLL System – Correctness

Some terminology:

Irreducible state: state to which no Basic DPLL rules apply

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

Exhausted execution: execution ending in an irreducible state

The Basic DPLL System – Correctness

Some terminology:

Irreducible state: state to which no Basic DPLL rules apply

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

Exhausted execution: execution ending in an irreducible state

Proposition (Strong Termination) Every execution in Basic DPLL is finite.

Note: This is not so immediate, because of **Backjump**.

The Basic DPLL System – Correctness

Some terminology:

Irreducible state: state to which no Basic DPLL rules apply

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

Exhausted execution: execution ending in an irreducible state

Proposition (Strong Termination) Every execution in Basic DPLL is finite.

Lemma Every exhausted execution ends with either $C = \text{no}$ or fail.

The Basic DPLL System – Correctness

Some terminology:

Irreducible state: state to which no Basic DPLL rules apply

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

Exhausted execution: execution ending in an irreducible state

Proposition (Soundness) For every exhausted execution starting with $\mathsf{F} = F_0$ and ending with fail, the clause set F_0 is unsatisfiable.

Proposition (Completeness) For every exhausted execution starting with $\mathsf{F} = F_0$ and ending with $\mathsf{C} = \mathsf{no}$, the clause set F_0 is satisfied by M.

The DPLL System – Strategies

- Applying
  - one Basic DPLL rule between each two **Learn** applications and
  - **Restart** less and less often

ensures termination

The DPLL System – Strategies

- A common basic strategy applies the rules with the following priorities:
  1. If $n > 0$ conflicts have been found so far, increase n and apply **Restart**

The DPLL System – Strategies

- A common basic strategy applies the rules with the following priorities:
  1. If $n > 0$ conflicts have been found so far, increase n and apply **Restart**
  2. If a clause is falsified by M, apply **Conflict**

The DPLL System – Strategies

- A common basic strategy applies the rules with the following priorities:
  1. If $n > 0$ conflicts have been found so far, increase n and apply **Restart**
  2. If a clause is falsified by M, apply **Conflict**
  3. Keep applying **Explain** until **Backjump** is applicable

The DPLL System – Strategies

- A common basic strategy applies the rules with the following priorities:
  1. If $n > 0$ conflicts have been found so far, increase n and apply **Restart**
  2. If a clause is falsified by M, apply **Conflict**
  3. Keep applying **Explain** until **Backjump** is applicable
  4. Apply **Learn**

The DPLL System – Strategies

- A common basic strategy applies the rules with the following priorities:
  1. If $n > 0$ conflicts have been found so far, increase n and apply **Restart**
  2. If a clause is falsified by M, apply **Conflict**
  3. Keep applying **Explain** until **Backjump** is applicable
  4. Apply **Learn**
  5. Apply **Backjump**

The DPLL System – Strategies

- A common basic strategy applies the rules with the following priorities:
    1. If $n > 0$ conflicts have been found so far, increase n and apply **Restart**
    2. If a clause is falsified by M, apply **Conflict**
    3. Keep applying **Explain** until **Backjump** is applicable
    4. Apply **Learn**
    5. Apply **Backjump**
    6. Apply **Propagate** to completion

The DPLL System – Strategies

- A common basic strategy applies the rules with the following priorities:
  1. If $n > 0$ conflicts have been found so far, increase n and apply **Restart**
  2. If a clause is falsified by M, apply **Conflict**
  3. Keep applying **Explain** until **Backjump** is applicable
  4. Apply **Learn**
  5. Apply **Backjump**
  6. Apply **Propagate** to completion
  7. Apply **Decide**

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) As before.

Proposition (Completeness) As before.

(For simplicity the statement of the termination result is not entirely accurate.
See [NOT06] for more details.)

From SAT to SMT

Same sort of states and transitions but

- F contains quantifier-free clauses in some theory T
- M is a sequence of theory literals and decision points
- the DPLL system augmented with rules

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

- maintains invariant: F $\models_T$ C and M $\models_{\mathrm{p}}$ $\neg$C when C $\neq$ no

Def. $F \models_T G$ iff every model of T that satisfies F satisfies G as well

SMT-level Rules

Fix a theory T

T-Conflict $$\frac{\mathsf{C} = \mathsf{no} \quad l_1, \ldots, l_n \in \mathsf{M} \quad l_1, \ldots, l_n \models_T \bot}{\mathsf{C} := \bar{l}_1 \vee \cdots \vee \bar{l}_n}$$

Not: $\bot$ = empty clause

NB: $\models_T$ decided by theory solver

SMT-level Rules

Fix a theory T

T-**Conflict** $$\frac{\mathsf{C} = \mathsf{no} \quad l_1, \ldots, l_n \in \mathsf{M} \quad l_1, \ldots, l_n \models_T \bot}{\mathsf{C} := \bar{l}_1 \vee \cdots \vee \bar{l}_n}$$

T-**Propagate** $$\frac{l \in \mathrm{Lit}(\mathsf{F}) \quad \mathsf{M} \models_T l \quad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\ l}$$

Not: $\bot$ = empty clause

NB: $\models_T$ decided by theory solver

SMT-level Rules

Fix a theory T

T-Conflict $$\frac{\mathsf{C} = \mathsf{no} \quad l_1, \ldots, l_n \in \mathsf{M} \quad l_1, \ldots, l_n \models_T \bot}{\mathsf{C} := \bar{l}_1 \vee \cdots \vee \bar{l}_n}$$

T-Propagate $$\frac{l \in \mathrm{Lit}(\mathsf{F}) \quad \mathsf{M} \models_T l \quad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\ l}$$

T-Explain $$\frac{\mathsf{C} = l \vee D \quad \bar{l}_1, \ldots, \bar{l}_n \models_T \bar{l} \quad \bar{l}_1, \ldots, \bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := l_1 \vee \cdots \vee l_n \vee D}$$

Not: $\bot$ = empty clause

NB: $\models_T$ decided by theory solver

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

Modeling the Very Lazy Theory Approach

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

Modeling the Very Lazy Theory Approach

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

M	F	C	rule
	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	//
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by **Propagate**$^{+}$
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by **Decide**
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee 2 \vee 4$	by T-**Conflict**
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	$\overline{1} \vee 2 \vee 4$	by **Learn**
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	no	by **Restart**
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	no	by **Propagate**$^{+}$
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4,\ \overline{1} \vee \overline{3} \vee 4$	$\overline{1} \vee \overline{3} \vee 4$	by T-**Conflict**, **Learn**
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4,\ \overline{1} \vee \overline{3} \vee 4$	no	by **Restart**
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4,\ \overline{1} \vee \overline{3} \vee 4$	$\overline{1} \vee \overline{3} \vee 4$	by **Conflict**
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

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

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

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

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

A Better Lazy Approach

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

M	F	C	rule
	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	//
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by **Propagate**$^{+}$
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by **Decide**
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee 2$	by **T-Conflict**
$1\ \overline{4}\ 2$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by **Backjump**
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by **Propagate**
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee \overline{3} \vee 4$	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**

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**

NB: 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

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 [Tin02]

T-**Propagate** $$\frac{l \in \mathrm{Lit}(\mathsf{F}) \quad \mathsf{M} \models_T l \quad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\ l}$$

T-**Explain** $$\frac{\mathsf{C} = l \vee D \quad \bar{l}_1, \ldots, \bar{l}_n \models_T \bar{l} \quad \bar{l}_1, \ldots, \bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := l_1 \vee \cdots \vee l_n \vee D}$$

Theory Propagation Example

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

Theory Propagation Example

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

M	F	C	rule
	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	//
1	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by **Propagate**
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by **Propagate**
$1\ \overline{4}\ 2$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by T-**Propagate** $(1 \models_T 2)$
$1\ \overline{4}\ 2\ \overline{3}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by T-**Propagate** $(1,\ \overline{4} \models_T \overline{3})$
$1\ \overline{4}\ 2\ \overline{3}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{2} \vee 3$	by **Conflict**
fail			by **Fail**

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

Theory Propagation Example (2)

$$\underbrace{g(a) = e}_{0} \vee \underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

Theory Propagation Example (2)

$$\underbrace{g(a) = e}_{0} \vee \underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

M	F	C	rule
	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	//
$\overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by **Propagate**
$\overline{4} \bullet 1$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by **Decide**
$1\ \overline{4}\ 2$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by T-**Propagate** $(1 \models_T 2)$
$1\ \overline{4}\ 2\ \overline{3}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by T-**Propagate** $(1,\ \overline{4} \models_T \overline{3})$
$\overline{4} \bullet 1\ 2\ \overline{3}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{2} \vee 3$	by **Conflict**
$\overline{4} \bullet 1\ 2\ \overline{3}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee 3$	by T-**Explain**
$\overline{4} \bullet 1\ 2\ \overline{3}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee 4$	by T-**Explain**
$\overline{4}\ \overline{1}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by **Backjump**
$\cdots$			(exercise)

Theory Propagation Features

- With exhaustive theory propagation every assignment M is T-satisfiable (since $M\,l$ is T-unsatisfiable iff $M \models_T \bar{l}$).

Theory Propagation Features

- With exhaustive theory propagation every assignment M is T-satisfiable (since $M\,l$ is T-unsatisfiable iff $M \models_T \bar{l}$).
- For theory propagation to be effective in practice, it needs specialized theory solvers.

Theory Propagation Features

- With exhaustive theory propagation every assignment M is T-satisfiable (since $M\,l$ is T-unsatisfiable iff $M \models_T \bar{l}$).
- For theory propagation to be effective in practice, it needs specialized theory solvers.
- For some theories, e.g., difference logic, detecting T-entailed literals is cheap and so theory propagation is extremely effective.

Theory Propagation Features

- With exhaustive theory propagation every assignment M is T-satisfiable (since $M\, l$ is T-unsatisfiable iff $M \models_T \bar{l}$).
- For theory propagation to be effective in practice, it needs specialized theory solvers.
- For some theories, e.g., difference logic, detecting T-entailed literals is cheap and so theory propagation is extremely effective.
- For others, e.g., the theory of equality, detecting all T-entailed literals is too expensive.

Theory Propagation Features

- With exhaustive theory propagation every assignment M is T-satisfiable (since $M\,l$ is T-unsatisfiable iff $M \models_T \bar{l}$).
- For theory propagation to be effective in practice, it needs specialized theory solvers.
- For some theories, e.g., difference logic, detecting T-entailed literals is cheap and so theory propagation is extremely effective.
- For others, e.g., the theory of equality, detecting all T-entailed literals is too expensive.
- If T-**Propagate** is not applied exhaustively, T-**Conflict** is needed to repair T-unsatisfiable assignments.

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**

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 $\stackrel{\text{def}}{=}$ (1) + (2)

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

Correctness

Updated terminology:

Irreducible state: state to which no Basic DPLL MT rules apply

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

Exhausted execution: execution ending in an irreducible state

Correctness

Updated terminology:

Irreducible state: state to which no Basic DPLL MT rules apply

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

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 $\mathsf{C} = \mathsf{no}$ or fail.

Correctness

Updated terminology:

Irreducible state: state to which no Basic DPLL MT rules apply

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

Exhausted execution: execution ending in an irreducible state

Proposition (Soundness) For every exhausted execution starting with $\mathsf{F} = F_0$ and ending with fail, the clause set F_0 is T-unsatisfiable.

Proposition (Completeness) For every exhausted execution starting with $\mathsf{F} = F_0$ and ending with $\mathsf{C} = \mathsf{no}$, F_0 is T-satisfiable; specifically, M is T-satisfiable and $\mathsf{M} \models_{\mathrm{p}} F_0$.

DPLL(T) Architecture

The approach formalized so far can be implemented with a simple architecture named DPLL(T) [GHN+04, NOT06]

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

DPLL(T) Architecture

The approach formalized so far can be implemented with a simple architecture named DPLL(T) [GHN+04, NOT06]

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

DPLL(X):

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

DPLL(T) Architecture

The approach formalized so far can be implemented with a simple architecture named DPLL(T) [GHN+04, NOT06]

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

T-solver:

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

Reasoning by Cases in Theory Solvers

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

Reasoning by Cases in Theory Solvers

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

Example: $T =$ the theory of arrays.

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

Reasoning by Cases in Theory Solvers

For certain theories, determining that a set 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.

Reasoning by Cases in Theory Solvers

For certain theories, determining that a set 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.

Reasoning by Cases in Theory Solvers

For certain theories, determining that a set 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

A *complete* T-solver reasons by cases via (internal) case splitting and backtracking mechanisms.

Case Splitting

A *complete* T-solver reasons by cases via (internal) case splitting and backtracking mechanisms.

An alternative is to lift case splitting and backtracking from the T-solver to the SAT engine.

Case Splitting

A *complete* T-solver reasons by cases via (internal) case splitting and backtracking mechanisms.

An alternative is to lift case splitting and backtracking from the T-solver to the SAT engine.

Basic idea: encode case splits as sets of clauses and send them as needed to the SAT engine for it to split on them.

Case Splitting

A *complete* T-solver reasons by cases via (internal) case splitting and backtracking mechanisms.

An alternative is to lift case splitting and backtracking from the T-solver to the SAT engine.

Basic idea: encode case splits as sets of clauses and send them as needed to the SAT engine for it to split on them.

Possible benefits:

- All case-splitting is coordinated by the SAT engine
- Only have to implement case-splitting infrastructure in one place
- Can learn a wider class of lemmas

Splitting on Demand [BNOT06]

Basic idea: encode case splits as a set of clauses and send them as needed to the SAT engine for it to split on them

Splitting on Demand [BNOT06]

Basic idea: encode case splits as a set of clauses and send them as needed to the SAT engine for it to split on them

Basic Scenario:

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

Splitting on Demand [BNOT06]

Basic idea: encode case splits as a set of clauses and send them as needed to the SAT engine for it to split on them

Basic Scenario:

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

- Main SMT module: "Is M T-unsatisfiable?"

Splitting on Demand [BNOT06]

Basic idea: encode case splits as a set of clauses and send them as needed to the SAT engine for it to split on them

Basic Scenario:

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

- Main SMT module: “Is M T-unsatisfiable?”
- T-solver: “I do not know yet, but it will help me if you consider these *theory lemmas*:

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

Modeling Splitting on Demand

To model the generation of theory lemmas for case splits, add the rule

T-**Learn**

$$\frac{\models_T \exists \mathbf{v}(l_1 \vee \cdots \vee l_n) \quad l_1, \ldots, l_n \in L_{\mathrm{S}} \quad \mathbf{v} \text{ vars not in } \mathsf{F}}{\mathsf{F} := \mathsf{F} \cup \{l_1 \vee \cdots \vee l_n\}}$$

where L_{S} is a finite set of literals dependent on the initial set of clauses (see [BNOT06] for a formal definition of L_{S})

Modeling Splitting on Demand

To model the generation of theory lemmas for case splits, add the rule

T-Learn

$$\frac{\models_T \exists \mathbf{v}(l_1 \vee \cdots \vee l_n) \quad l_1, \ldots, l_n \in L_{\mathrm{S}} \quad \mathbf{v} \text{ vars not in F}}{\mathsf{F} := \mathsf{F} \cup \{l_1 \vee \cdots \vee l_n\}}$$

where L_{S} is a finite set of literals dependent on the initial set of clauses (see [BNOT06] for a formal definition of L_{S})

NB: For many theories with a theory solver, there exists an appropriate finite L_{S} for every input F

The set L_{S} does not need to be computed explicitly

Modeling Splitting on Demand

Now we can relax the requirement on the theory solver:

When $M \models_T F$, it must either

- determine whether $M \models_T \perp$ or
- generate a new clause by T-**Learn** containing at least one literal of L_S undefined in M

Modeling Splitting on Demand

Now we can relax the requirement on the theory solver:

When $\mathsf{M} \models_T \mathsf{F}$, it must either

- determine whether $\mathsf{M} \models_T \bot$ or
- generate a new clause by T-**Learn** containing at least one literal of L_{S} undefined in M

The T-solver is required to determine whether $\mathsf{M} \models_T \bot$ only if all literals in L_{S} are defined in M

Modeling Splitting on Demand

Now we can relax the requirement on the theory solver:

When $\mathsf{M} \models_T \mathsf{F}$, it must either

- determine whether $\mathsf{M} \models_T \bot$ or
- generate a new clause by T-**Learn** containing at least one literal of L_{S} undefined in M

The T-solver is required to determine whether $\mathsf{M} \models_T \bot$ only if all literals in L_{S} are defined in M

NB: In practice, to determine if $\mathsf{M} \models_T \bot$ the T-solver only needs a small subset of L_{S} to be defined in M

Example — Theory of Finite Sets

$$F: \quad x = y \cup z \;\;\wedge\;\; y \neq \emptyset \vee x \neq z$$

M	F	rule
$x = y \cup z$	F	by **Propagate**$^+$
$x = y \cup z \;\bullet\; y = \emptyset$	F	by **Decide**
$x = y \cup z \;\bullet\; y = \emptyset\; x \neq z$	F	by **Propagate**
$x = y \cup z \;\bullet\; y = \emptyset\; x \neq z$	$F, (x = z \vee e \in x \vee e \in z),$ $(x = z \vee e \notin x \vee e \notin z)$	by T-**Learn**
$x = y \cup z \;\bullet\; y = \emptyset\; x \neq z \;\bullet\; e \in x$	$F, (x = z \vee e \in x \vee e \in z),$ $(x = z \vee e \notin x \vee e \notin z)$	by **Decide**
$x = y \cup z \;\bullet\; y = \emptyset\; x \neq z \;\bullet\; e \in x\; e \notin z$	$F, (x = z \vee e \in x \vee e \in z),$ $(x = z \vee e \notin x \vee e \notin z)$	by **Propagate**

T-solver can make the following deductions at this point:

$$e \in x \quad \cdots \quad \Rightarrow \quad e \in y \cup z \quad \cdots \quad \Rightarrow \quad e \in y \quad \cdots \quad \Rightarrow \quad e \in \emptyset \quad \Rightarrow \quad \bot$$

This enables an application of T-**Conflict** with clause

$$x \neq y \cup z \;\vee\; y \neq \emptyset \;\vee\; x = z \;\vee\; e \notin x \;\vee\; e \in z$$

Correctness Results

Correctness results can be extended to the new rule.

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

Completeness: As long as the theory solver can decide $\mathsf{M} \models_T \bot$ when all literals in L_{S} are determined, the system is still complete.

Termination: The system terminates under the same conditions as before. Roughly:

- Any lemma is (re)learned only finitely many times
- **Restart** is applied with increased periodicity

Part II

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

Need for Combining Theories and Solvers

Recall: Many applications give rise to formulas like:

$$a \approx b + 2 \wedge A \approx \mathrm{write}(B, a + 1, 4) \wedge$$
$$(\mathrm{read}(A, b + 3) \approx 2 \vee f(a - 1) \neq f(b + 1))$$

Need for Combining Theories and Solvers

Recall: Many applications give rise to formulas like:

$$a \approx b + 2 \;\wedge\; A \approx \mathrm{write}(B, a+1, 4) \;\wedge$$
$$(\mathrm{read}(A, b+3) \approx 2 \;\vee\; f(a-1) \neq f(b+1))$$

Solving that formula requires reasoning over

- the theory of linear arithmetic (T_{LA})
- the theory of arrays (T_{A})
- the theory of uninterpreted functions (T_{EUF})

Need for Combining Theories and Solvers

Recall: Many applications give rise to formulas like:

$$a \approx b + 2 \wedge A \approx \mathrm{write}(B, a + 1, 4) \wedge$$
$$(\mathrm{read}(A, b + 3) \approx 2 \vee f(a - 1) \neq f(b + 1))$$

Solving that formula requires reasoning over

- the theory of linear arithmetic (T_{LA})
- the theory of arrays (T_{A})
- the theory of uninterpreted functions (T_{EUF})

Question: Given solvers for each theory, can we combine them modularly into one for $T_{\mathrm{LA}} \cup T_{\mathrm{A}} \cup T_{\mathrm{EUF}}$?

Need for Combining Theories and Solvers

Recall: Many applications give rise to formulas like:

$$a \approx b + 2 \land A \approx \mathrm{write}(B, a + 1, 4) \land$$
$$(\mathrm{read}(A, b + 3) \approx 2 \lor f(a - 1) \neq f(b + 1))$$

Solving that formula requires reasoning over

- the theory of linear arithmetic (T_{LA})
- the theory of arrays (T_{A})
- the theory of uninterpreted functions (T_{EUF})

Question: Given solvers for each theory, can we combine them modularly into one for $T_{\mathrm{LA}} \cup T_{\mathrm{A}} \cup T_{\mathrm{EUF}}$?

Under certain conditions, we can do it with the Nelson-Oppen combination method [NO79, Opp80]

Motivating Example (Convex Case)

Consider the following set of literals over $T_{\mathrm{LRA}} \cup T_{\mathrm{EUF}}$ (T_{LRA}, linear real arithmetic):

$$
\begin{array}{rcl}
f(f(x) - f(y)) & = & a \\
f(0) & > & a + 2 \\
x & = & y
\end{array}
$$

Motivating Example (Convex Case)

Consider the following set of literals over $T_{\mathrm{LRA}} \cup T_{\mathrm{EUF}}$:

$$\begin{array}{rcl} f(f(x) - f(y)) & = & a \\ f(0) & > & a + 2 \\ x & = & y \end{array}$$

First step: *purify* literals so that each belongs to a single theory

Motivating Example (Convex Case)

Consider the following set of literals over $T_{\text{LRA}} \cup T_{\text{EUF}}$:

$$
\begin{aligned}
f(f(x) - f(y)) &= a \\
f(0) &> a + 2 \\
x &= y
\end{aligned}
$$

First step: *purify* literals so that each belongs to a single theory

$$
f(f(x) - f(y)) = a \quad \Longrightarrow \quad \begin{aligned} f(e_1) &= a \\ e_1 &= f(x) - f(y) \end{aligned} \quad \Longrightarrow \quad \begin{aligned} f(e_1) &= a \\ e_1 &= e_2 - e_3 \\ e_2 &= f(x) \\ e_3 &= f(y) \end{aligned}
$$

Motivating Example (Convex Case)

Consider the following set of literals over $T_{\mathrm{LRA}} \cup T_{\mathrm{EUF}}$:

$$\begin{array}{rcl} f(f(x) - f(y)) & = & a \\ f(0) & > & a + 2 \\ x & = & y \end{array}$$

First step: *purify* literals so that each belongs to a single theory

$$\begin{array}{ccccc} f(0) = a + 2 & \Longrightarrow & f(e_4) = a + 2 & \Longrightarrow & f(e_4) = e_5 \\ & & e_4 = 0 & & e_4 = 0 \\ & & & & e_5 > a + 2 \end{array}$$

Motivating Example (Convex Case)

Second step: exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

L_1	L_2
$f(e_1) = a$	$e_2 - e_3 = e_1$
$f(x) = e_2$	$e_4 = 0$
$f(y) = e_3$	$e_5 > a + 2$
$f(e_4) = e_5$	
$x = y$	

Motivating Example (Convex Case)

Second step: exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

L_1	L_2
$f(e_1) = a$	$e_2 - e_3 = e_1$
$f(x) = e_2$	$e_4 = 0$
$f(y) = e_3$	$e_5 > a + 2$
$f(e_4) = e_5$	
$x = y$	

$$L_1 \models_{\mathrm{EUF}} e_2 = e_3$$

Motivating Example (Convex Case)

Second step: exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

L_1	L_2
$f(e_1) = a$	$e_2 - e_3 = e_1$
$f(x) = e_2$	$e_4 = 0$
$f(y) = e_3$	$e_5 > a + 2$
$f(e_4) = e_5$	$e_2 = e_3$
$x = y$	

Motivating Example (Convex Case)

Second step: exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

L_1	L_2
$f(e_1) = a$	$e_2 - e_3 = e_1$
$f(x) = e_2$	$e_4 = 0$
$f(y) = e_3$	$e_5 > a + 2$
$f(e_4) = e_5$	$e_2 = e_3$
$x = y$	

$$L_2 \models_{\mathrm{LRA}} e_1 = e_4$$

Motivating Example (Convex Case)

Second step: exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

L_1	L_2
$f(e_1) = a$	$e_2 - e_3 = e_1$
$f(x) = e_2$	$e_4 = 0$
$f(y) = e_3$	$e_5 > a + 2$
$f(e_4) = e_5$	$e_2 = e_3$
$x = y$	
$e_1 = e_4$	

Motivating Example (Convex Case)

Second step: exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

L_1	L_2
$f(e_1) = a$	$e_2 - e_3 = e_1$
$f(x) = e_2$	$e_4 = 0$
$f(y) = e_3$	$e_5 > a + 2$
$f(e_4) = e_5$	$e_2 = e_3$
$x = y$	
$e_1 = e_4$	

$$L_1 \models_{\mathrm{EUF}} a = e_5$$

Motivating Example (Convex Case)

Second step: exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

L_1	L_2
$f(e_1) = a$	$e_2 - e_3 = e_1$
$f(x) = e_2$	$e_4 = 0$
$f(y) = e_3$	$e_5 > a + 2$
$f(e_4) = e_5$	$e_2 = e_3$
$x = y$	$a = e_5$
$e_1 = e_4$	

Motivating Example (Convex Case)

Second step: exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

L_1	L_2
$f(e_1) = a$	$e_2 - e_3 = e_1$
$f(x) = e_2$	$e_4 = 0$
$f(y) = e_3$	$e_5 > a + 2$
$f(e_4) = e_5$	$e_2 = e_3$
$x = y$	$a = e_5$
$e_1 = e_4$	

Third step: check for satisfiability locally

$L_1 \not\models_{\mathrm{EUF}} \bot$

$L_2 \models_{\mathrm{LRA}} \bot$

Motivating Example (Convex Case)

Second step: exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

L_1	L_2
$f(e_1) = a$	$e_2 - e_3 = e_1$
$f(x) = e_2$	$e_4 = 0$
$f(y) = e_3$	$e_5 > a + 2$
$f(e_4) = e_5$	$e_2 = e_3$
$x = y$	$a = e_5$
$e_1 = e_4$	

Third step: check for satisfiability locally

$L_1 \not\models_{\mathrm{EUF}} \bot$

$L_2 \models_{\mathrm{LRA}} \bot$

Report unsatisfiable

Motivating Example (Non-convex Case)

Consider the following unsatisfiable set of literals over $T_{\mathrm{LIA}} \cup T_{\mathrm{EUF}}$ (T_{LIA}, linear integer arithmetic) :

$$
\begin{array}{rcl}
1 \leq & x & \leq 2 \\
f(1) & = & a \\
f(x) & = & b \\
a & = & b + 2
\end{array}
$$

Motivating Example (Non-convex Case)

Consider the following unsatisfiable set of literals over $T_{\mathrm{LIA}} \cup T_{\mathrm{EUF}}$:

$$\begin{array}{rcl} 1 \leq & x & \leq 2 \\ f(1) & = & a \\ f(x) & = & b \\ a & = & b + 2 \end{array}$$

First step: *purify* literals so that each belongs to a single theory

Motivating Example (Non-convex Case)

Consider the following unsatisfiable set of literals over $T_{\mathrm{LIA}} \cup T_{\mathrm{EUF}}$:

$$\begin{array}{rcl} 1 \leq & x & \leq 2 \\ f(1) & = & a \\ f(x) & = & b \\ a & = & b+2 \end{array}$$

First step: *purify* literals so that each belongs to a single theory

$$f(1) = a \quad \Longrightarrow \quad \begin{array}{r} f(e_1) = a \\ e_1 = 1 \end{array}$$

Motivating Example (Non-convex Case)

Consider the following unsatisfiable set of literals over $T_{\mathrm{LIA}} \cup T_{\mathrm{EUF}}$:

$$\begin{array}{rcl} 1 \leq & x & \leq 2 \\ f(1) & = & a \\ f(x) & = & b \\ a & = & b+2 \end{array}$$

First step: *purify* literals so that each belongs to a single theory

$$f(2) = f(1) + 3 \implies \begin{array}{r} e_2 = 2 \\ f(e_2) = e_3 \\ f(e_1) = e_4 \\ e_3 = e_4 + 3 \end{array}$$

Motivating Example (Non-convex Case)

Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

L_1	L_2
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b + 2$	$f(e_1) = e_4$
$e_2 = 2$	
$e_3 = e_4 + 3$	
$a = e_4$	

Motivating Example (Non-convex Case)

Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

L_1	L_2
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b + 2$	$f(e_1) = e_4$
$e_2 = 2$	
$e_3 = e_4 + 3$	
$a = e_4$	

No more entailed equalities, but $L_1 \models_{\text{LIA}} x = e_1 \vee x = e_2$

Motivating Example (Non-convex Case)

Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

L_1	L_2
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b + 2$	$f(e_1) = e_4$
$e_2 = 2$	
$e_3 = e_4 + 3$	
$a = e_4$	

Consider each case of $x = e_1 \vee x = e_2$ separately

Motivating Example (Non-convex Case)

Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

L_1	L_2
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b + 2$	$f(e_1) = e_4$
$e_2 = 2$	
$e_3 = e_4 + 3$	
$a = e_4$	

Case 1) $x = e_1$

Motivating Example (Non-convex Case)

Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

L_1	L_2
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b + 2$	$f(e_1) = e_4$
$e_2 = 2$	$x = e_1$
$e_3 = e_4 + 3$	
$a = e_4$	
$x = e_1$	

Motivating Example (Non-convex Case)

Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

L_1	L_2
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b + 2$	$f(e_1) = e_4$
$e_2 = 2$	$x = e_1$
$e_3 = e_4 + 3$	
$a = e_4$	
$x = e_1$	

$L_2 \models_{\mathrm{EUF}} a = b$, which entails $\perp$ when sent to L_1

Motivating Example (Non-convex Case)

Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

L_1	L_2
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b + 2$	$f(e_1) = e_4$
$e_2 = 2$	
$e_3 = e_4 + 3$	
$a = e_4$	

Motivating Example (Non-convex Case)

Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

L_1	L_2
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b + 2$	$f(e_1) = e_4$
$e_2 = 2$	
$e_3 = e_4 + 3$	
$a = e_4$	

Case 2) $x = e_2$

Motivating Example (Non-convex Case)

Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

L_1	L_2
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b + 2$	$f(e_1) = e_4$
$e_2 = 2$	$x = e_2$
$e_3 = e_4 + 3$	
$a = e_4$	
$x = e_2$	

Motivating Example (Non-convex Case)

Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

L_1	L_2
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b + 2$	$f(e_1) = e_4$
$e_2 = 2$	$x = e_2$
$e_3 = e_4 + 3$	
$a = e_4$	
$x = e_2$	

$L_2 \models_{\mathrm{EUF}} e_3 = b$, which entails $\perp$ when sent to L_1

The Nelson-Oppen Method

- For $i = 1, 2$, let T_i be a first-order theory of *signature* Σ_i
  (set of function and predicate symbols in T_i other than $=$)
- Let $T = T_1 \cup T_2$
- Let $\mathcal{C}$ be a finite set of *free* constants (i.e., not in $\Sigma_1 \cup \Sigma_2$)

The Nelson-Oppen Method

- For $i = 1, 2$, let T_i be a first-order theory of *signature* Σ_i
  (set of function and predicate symbols in T_i other than $=$)
- Let $T = T_1 \cup T_2$
- Let $\mathcal{C}$ be a finite set of *free* constants (i.e., not in $\Sigma_1 \cup \Sigma_2$)

We consider only input problems of the form

$$L_1 \cup L_2$$

where each L_i is a finite set of *ground* (i.e., variable-free) $(\Sigma_i \cup \mathcal{C})$-literals

The Nelson-Oppen Method

- For $i = 1, 2$, let T_i be a first-order theory of *signature* Σ_i (set of function and predicate symbols in T_i other than $=$)
- Let $T = T_1 \cup T_2$
- Let $\mathcal{C}$ be a finite set of *free* constants (i.e., not in $\Sigma_1 \cup \Sigma_2$)

We consider only input problems of the form

$$L_1 \cup L_2$$

where each L_i is a finite set of *ground* (i.e., variable-free) $(\Sigma_i \cup \mathcal{C})$-literals

NB: Because of purification, there is no loss of generality in considering only ground $(\Sigma_i \cup \mathcal{C})$-literals

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:	$L_1 \cup L_2$ with L_i finite set of ground $(\Sigma_i \cup \mathcal{C})$-literals
Output:	**sat** or **unsat**

The Nelson-Oppen Method

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

Input: $L_1 \cup L_2$ with L_i finite set of ground $(\Sigma_i \cup \mathcal{C})$-literals
Output: **sat** or **unsat**

1. Guess an *arrangement* A, i.e., a set of equalities and disequalities over $\mathcal{C}$ such that

$$c = d \in A \quad \text{or} \quad c \neq d \in A \quad \text{for all } c, d \in \mathcal{C}$$

The Nelson-Oppen Method

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

Input: $L_1 \cup L_2$ with L_i finite set of ground $(\Sigma_i \cup \mathcal{C})$-literals
Output: **sat** or **unsat**

1. Guess an *arrangement* A, i.e., a set of equalities and disequalities over $\mathcal{C}$ such that

$$c = d \in A \quad \text{or} \quad c \neq d \in A \quad \text{for all } c, d \in \mathcal{C}$$

2. If $L_i \cup A$ 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: $L_1 \cup L_2$ with L_i finite set of ground $(\Sigma_i \cup \mathcal{C})$-literals
Output: **sat** or **unsat**

1. Guess an *arrangement* A, i.e., a set of equalities and disequalities over $\mathcal{C}$ such that

$$c = d \in A \quad \text{or} \quad c \neq d \in A \quad \text{for all } c, d \in \mathcal{C}$$

2. If $L_i \cup A$ is T_i-unsatisfiable for $i = 1$ or $i = 2$, return **unsat**
3. Otherwise, return **sat**

Correctness of the NO Method

Proposition (Termination) The method is terminating.

(Trivially, because there is only a finite number of arrangements to guess)

Correctness of the NO Method

Proposition (Termination) The method is terminating.

(Trivially, because there is only a finite number of arrangements to guess)

Proposition (Soundness) If the method returns **unsat** for every arrangement, the input is $(T_1 \cup T_2)$-unsatisfiable.

(Because satisfiability in $(T_1 \cup T_2)$ is always preserved)

Correctness of the NO Method

Proposition (Termination) The method is terminating.

(Trivially, because there is only a finite number of arrangements to guess)

Proposition (Soundness) If the method returns **unsat** for every arrangement, the input is $(T_1 \cup T_2)$-unsatisfiable.

(Because satisfiability in $(T_1 \cup T_2)$ is always preserved)

Proposition (Completeness) If $\Sigma_1 \cap \Sigma_2 = \emptyset$ and T_1 and T_2 are stably infinite, when the method returns **sat** for some arrangement, the input is $(T_1 \cup T_2)$-is satisfiable.

(Only non-immediate aspect)

Stably Infinite Theories

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

Stably Infinite Theories

Def. A 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 (e.g., integer arithmetic)
- Complete theories with an infinite model (e.g., theory of dense linear orders, theory of lists)
- Convex theories (e.g., EUF, linear real arithmetic)

Stably Infinite Theories

Def. A 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 (e.g., integer arithmetic)
- Complete theories with an infinite model (e.g., theory of dense linear orders, theory of lists)
- Convex theories (e.g., EUF, linear real arithmetic)

Def. A theory T is *convex* iff, for any set L of literals
$L \models_T s_1 = t_1 \vee \cdots \vee s_n = t_n \implies L \models_T s_i = t_i$ for some i

NB: With convex theories, arrangements do not need to be guessed—they can be computed by (theory) propagation

Stably Infinite Theories

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

Stably Infinite Theories

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

Other interesting theories are not stably infinite:

- Theories of a finite structure (e.g., theory of bit vectors of finite size, arithmetic modulo n)
- Theories with models of bounded cardinality (e.g., theory of strings of bounded length)
- Some equational/Horn theories

Stably Infinite Theories

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

Other interesting theories are not stably infinite:

- Theories of a finite structure (e.g., theory of bit vectors of finite size, arithmetic modulo n)
- Theories with models of bounded cardinality (e.g., theory of strings of bounded length)
- Some equational/Horn theories

The Nelson-Oppen method has been extended to some classes of non-stably infinite theories [TZ05, RRZ05, JB10]

SMT Solving with Multiple Theories

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

How can we integrate all of them cooperatively into a single SMT solver for $T = T_1 \cup \cdots \cup T_n$?

SMT Solving with Multiple Theories

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

How can we integrate all of them cooperatively into a single SMT solver for $T = T_1 \cup \cdots \cup T_n$?

Quick Solution:

1. Combine $S_1, \ldots, S_n$ with Nelson-Oppen into a theory solver for T
2. Build a DPLL(T) solver as usual

SMT Solving with Multiple Theories

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

How can we integrate all of them cooperatively into a single SMT solver for $T = T_1 \cup \cdots \cup T_n$?

Better Solution [Bar02, BBC+05b, BNOT06]:

1. Extend DPLL(T) to DPLL($T_1, \ldots, T_n$)
2. Lift Nelson-Oppen to the DPLL($X_1, \ldots, X_n$) level
3. Build a DPLL($T_1, \ldots, T_n$) solver

Modeling DPLL($T_1, \ldots, T_n$) Abstractly

- 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 $\mathcal{C}$ be a set of free constants
- Assume wlog that each input literal has signature $(\Sigma_1 \cup \mathcal{C})$ or $(\Sigma_2 \cup \mathcal{C})$ (no *mixed* literals)
- Let $\mathsf{M}|_i \stackrel{\text{def}}{=} \{(\Sigma_i \cup \mathcal{C})\text{-literals of M and their complement}\}$
- Let $\mathrm{I}(\mathsf{M}) \stackrel{\text{def}}{=} \{c = d \mid c, d \text{ occur in } \mathcal{C}, \mathsf{M}|_1 \text{ and } \mathsf{M}|_2\} \cup$
  $\{c \neq d \mid c, d \text{ occur in } \mathcal{C}, \mathsf{M}|_1 \text{ and } \mathsf{M}|_2\}$
  (*interface literals*)

Abstract DPLL Modulo Multiple Theories

Propagate, **Conflict**, **Explain**, **Backjump**, **Fail** (unchanged)

Abstract DPLL Modulo Multiple Theories

Propagate, **Conflict**, **Explain**, **Backjump**, **Fail** (unchanged)

Decide $\dfrac{l \in \mathrm{Lit}(\mathsf{F}) \cup \mathrm{I}(\mathsf{M}) \quad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M} \bullet l}$

Only change: decide on interface equalities as well

Abstract DPLL Modulo Multiple Theories

Propagate, **Conflict**, **Explain**, **Backjump**, **Fail** (unchanged)

Decide $\dfrac{l \in \mathrm{Lit}(\mathsf{F}) \cup \mathrm{I}(\mathsf{M}) \quad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M} \bullet l}$

Only change: decide on interface equalities as well

T-**Propagate**

$$\dfrac{l \in \mathrm{Lit}(\mathsf{F}) \cup \mathrm{I}(\mathsf{M}) \quad i \in \{1, 2\} \quad \mathsf{M} \models_{T_i} l \quad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\ l}$$

Only change: propagate interface equalities as well, but reason locally in each T_i

Abstract DPLL Modulo Multiple Theories

T-Conflict

$$\frac{\mathsf{C} = \mathsf{no} \quad l_1, \ldots, l_n \in \mathsf{M} \quad l_1, \ldots, l_n \models_{T_i} \perp \quad i \in \{1, 2\}}{\mathsf{C} := \bar{l}_1 \vee \cdots \vee \bar{l}_n}$$

T-Explain

$$\frac{\mathsf{C} = l \vee D \quad \bar{l}_1, \ldots, \bar{l}_n \models_{T_i} \bar{l} \quad i \in \{1, 2\} \quad \bar{l}_1, \ldots, \bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := l_1 \vee \cdots \vee l_n \vee D}$$

Only change: reason locally in each T_i

Abstract DPLL Modulo Multiple Theories

T-Conflict

$$\frac{\mathsf{C} = \mathsf{no} \quad l_1, \ldots, l_n \in \mathsf{M} \quad l_1, \ldots, l_n \models_{T_i} \bot \quad i \in \{1,2\}}{\mathsf{C} := \bar{l}_1 \vee \cdots \vee \bar{l}_n}$$

T-Explain

$$\frac{\mathsf{C} = l \vee D \quad \bar{l}_1, \ldots, \bar{l}_n \models_{T_i} \bar{l} \quad i \in \{1,2\} \quad \bar{l}_1, \ldots, \bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := l_1 \vee \cdots \vee l_n \vee D}$$

Only change: reason locally in each T_i

I-Learn

$$\frac{\models_{T_i} l_1 \vee \cdots \vee l_n \quad l_1, \ldots, l_n \in \mathsf{M}|_i \cup \mathrm{I}(\mathsf{M}) \quad i \in \{1,2\}}{\mathsf{F} := \mathsf{F} \cup \{l_1 \vee \cdots \vee l_n\}}$$

New rule: for entailed disjunctions of interface literals

Example — Convex Theories

$$F := \overbrace{f(e_1) = a}^{0} \wedge \overbrace{f(x) = e_2}^{1} \wedge \overbrace{f(y) = e_3}^{2} \wedge \overbrace{f(e_4) = e_5}^{3} \wedge \overbrace{x = y}^{4} \wedge$$

$$\underbrace{e_2 - e_3 = e_1}_{5} \wedge \underbrace{e_4 = 0}_{6} \wedge \underbrace{e_5 > a + 2}_{7}$$

$$\underbrace{e_2 = e_3}_{8} \quad \underbrace{e_1 = e_4}_{9} \quad \underbrace{a = e_5}_{10}$$

Example — Convex Theories

$$F := \overbrace{f(e_1) = a}^{0} \wedge \overbrace{f(x) = e_2}^{1} \wedge \overbrace{f(y) = e_3}^{2} \wedge \overbrace{f(e_4) = e_5}^{3} \wedge \overbrace{x = y}^{4} \wedge$$
$$\underbrace{e_2 - e_3 = e_1}_{5} \wedge \underbrace{e_4 = 0}_{6} \wedge \underbrace{e_5 > a + 2}_{7}$$
$$\underbrace{e_2 = e_3}_{8} \quad \underbrace{e_1 = e_4}_{9} \quad \underbrace{a = e_5}_{10}$$

M	F	C	rule
	F	no	//
0 1 2 3 4 5 6 7	F	no	by **Propagate**$^+$
0 1 2 3 4 5 6 7 8	F	no	by T-**Propagate** $(1, 2, 4 \models_{\mathrm{EUF}} 8)$
0 1 2 3 4 5 6 7 8 9	F	no	by T-**Propagate** $(5, 6, 8 \models_{\mathrm{LRA}} 9)$
0 1 2 3 4 5 6 7 8 9 10	F	no	by T-**Propagate** $(0, 3, 9 \models_{\mathrm{EUF}} 10)$
0 1 2 3 4 5 6 7 8 9 10	F	$\overline{7} \vee \overline{10}$	by T-**Conflict** $(7, 10 \models_{\mathrm{LRA}} \perp)$
fail			by **Fail**

Example — Non-convex Theories

$$F := \overbrace{f(e_1) = a}^{0} \wedge \overbrace{f(x) = b}^{1} \wedge \overbrace{f(e_2) = e_3}^{2} \wedge \overbrace{f(e_1) = e_4}^{3} \wedge$$

$$\underbrace{1 \leq x}_{4} \wedge \underbrace{x \leq 2}_{5} \wedge \underbrace{e_1 = 1}_{6} \wedge \underbrace{a = b + 2}_{7} \wedge \underbrace{e_2 = 2}_{8} \wedge \underbrace{e_3 = e_4 + 3}_{9}$$

$$\underbrace{a = e_4}_{10} \quad \underbrace{x = e_1}_{11} \quad \underbrace{x = e_2}_{12} \quad \underbrace{a = b}_{13}$$

Example — Non-convex Theories

$$F := \overbrace{f(e_1) = a}^{0} \land \overbrace{f(x) = b}^{1} \land \overbrace{f(e_2) = e_3}^{2} \land \overbrace{f(e_1) = e_4}^{3} \land$$
$$\underbrace{1 \le x}_{4} \land \underbrace{x \le 2}_{5} \land \underbrace{e_1 = 1}_{6} \land \underbrace{a = b + 2}_{7} \land \underbrace{e_2 = 2}_{8} \land \underbrace{e_3 = e_4 + 3}_{9}$$
$$\underbrace{a = e_4}_{10} \quad \underbrace{x = e_1}_{11} \quad \underbrace{x = e_2}_{12} \quad \underbrace{a = b}_{13}$$

M	F	C	rule
	F	no	//
$0 \cdots 9$	F	no	by **Propagate**$^+$
$0 \cdots 9\ 10$	F	no	by T-**Propagate** $(0, 3 \models_{\mathrm{EUF}} 10)$
$0 \cdots 9\ 10$	$F,\ \overline{4} \lor \overline{5} \lor 11 \lor 12$	no	by **I-Learn** $(\models_{\mathrm{LIA}} \overline{4} \lor \overline{5} \lor 11 \lor 12)$
$0 \cdots 9\ 10 \bullet 11$	$F,\ \overline{4} \lor \overline{5} \lor 11 \lor 12$	no	by **Decide**
$0 \cdots 9\ 10 \bullet 11\ 13$	$F,\ \overline{4} \lor \overline{5} \lor 11 \lor 12$	no	by T-**Propagate** $(0, 1, 11 \models_{\mathrm{EUF}} 13)$
$0 \cdots 9\ 10 \bullet 11\ 13$	$F,\ \overline{4} \lor \overline{5} \lor 11 \lor 12$	$\overline{7} \lor \overline{13}$	by T-**Conflict** $(7, 13 \models_{\mathrm{EUF}} \perp)$
$0 \cdots 9\ 10\ \overline{13}$	$F,\ \overline{4} \lor \overline{5} \lor 11 \lor 12$	no	by **Backjump**
$0 \cdots 9\ 10\ \overline{13}\ \overline{11}$	$F,\ \overline{4} \lor \overline{5} \lor 11 \lor 12$	no	by T-**Propagate** $(0, 1, \overline{13} \models_{\mathrm{EUF}} \overline{11})$
$0 \cdots 9\ 10\ \overline{13}\ \overline{11}\ 12$	$F,\ \overline{4} \lor \overline{5} \lor 11 \lor 12$	no	by **Propagate**
$\cdots$			(exercise)
fail	$\cdots$	$\cdots$	by **Fail**

Suggested Readings

1. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. **Solving SAT and SAT Modulo Theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T)**. Journal of the ACM, 53(6):937-977, 2006.

2. R. Sebastiani. **Lazy Satisfiability Modulo Theories**. Journal on Satisfiability, Boolean Modeling and Computation 3:141-224, 2007.

3. S. Krstić and A. Goel. **Architecting Solvers for SAT Modulo Theories: Nelson-Oppen with DPLL**. In Proceeding of the Symposium on Frontiers of Combining Systems (FroCoS'07). Volume 4720 of LNCS. Springer, 2007.

4. C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. **Satisfiability Modulo Theories**. In Handbook of Satisfiability. IOS Press, 2009.

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

[**AMP06**] Alessandro Armando, Jacopo Mantovani, and Lorenzo Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. In *Proceedings of the 13th International SPIN Workshop on Model Checking of Software (SPIN'06)*, volume 3925 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2006

[**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

References

[BB09] R. Brummayer and A. Biere. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In S. Kowalewski and A. Philippou, editors, *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'05*, volume 5505 of *Lecture Notes in Computer Science*, pages 174–177. Springer, 2009

[BBC+05a] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. An incremental and layered procedure for the satisfiability of linear arithmetic logic. In *Tools and Algorithms for the Construction and Analysis of Systems, 11th Int. Conf., (TACAS)*, volume 3440 of *Lecture Notes in Computer Science*, pages 317–333, 2005

[BBC+05b] 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*, volume 3576 of *Lecture Notes in Computer Science*, pages 335–349. Springer, 2005

References

[**BCF$^+$07**] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, Ziyad Hanna, Alexander Nadel, Amit Palti, and Roberto Sebastiani. A lazy and layered SMT(BV) solver for hard industrial verification problems. In Werner Damm and Holger Hermanns, editors, *Proceedings of the 19^{th} International Conference on Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 547–560. Springer-Verlag, July 2007

[**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

[**BD94**] J. R. Burch and D. L. Dill. Automatic verification of pipelined microprocessor control. In *Procs. 6th Int. Conf. Computer Aided Verification (CAV)*, LNCS 818, pages 68–80, 1994

[**BDS02**] 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

References

[BGV01] R. E. Bryant, S. M. German, and M. N. Velev. Processor Verification Using Efficient Reductions of the Logic of Uninterpreted Functions to Propositional Logic. *ACM Transactions on Computational Logic, TOCL*, 2(1):93–134, 2001

[BLNM⁺09] C. Borralleras, S. Lucas, R. Navarro-Marset, E. Rodríguez-Carbonell, and A. Rubio. Solving Non-linear Polynomial Arithmetic via SAT Modulo Linear Arithmetic. In R. A. Schmidt, editor, *22nd International Conference on Automated Deduction , CADE-22*, volume 5663 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2009

[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

[BNO⁺08a] M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. A Write-Based Solver for SAT Modulo the Theory of Arrays. In *Formal Methods in Computer-Aided Design, FMCAD*, pages 1–8, 2008

[BNO⁺08b] Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. The Barcelogic SMT solver. In *Computer-aided Verification (CAV)*, volume 5123 of *Lecture Notes in Computer Science*, pages 294–298. Springer, 2008

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

[BV02] R. E. Bryant and M. N. Velev. Boolean Satisfiability with Transitivity Constraints. *ACM Transactions on Computational Logic, TOCL*, 3(4):604–627, 2002

[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

[CM06] S. Cotton and O. Maler. Fast and Flexible Difference Constraint Propagation for DPLL(T). In A. Biere and C. P. Gomes, editors, *9th International Conference on Theory and Applications of Satisfiability Testing, SAT'06*, volume 4121 of *Lecture Notes in Computer Science*, pages 170–183. Springer, 2006

References

[**DdM06**] Bruno Dutertre and Leonardo de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In T. Ball and R. B. Jones, editors, *18th International Conference on Computer Aided Verification, CAV'06*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006

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

[**dMB09**] L. de Moura and N. Bjørner. Generalized, efficient array decision procedures. In *9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009*, pages 45–52. IEEE, 2009

[**dMR02**] L. de Moura and H. Rueß. Lemmas on Demand for Satisfiability Solvers. In *5th International Conference on Theory and Applications of Satisfiability Testing, SAT'02*, pages 244–251, 2002

[**DP60**] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960

[**FLL$^+$02**] C. Flanagan, K. R. M Leino, M. Lillibridge, G. Nelson, and J. B. Saxe. Extended static checking for Java. In *Proc. ACM Conference on Programming Language Design and Implementation*, pages 234–245, June 2002

References

[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

[HT08] George Hagen and Cesare Tinelli. Scaling up the formal verification of Lustre programs with SMT-based techniques. In A. Cimatti and R. Jones, editors, *Proceedings of the 8th International Conference on Formal Methods in Computer-Aided Design (FMCAV'08), Portland, Oregon*, pages 109–117. IEEE, 2008

[JB10] Dejan Jovanović and Clark Barrett. Polite theories revisited. In Chris Fermüller and Andrei Voronkov, editors, *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 6397 of *Lecture Notes in Computer Science*, pages 402–416. Springer-Verlag, 2010

[KG07] Sava Krstić and Amit Goel. Architecting solvers for SAT modulo theories: Nelson-Oppen with DPLL. In B. Konev and F. Wolter, editors, *Proceeding of the Symposium on Frontiers of Combining Systems (Liverpool, England)*, volume 4720 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2007

References

[LM05] Shuvendu K. Lahiri and Madanlal Musuvathi. An Efficient Decision Procedure for UTVPI Constraints. In B. Gramlich, editor, *5th International Workshop on Frontiers of Combining Systems, FroCos'05*, volume 3717 of *Lecture Notes in Computer Science*, pages 168–183. Springer, 2005

[LNO06] S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras. SMT Techniques for Fast Predicate Abstraction. In T. Ball and R. B. Jones, editors, *18th International Conference on Computer Aided Verification, CAV'06*, volume 4144 of *Lecture Notes in Computer Science*, pages 413–426. Springer, 2006

[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

[NO80] Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980

[NO05] Robert Nieuwenhuis and Albert Oliveras. DPLL(T) with Exhaustive Theory Propagation and its Application to Difference Logic. In Kousha Etessami and Sriram K. Rajamani, editors, *Proceedings of the 17th International Conference on Computer Aided Verification, CAV'05 (Edimburgh, Scotland)*, volume 3576 of *Lecture Notes in Computer Science*, pages 321–334. Springer, July 2005

References

[NO07] R. Nieuwenhuis and A. Oliveras. Fast Congruence Closure and Extensions. *Information and Computation, IC*, 2005(4):557–580, 2007

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006

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

[PRSS99] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding Equality Formulas by Small Domains Instantiations. In N. Halbwachs and D. Peled, editors, *11th International Conference on Computer Aided Verification, CAV'99*, volume 1633 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1999

[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*, volume 3717 of *Lecture Notes in Computer Science*, pages 48–64. Springer, 2005

[SBDL01] A. Stump, C. W. Barrett, D. L. Dill, and J. R. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *16th Annual IEEE Symposium on Logic in Computer Science, LICS'01*, pages 29–37. IEEE Computer Society, 2001

[Sha02] Natarajan Shankar. Little engines of proof. In Lars-Henrik Eriksson and Peter A. Lindsay, editors, *FME 2002: Formal Methods - Getting IT Right, Proceedings of the International Symposium of Formal Methods Europe (Copenhagen, Denmark)*, volume 2391 of *Lecture Notes in Computer Science*, pages 1–20. Springer, July 2002

[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

[SSB02] O. Strichman, S. A. Seshia, and R. E. Bryant. Deciding Separation Formulas with SAT. In E. Brinksma and K. G. Larsen, editors, *14th International Conference on Computer Aided Verification, CAV'02*, volume 2404 of *Lecture Notes in Computer Science*, pages 209–222. Springer, 2002

References

[TdH08] N. Tillmann and J. de Halleux. Pex-White Box Test Generation for .NET. In B. Beckert and R. Hähnle, editors, *2nd International Conference on Tests and Proofs, TAP'08*, volume 4966 of *Lecture Notes in Computer Science*, pages 134–153. Springer, 2008

[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] C. Tinelli. A DPLL-based calculus for ground satisfiability modulo theories. In G. Ianni and S. 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

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

References

[WIGG05] C. Wang, F. Ivancic, M. K. Ganai, and A. Gupta. Deciding Separation Logic Formulae by SAT and Incremental Negative Cycle Elimination. In G. Sutcliffe and A. Voronkov, editors, *12h International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'05*, volume 3835 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2005

[ZM10] Harald Zankl and Aart Middeldorp. Satisfiability of Non-linear (Ir)rational Arithmetic. In Edmund M. Clarke and Andrei Voronkov, editors, *16th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'10*, volume 6355 of *Lecture Notes in Computer Science*, pages 481–500. Springer, 2010