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.

**Disclaimer:** 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 ≡ 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, ...)

SAT/SMT Summer School 2012
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):

  \[ g(a) = c \land (f(g(a)) \neq f(c) \lor g(a) = d) \land c \neq d \]

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

\[ a \times (|b| + c) = d \land b \times (|a| + c) \neq d \land a = b \]

is unsatisfiable, but no arithmetic reasoning is needed

If we abstract it to

\[ \text{mul}(a, \text{add}(\text{abs}(b), c)) = d \land \text{mul}(b, \text{add}(\text{abs}(a), c)) \neq d \land a = b \]

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:

- \( \forall a \ \forall i \ \forall v \ \text{read}(\text{write}(a, i, v), i) = v \)
- \( \forall a \ \forall i \ \forall j \ \forall v \ i \neq j \rightarrow \text{read}(\text{write}(a, i, v), j) = \text{read}(a, j) \)

Sometimes also with **extensionality**:

- \( \forall a \ \forall b \ (\forall i \ \text{read}(a, i) = \text{read}(b, i) \rightarrow a = b) \)

Is the following set of literals satisfiable in this theory?

\[
\text{write}(a, i, x) \neq b, \ \text{read}(b, i) = y, \ \text{read}(\text{write}(b, i, x), j) = y, \ a = b, \ i = j
\]
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?

\[ a[0 : 1] \neq b[0 : 1] \land (a | b) = c \land c[0] = 0 \land a[1] + b[1] = 0 \]
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:

\[
i = j + 2 \land a = \text{write}(b, i + 1, 4) \land \\
(\text{read}(a, j + 3) = 2 \lor f(i - 1) \neq f(j + 1))
\]

Often **decision procedures** for each theory combine modularly

[NO79, TH96, BBC\textsuperscript{+}05b]
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.

(Converting the formula in DNF and checking 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

\[
g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d
\]

Theory $T$: Equality with Uninterpreted Functions
(Very) Lazy Approach for SMT – Example

\[ g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \]

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

\[
g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d
\]

- Send \(\{1, \overline{2} \lor 3, \overline{4}\}\) to SAT solver.
(Very) Lazy Approach for SMT – Example

\[ g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d \]

1 \quad 2 \quad 3 \quad 4

- Send \( \{1, \overline{2} \lor 3, \overline{4}\} \) 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

\[
g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d
\]

1. Send \{1, \overline{2} \lor 3, \overline{4}\} to SAT solver.
2. SAT solver returns model \{1, \overline{2}, \overline{4}\}.
   Theory solver finds (concretization of) \{1, \overline{2}, \overline{4}\} unsat.
3. Send \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4\} to SAT solver.
(Very) Lazy Approach for SMT – Example

\[
g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d
\]

1. Send \{1, \overline{2} \lor 3, \overline{4}\} to SAT solver.
2. SAT solver returns model \{1, \overline{2}, \overline{4}\}.
   Theory solver finds (concretization of) \{1, \overline{2}, \overline{4}\} unsat.
3. Send \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4\} to SAT solver.
4. SAT solver returns model \{1, 3, \overline{4}\}.
   Theory solver finds \{1, 3, \overline{4}\} unsat.
(Very) Lazy Approach for SMT – Example

\[
g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d
\]

1. Send \{1, 2 \lor 3, 4\} to SAT solver.
2. SAT solver returns model \{1, 2, 4\}.
   Theory solver finds (concretization of) \{1, 2, 4\} unsat.
3. Send \{1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4\} to SAT solver.
4. SAT solver returns model \{1, 3, 4\}.
   Theory solver finds \{1, 3, 4\} unsat.
5. Send \{1, 2 \lor 3, 4, \overline{1} \lor 2, \overline{1} \lor 3 \lor 4\} to SAT solver.
(Very) Lazy Approach for SMT – Example

\[ g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d \]

1

2

3

4

• Send \( \{1, \overline{2} \lor 3, \overline{4}\} \) to SAT solver.

• SAT solver returns model \( \{1, \overline{2}, \overline{4}\} \).
  Theory solver finds (concretization of) \( \{1, \overline{2}, \overline{4}\} \) unsat.

• Send \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4\} \) to SAT solver.

• SAT solver returns model \( \{1, 3, \overline{4}\} \).
  Theory solver finds \( \{1, 3, \overline{4}\} \) unsat.

• Send \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2, \overline{1} \lor \overline{3} \lor 4\} \) to SAT solver.

• SAT solver finds \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4, \overline{1} \lor \overline{3} \lor 4\} \) 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 \([\text{NOT06}, \text{KG07}]\).
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 backtracking, 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 backtracking, 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 \textit{decision points} • denoting a partial truth \textit{assignment}
- \( F \) is a set of clauses denoting a CNF \textit{formula}

\textbf{Def.} If \( M = M_0 \cdot M_1 \cdot \cdots \cdot M_n \) where each \( M_i \) contains no decision points

- \( M_i \) is \textit{decision level} \( i \) of \( M \)
- \( M[i] \overset{\text{def}}{=} M_0 \cdot \cdots \cdot M_i \)
An Abstract Framework for DPLL

States:

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

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]

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

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**

\[ l_1 \lor \cdots \lor l_n \lor l \in F \]
\[ \bar{l}_1, \ldots, \bar{l}_n \in M \]
\[ l, \bar{l} \notin M \]

\[ M := M \setminus l \]

**Not.** Clauses are treated modulo ACI of \( \lor \)
Transition Rules for the Original DPLL

Extending the assignment

\[ \text{Propagate} \quad \frac{l_1 \lor \cdots \lor l_n \lor l \in F \quad \bar{l}_1, \ldots, \bar{l}_n \in M \quad l, \bar{l} \notin M}{M := M \cdot l} \]

\textbf{Not.} Clauses are treated modulo ACI of $\lor$

\[ \text{Decide} \quad \frac{l \in \text{Lit}(F) \quad l, \bar{l} \notin M}{M := M \cdot l} \]

\textbf{Not.} $\text{Lit}(F) \overset{\text{def}}{=} \{l \mid l \text{ literal of } F\} \cup \{\bar{l} \mid l \text{ literal of } F\}$
Transition Rules for the Original DPLL

Repairing the assignment

\[ l_1 \lor \cdots \lor l_n \in F \quad \overline{l}_1, \ldots, \overline{l}_n \in M \quad \bullet \notin M \]

fail
Transition Rules for the Original DPLL

Repairing the assignment

\[
\text{Fail} \quad \frac{l_1 \lor \cdots \lor l_n \in F \quad \overline{l}_1, \ldots, \overline{l}_n \in M \quad \bullet \notin M}{\text{fail}}
\]

\[
\text{Backtrack} \quad \frac{l_1 \lor \cdots \lor l_n \in F \quad \overline{l}_1, \ldots, \overline{l}_n \in M \quad M = M \bullet l \quad N \quad \bullet \notin N}{M := M \overline{l}}
\]

\text{NB:} \text{ Last premise of } \text{Backtrack} \text{ 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 $\text{no}$ 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** \[
\begin{align*}
C &= \text{no} \quad \ell_1 \lor \cdots \lor \ell_n \in F \quad \bar{\ell}_1, \ldots, \bar{\ell}_n \in M \\
C &:= \ell_1 \lor \cdots \lor \ell_n
\end{align*}
\]

**Explain** \[
\begin{align*}
C &= \ell \lor D \quad \ell_1 \lor \cdots \lor \ell_n \lor \bar{l} \in F \quad \bar{\ell}_1, \ldots, \bar{\ell}_n \prec_M \bar{l} \\
C &:= \ell_1 \lor \cdots \lor \ell_n \lor D
\end{align*}
\]

**Backjump** \[
\begin{align*}
C &= \ell_1 \lor \cdots \lor \ell_n \lor l \quad \text{lev} \ \bar{\ell}_1, \ldots, \text{lev} \ \bar{\ell}_n \leq i < \text{lev} \ \bar{l} \\
C &:= \text{no} \quad M := M^{[i]} \ l
\end{align*}
\]

**Not.** \( l \prec_M l' \) if \( l \) occurs before \( l' \) in \( M \)

\( \text{lev} \ l = i \) iff \( l \) occurs in decision level \( i \) of \( M \)
From DPLL to CDCL Solvers (2)

Replace **Backtrack** with

**Conflict**

\[ C = \text{no} \quad l_1 \lor \cdots \lor l_n \in F \quad \bar{l}_1, \ldots, \bar{l}_n \in M \]

\[ C := l_1 \lor \cdots \lor l_n \]

**Explain**

\[ C = l \lor D \quad l_1 \lor \cdots \lor l_n \lor \bar{l} \in F \quad \bar{l}_1, \ldots, \bar{l}_n \prec_M \bar{l} \]

\[ C := l_1 \lor \cdots \lor l_n \lor D \]

**Backjump**

\[ C = l_1 \lor \cdots \lor l_n \lor l \quad \text{lev } \bar{l}_1, \ldots, \text{lev } \bar{l}_n \leq i < \text{lev } \bar{l} \]

\[ C := \text{no} \quad M := M^{[i]} l \]

Maintain **invariant**: \( F \models_p C \) and \( M \models_p \neg C \) when \( C \neq \text{no} \)

**Not.** \( \models_p \) denotes propositional entailment
From DPLL to CDCL Solvers (3)

Modify \textbf{Fail} to

\[
\text{Fail} \quad \frac{C \neq \text{no} \quad \bullet \notin M}{\text{fail}}
\]
Execution Example

\[ F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, \overline{2} \lor \overline{5} \lor 6 \lor 7\} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(F)</td>
<td>no</td>
<td>//</td>
</tr>
</tbody>
</table>
## Execution Example

\[ F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, \overline{2} \lor \overline{5} \lor 6 \lor \overline{7}\} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(F)</td>
<td>no</td>
<td>//</td>
</tr>
<tr>
<td>1</td>
<td>(F)</td>
<td>no</td>
<td>by <strong>Propagate</strong></td>
</tr>
</tbody>
</table>
### Execution Example

\[
F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, \overline{2} \lor \overline{5} \lor 6 \lor \overline{7}\}
\]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F)</td>
<td>no</td>
<td>//</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
</tbody>
</table>
# Execution Example

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$F$</td>
<td>no</td>
<td>//</td>
</tr>
<tr>
<td>1</td>
<td>$F$</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2</td>
<td>$F$</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2•3</td>
<td>$F$</td>
<td>no</td>
<td>by Decide</td>
</tr>
</tbody>
</table>
**Execution Example**

\[ F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, 2 \lor \overline{5} \lor 6 \lor \overline{7}\} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( F )</td>
<td>no</td>
<td>/\</td>
</tr>
<tr>
<td>1</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3</td>
<td>( F )</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
</tbody>
</table>
## Execution Example

\[ F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, \overline{2} \lor \overline{5} \lor 6 \lor \overline{7}\} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F)</td>
<td>no</td>
<td></td>
<td>//</td>
</tr>
<tr>
<td>1</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
</tbody>
</table>
## Execution Example

\[ F := \{1, \, \bar{1} \lor 2, \, 3 \lor 4, \, 5 \lor \bar{6}, \, \bar{1} \lor 5 \lor 7, \, 2 \lor 5 \lor \bar{6} \lor \bar{7}\} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>no</td>
<td></td>
<td>//</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2</td>
<td>F</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3</td>
<td>F</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>F</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>F</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>F</td>
<td>no</td>
<td>by Propagate</td>
</tr>
</tbody>
</table>
# Execution Example

\[ F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, \overline{1} \lor \overline{5} \lor 7, \overline{2} \lor \overline{5} \lor 6 \lor \overline{7}\} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(F)</td>
<td>no</td>
<td>//</td>
</tr>
<tr>
<td>1</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 • 3</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 • 3 4</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 • 3 4 • 5</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 • 3 4 • 5 6</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 • 3 4 • 5 6 7</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
</tbody>
</table>
## Execution Example

\[ F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, \overline{2} \lor \overline{5} \lor 6 \lor 7\} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(F)</td>
<td>no</td>
<td>//</td>
</tr>
<tr>
<td>1</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 (\bullet) 3</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 (\bullet) 3 4</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 (\bullet) 3 4 (\bullet) 5</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 (\bullet) 3 4 (\bullet) 5 6</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 (\bullet) 3 4 (\bullet) 5 6 7</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 (\bullet) 3 4 (\bullet) 5 6 7</td>
<td>(\overline{2} \lor \overline{5} \lor 6 \lor 7)</td>
<td>no</td>
<td>by Conflict</td>
</tr>
</tbody>
</table>

Foundations of Lazy SMT and DPLL(\(T\)) – p.30/86
### Execution Example

\[ F := \{ 1, \bar{1} \lor 2, \bar{3} \lor 4, \bar{5} \lor 6, \bar{1} \lor \bar{5} \lor 7, \bar{2} \lor \bar{5} \lor 6 \lor \bar{7} \} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(F)</td>
<td>no</td>
<td>//</td>
</tr>
<tr>
<td>1</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>(F)</td>
<td>(\bar{2} \lor \bar{5} \lor 6 \lor \bar{7})</td>
<td>by Conflict</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>(F)</td>
<td>(1 \lor \bar{2} \lor \bar{5} \lor 6)</td>
<td>by Explain with (\bar{1} \lor \bar{5} \lor 7)</td>
</tr>
</tbody>
</table>
## Execution Example

\[ F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, 2 \lor \overline{5} \lor 6 \lor \overline{7}\} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( F )</td>
<td>no</td>
<td>//</td>
</tr>
<tr>
<td>1</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 ( \cdot 3 )</td>
<td>( F )</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 ( \cdot 3 \ 4 )</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 ( \cdot 3 \ 4 \ \cdot 5 )</td>
<td>( F )</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 ( \cdot 3 \ 4 \ \cdot 5 \ \overline{6} )</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 ( \cdot 3 \ 4 \ \cdot 5 \ \overline{6} \ 7 )</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 ( \cdot 3 \ 4 \ \cdot 5 \ \overline{6} \ 7 )</td>
<td>( F \ \overline{2} \lor \overline{5} \lor 6 \lor \overline{7} )</td>
<td>no</td>
<td>by Conflict</td>
</tr>
<tr>
<td>1 2 ( \cdot 3 \ 4 \ \cdot 5 \ \overline{6} \ 7 )</td>
<td>( F \ \overline{1} \lor 2 \lor \overline{5} \lor 6 )</td>
<td>by Explain with ( \overline{1} \lor \overline{5} \lor 7 )</td>
<td></td>
</tr>
<tr>
<td>1 2 ( \cdot 3 \ 4 \ \cdot 5 \ \overline{6} \ 7 )</td>
<td>( F \ \overline{1} \lor 2 \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor 6 )</td>
<td></td>
</tr>
</tbody>
</table>
### Execution Example

\[ F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor 5 \lor 7, \overline{2} \lor 5 \lor 6 \lor 7\} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( F )</td>
<td>no</td>
<td>//</td>
</tr>
<tr>
<td>1</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 ( \bullet 3 )</td>
<td>( F )</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 ( \bullet 3 ) ( \bullet 4 )</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 ( \bullet 3 ) ( \bullet 4 ) ( \bullet 5 )</td>
<td>( F )</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 ( \bullet 3 ) ( \bullet 4 ) ( \bullet 5 ) ( \overline{6} )</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 ( \bullet 3 ) ( \bullet 4 ) ( \bullet 5 ) ( \overline{6} ) ( \overline{7} )</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 ( \bullet 3 ) ( \bullet 4 ) ( \bullet 5 ) ( \overline{6} ) ( \overline{7} )</td>
<td>( F )</td>
<td>( \overline{2} \lor \overline{5} \lor 6 \lor 7 )</td>
<td>by Conflict</td>
</tr>
<tr>
<td>1 2 ( \bullet 3 ) ( \bullet 4 ) ( \bullet 5 ) ( \overline{6} ) ( \overline{7} )</td>
<td>( F )</td>
<td>( 1 \lor \overline{2} \lor \overline{5} \lor 6 )</td>
<td>by Explain with ( \overline{1} \lor \overline{5} \lor 7 )</td>
</tr>
<tr>
<td>1 2 ( \bullet 3 ) ( \bullet 4 ) ( \bullet 5 ) ( \overline{6} ) ( \overline{7} )</td>
<td>( F )</td>
<td>( 1 \lor \overline{2} \lor \overline{5} )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 ( \overline{5} )</td>
<td>( F )</td>
<td>no</td>
<td>by Backjump</td>
</tr>
</tbody>
</table>
# Execution Example

\[
F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, 2 \lor \overline{5} \lor 6 \lor \overline{7}\}
\]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(F)</td>
<td>no</td>
<td>//</td>
</tr>
<tr>
<td>1</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 3 4 5 (\overline{6})</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3 4 5 (\overline{6}) (\overline{7})</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3 4 5 (\overline{6}) (\overline{7}) (\overline{7})</td>
<td>(F)</td>
<td>(\overline{2} \lor \overline{5} \lor 6 \lor \overline{7})</td>
<td>by Conflict</td>
</tr>
<tr>
<td>1 2 3 4 5 (\overline{6}) (\overline{7}) (\overline{7})</td>
<td>(F)</td>
<td>(1 \lor \overline{2} \lor \overline{5} \lor 6)</td>
<td>by Explain with (\overline{1} \lor \overline{5} \lor 7)</td>
</tr>
<tr>
<td>1 2 3 4 5 (\overline{6}) (\overline{7}) (\overline{7})</td>
<td>(F)</td>
<td>(1 \lor \overline{2} \lor \overline{5})</td>
<td>by Explain with (5 \lor 6)</td>
</tr>
<tr>
<td>(1 2 5)</td>
<td>(F)</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>(1 2 \overline{5} \bullet 3)</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

SAT/SMT Summer School 2012

Foundations of Lazy SMT and DPLL(\(T\)) – p.30/86
Also add

\[
\begin{align*}
\text{Learn} & \quad \frac{F \models_p C \quad C \notin F}{F := F \cup \{C\}} \\
\text{Forget} & \quad C = \text{no} \quad F = G \cup \{C\} \quad G \models_p C \quad F := G
\end{align*}
\]

\text{Restart} \quad M := M^{[0]} \quad C := \text{no}

\textbf{NB:} Learn can be applied to any clause stored in C when C \neq \text{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** $\overset{\text{def}}{=} \{ \text{Propagate, Decide, Conflict, Explain, Backjump} \}$

**DPLL** $\overset{\text{def}}{=} \text{Basic DPLL} + \{ \text{Learn, Forget, Restart} \}$
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 $M = ()$ and $C = \text{no}$

*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 $M = ()$ and $C = no$

*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 $M = ()$ and $C = \text{no}$

*Exhausted execution:* execution ending in an irreducible state

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

**Proposition** (Completeness) For every exhausted execution starting with $F = F_0$ and ending with $C = \text{no}$, 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 \textbf{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 \( M \models_p \neg C \) when \( C \neq \text{no} \)

**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 $\begin{array}{c}
C = \text{no} \quad l_1, \ldots, l_n \in M \\
\therefore \quad C := \overline{l_1} \lor \cdots \lor \overline{l_n}
\end{array}$

**Not:** $\perp = \text{empty clause}$

**NB:** $\models_T$ decided by theory solver
SMT-level Rules

Fix a theory $T$

**$T$-Conflict**
\[
\text{C} = \text{no} \quad l_1, \ldots, l_n \in M \quad l_1, \ldots, l_n \models_T \bot \\
\text{C} := \overline{l}_1 \lor \cdots \lor \overline{l}_n
\]

**$T$-Propagate**
\[
l \in \text{Lit}(F) \quad M \models_T l \quad l, \overline{l} \notin M \\
M := M \downarrow l
\]

**Not:** $\bot = \text{empty clause}$

**NB:** $\models_T$ decided by theory solver
SMT-level Rules

Fix a theory $T$

**T-Conflict**

$C = no \quad l_1, \ldots, l_n \in M \quad l_1, \ldots, l_n \models_T \bot$

$C := \overline{l_1} \lor \cdots \lor \overline{l_n}$

**T-Propagate**

$l \in \text{Lit}(F) \quad M \models_T l \quad l, \overline{l} \notin M$

$M := M \cup \{l\}$

**T-Explain**

$C = l \lor D \quad \overline{l_1}, \ldots, \overline{l_n} \models_T \overline{l} \quad \overline{l_1}, \ldots, \overline{l_n} \prec_M \overline{l}$

$C := l_1 \lor \cdots \lor l_n \lor D$

**Not:** $\bot = \text{empty clause}$

**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

\[
\begin{align*}
\begin{array}{c}
g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \\
\end{array}
\end{align*}
\]

1 \quad 3

\[
\begin{align*}
\begin{array}{c}
g(a) = d \quad \land \\
\end{array}
\end{align*}
\]

2 \quad 4

\[
\begin{align*}
\begin{array}{c}
c \neq d \\
\end{array}
\end{align*}
\]
## Modeling the Very Lazy Theory Approach

\[
g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d
\]

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>F</td>
<td>C</td>
<td>rule</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>no</td>
<td>//</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 4</td>
<td>1, 2 ∨ 3, 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 4 • 2</td>
<td>1, 2 ∨ 3, 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 4 • 2</td>
<td>1, 2 ∨ 3, 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 4 • 2</td>
<td>1, 2 ∨ 3, 4, 1 ∨ 2 ∨ 4</td>
<td>1 ∨ 2 ∨ 4</td>
<td>by $T$-Conflict</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 4</td>
<td>1, 2 ∨ 3, 4, 1 ∨ 2 ∨ 4</td>
<td>no</td>
<td>by Restart</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 ∨ 3, 4, 1 ∨ 2 ∨ 4</td>
<td>no</td>
<td>by Propagate$^+$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 ∨ 3, 4, 1 ∨ 2 ∨ 4, 1 ∨ 3 ∨ 4</td>
<td>1 ∨ 3 ∨ 4</td>
<td>by $T$-Conflict, Learn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 ∨ 3, 4, 1 ∨ 2 ∨ 4, 1 ∨ 3 ∨ 4</td>
<td>no</td>
<td>by Restart</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 ∨ 3, 4, 1 ∨ 2 ∨ 4, 1 ∨ 3 ∨ 4</td>
<td>1 ∨ 3 ∨ 4</td>
<td>by Conflict</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td>by Fail</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

\[
\begin{align*}
g(a) = c & \quad \land \quad f(g(a)) \neq f(c) & \quad \lor \quad g(a) = d & \quad \land \quad c \neq d \\
1 & & 2 & & 3 & & 4
\end{align*}
\]
A Better Lazy Approach

\[
g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d
\]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>( \overline{1} \lor \overline{3} \lor \overline{4} )</td>
<td>no by Propagate$^+$</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>( \overline{1} \lor \overline{3} \lor \overline{4} )</td>
<td>no by Propagate$^+$</td>
</tr>
<tr>
<td>1</td>
<td>4  • \overline{2}</td>
<td>( \overline{1} \lor \overline{3} \lor \overline{4} )</td>
<td>( \overline{1} \lor \overline{2} ) by T-Conflict</td>
</tr>
<tr>
<td>1</td>
<td>4  • \overline{2}</td>
<td>( \overline{1} \lor \overline{3} \lor \overline{4} )</td>
<td>( \overline{1} \lor \overline{2} ) by T-Conflict</td>
</tr>
<tr>
<td>1</td>
<td>4  • \overline{2}</td>
<td>( \overline{1} \lor \overline{3} \lor \overline{4} )</td>
<td>( \overline{1} \lor \overline{2} ) by T-Conflict</td>
</tr>
<tr>
<td>1</td>
<td>4  • \overline{2}</td>
<td>( \overline{1} \lor \overline{3} \lor \overline{4} )</td>
<td>( \overline{1} \lor \overline{2} ) by T-Conflict</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

\[
\begin{array}{c}
\frac{\ell \in \text{Lit}(F) \quad M \models_T \ell \quad \ell, \bar{\ell} \notin M}{M := M \uplus \ell}
\end{array}
\]

$T$-Explain

\[
\begin{array}{c}
\frac{C = \ell \lor D \quad \bar{\ell}_1, \ldots, \bar{\ell}_n \models_T \bar{\ell} \quad \bar{\ell}_1, \ldots, \bar{\ell}_n \prec_M \bar{\ell}}{C := \ell_1 \lor \cdots \lor \ell_n \lor D}
\end{array}
\]
Theory Propagation Example

\[
g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d
\]
### Theory Propagation Example

\[
g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d
\]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2 ∨ 3, 4</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2 ∨ 3, 4</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2 ∨ 3, 4</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2 ∨ 3, 4</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NB:** \(T\)-propagation eliminates search altogether in this case, no applications of Decide are needed.
Theory Propagation Example (2)

\[
g(a) = e \lor g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d
\]
Theory Propagation Example (2)

\( g(a) = e \lor g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \)

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>//</td>
<td></td>
</tr>
<tr>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Propagate</td>
<td></td>
</tr>
<tr>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Decide</td>
<td></td>
</tr>
<tr>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by ( T )-Propagate ((1 \models_T 2))</td>
<td></td>
</tr>
<tr>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by ( T )-Propagate ((1, 4 \models_T 3))</td>
<td></td>
</tr>
<tr>
<td>1, 2 \lor 3, 4</td>
<td>(2 \lor 3)</td>
<td>by Conflict</td>
<td></td>
</tr>
<tr>
<td>1, 2 \lor 3, 4</td>
<td>(1 \lor 3)</td>
<td>by ( T )-Explain</td>
<td></td>
</tr>
<tr>
<td>1, 2 \lor 3, 4</td>
<td>(1 \lor 4)</td>
<td>by ( T )-Explain</td>
<td></td>
</tr>
<tr>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Backjump</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>(exercise)</td>
<td></td>
</tr>
</tbody>
</table>
Theory Propagation Features

- With exhaustive theory propagation every assignment $M$ is $T$-satisfiable (since $M \perp$ is $T$-unsatisfiable iff $M \models_T \perp$).

$\Box$
Theory Propagation Features

• With exhaustive theory propagation every assignment $M$ is $T$-satisfiable (since $M \models T l$ 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 $M \models T\ l$ is $T$-unsatisfiable iff $M \models T \ \overline{l}$).

• 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 $M \models T \overline{\mathbf{l}}$ is $T$-unsatisfiable iff $M \models T \mathbf{l}$).

- 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 $M \models T$ 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 $\overset{\text{def}}{=} (1) + (2)$

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

Updated terminology:

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

*Execution*: sequence of transitions allowed by the rules and starting with $M = ()$ and $C = no$

*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 \( M = () \) and \( C = \text{no} \)

*Exhausted execution:* execution ending in an irreducible state

**Proposition** (Termination) Every execution in which
(a) *Learn/Forget* are applied only *finitely many times* and
(b) *Restart* is applied with *increased periodicity*
is finite.

**Lemma** Every exhausted execution ends with either \( C = \text{no} \) 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 $M = ()$ and $C = \text{no}$

Exhausted execution: execution ending in an irreducible state

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

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

SAT/SMT Summer School 2012
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]

DPLL($T$) = DPLL($X$) engine + $T$-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.

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

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

$i = j$) 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.

\[
M = \{ r(w(a, i, x), j) \neq x, \ r(w(a, i, x), j) \neq r(a, j) \} \\
\]

\( i = j \) 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.

\[
M = \{ r(w(a, i, x), j) \neq x, \ r(w(a, i, x), j) \neq r(a, j) \}
\]

1. \( i = j \) Then, \( r(w(a, i, x), j) = x \). Contradiction with 1.
2. \( i \neq j \) 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 \cite{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:**

\[ M = \{ \ldots, s = r(w(a, i, t), j), \ldots \} \]
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:**

\[ M = \{ \ldots, s = r(w(a, i, t), j), \ldots \} \]

- Main SMT module: “Is M \( T \)-unsatisfiable?”
Splitting on Demand \[\text{[BNOT06]}\]

**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:**

\[M = \{\ldots, s = r(w(a, i, t), j), \ldots\}\]

- **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*:

\[
s = s' \land i = j \rightarrow s = t, \quad s = s' \land i \neq j \rightarrow s = r(a, j)
\]
Modeling Splitting on Demand

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

\[ T\text{-Learn} \]

\[ \models_T \exists v (l_1 \lor \cdots \lor l_n) \quad l_1, \ldots, l_n \in L_S \quad v \text{ vars not in } F \]

\[ F := F \cup \{l_1 \lor \cdots \lor l_n\} \]

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

\[ T\text{-Learn} \]

\[ \models_T \exists v (l_1 \lor \cdots \lor l_n) \quad l_1, \ldots, l_n \in L_S \quad v \text{ vars not in } F \]

\[ F := F \cup \{l_1 \lor \cdots \lor l_n\} \]

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 $M \models_T \bot$ 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 $M \models_T F$, it must either

- determine whether $M \models_T \bot$ 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 $M \models_T \bot$ 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 $M \models_T F$, it must either

- determine whether $M \models_T \bot$ 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 $M \models_T \bot$ only if all literals in $L_S$ are defined in $M$

**NB:** In practice, to determine if $M \models_T \bot$ the $T$-solver only needs a small subset of $L_S$ to be defined in $M$
Example — Theory of Finite Sets

\[ F : \ x = y \cup z \ \land \ y \neq \emptyset \lor x \neq z \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = y \cup z )</td>
<td>( F )</td>
<td>by Propagate$^+$</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset )</td>
<td>( F )</td>
<td>by Decide</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land x \neq z )</td>
<td>( F )</td>
<td>by Propagate</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land x \neq z \land e \in x )</td>
<td>( F, (x = z \lor e \in x \land e \in z) ), ( (x = z \lor e \notin x \land e \notin z) )</td>
<td>by T-Learn</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land x \neq z \land e \in x \land e \notin z )</td>
<td>( F, (x = z \lor e \in x \land e \in z) ), ( (x = z \lor e \notin x \land e \notin z) )</td>
<td>by Decide</td>
</tr>
</tbody>
</table>

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

\[ e \in x \ \cdots \ \Rightarrow e \in y \cup z \ \cdots \ \Rightarrow e \in y \ \cdots \ \Rightarrow e \in \emptyset \ \Rightarrow \bot \]

This enables an application of T-Conflict with clause

\[ \neg x = y \cup z \lor \neg y \notin \emptyset \lor x = z \lor e \notin x \lor e \in z \]
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 $M \models_T \bot$ 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:

\[ a \approx b + 2 \land A \approx \text{write}(B, a + 1, 4) \land \]
\[ (\text{read}(A, b + 3) \approx 2 \lor f(a - 1) \neq f(b + 1)) \]
Need for Combining Theories and Solvers

Recall: Many applications give rise to formulas like:

\[
\begin{align*}
  a & \approx b + 2 \land A \approx \text{write}(B, a + 1, 4) \land \\
  (\text{read}(A, b + 3) & \approx 2 \lor f(a - 1) \neq f(b + 1))
\end{align*}
\]

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:

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

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 \(T_{LA} \cup T_A \cup T_{EUF}\)?
Need for Combining Theories and Solvers

**Recall:** Many applications give rise to formulas like:

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

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 $T_{LA} \cup T_{A} \cup T_{EUF}$?

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 $T_{LRA} \cup T_{EUF}$ ($T_{LRA}$, linear real arithmetic):

\[
\begin{align*}
    f(f(x) - f(y)) &= a \\
    f(0) &> a + 2 \\
    x &= y
\end{align*}
\]
Motivating Example (Convex Case)

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

$$f(f(x) - f(y)) = a$$
$$f(0) > a + 2$$
$$x = y$$

First step: *purify* literals so that each belongs to a single theory
Motivating Example (Convex Case)

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

\[
\begin{align*}
  f(f(x) - f(y)) &= a \\
  f(0) &> a + 2 \\
  x &= y
\end{align*}
\]

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

\[
\begin{align*}
  f(f(x) - f(y)) &= a &\implies& f(e_1) &= a &\implies& f(e_1) &= a \\
  e_1 &= f(x) - f(y) &\implies& e_1 &= e_2 - e_3 \\
  e_2 &= f(x) \\
  e_3 &= f(y)
\end{align*}
\]
Motivating Example (Convex Case)

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

\[
\begin{align*}
  f(f(x) - f(y)) &= a \\
  f(0) &> a + 2 \\
  x &= y
\end{align*}
\]

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

\[
\begin{align*}
  f(0) &= a + 2 &\implies f(e_4) &= a + 2 &\implies f(e_4) &= e_5 \\
  e_4 &= 0 &e_4 &= 0 \\
  e_5 &> a + 2
\end{align*}
\]
Motivating Example (Convex Case)

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

<table>
<thead>
<tr>
<th></th>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td></td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td></td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 &gt; a + 2$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Motivating Example (Convex Case)

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

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td></td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
</tr>
</tbody>
</table>

$L_1 \models_{	ext{EUF}} e_2 = e_3$
Motivating Example (Convex Case)

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

<table>
<thead>
<tr>
<th></th>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1)$</td>
<td>$a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x)$</td>
<td>$e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y)$</td>
<td>$e_3$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4)$</td>
<td>$e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Motivating Example (Convex Case)

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

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
</tr>
</tbody>
</table>

$L_2 \models_{LRA} e_1 = e_4$
Motivating Example (Convex Case)

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

<table>
<thead>
<tr>
<th></th>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1)$</td>
<td>$a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x)$</td>
<td>$e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y)$</td>
<td>$e_3$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4)$</td>
<td>$e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x$</td>
<td>$y$</td>
<td></td>
</tr>
<tr>
<td>$e_1$</td>
<td>$e_4$</td>
<td></td>
</tr>
</tbody>
</table>
Motivating Example (Convex Case)

Second step: exchange entailed interface equalities, equalities over shared constants \( e_1, e_2, e_3, e_4, e_5, a \)

<table>
<thead>
<tr>
<th>( L_1 )</th>
<th>( L_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(e_1) = a )</td>
<td>( e_2 - e_3 = e_1 )</td>
</tr>
<tr>
<td>( f(x) = e_2 )</td>
<td>( e_4 = 0 )</td>
</tr>
<tr>
<td>( f(y) = e_3 )</td>
<td>( e_5 &gt; a + 2 )</td>
</tr>
<tr>
<td>( f(e_4) = e_5 )</td>
<td>( e_2 = e_3 )</td>
</tr>
<tr>
<td>( x = y )</td>
<td></td>
</tr>
<tr>
<td>( e_1 = e_4 )</td>
<td></td>
</tr>
<tr>
<td>( L_1 \models_{\text{EUF}} a = e_5 )</td>
<td></td>
</tr>
</tbody>
</table>
Motivating Example (Convex Case)

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

<table>
<thead>
<tr>
<th></th>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
<td></td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
<td></td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 &gt; a + 2$</td>
<td></td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td>$e_2 = e_3$</td>
<td></td>
</tr>
<tr>
<td>$x = y$</td>
<td>$a = e_5$</td>
<td></td>
</tr>
<tr>
<td>$e_1 = e_4$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Motivating Example (Convex Case)**

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

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$a = e_5$</td>
</tr>
<tr>
<td>$e_1 = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

**Third step:** check for satisfiability locally

$L_1 \not\models_{EUF} ⊥$

$L_2 \models_{LRA} ⊥$
Motivating Example (Convex Case)

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

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$a = e_5$</td>
</tr>
<tr>
<td>$e_1 = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

**Third step:** check for satisfiability locally

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

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

Report unsatisfiable
Motivating Example (Non-convex Case)

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

\[
\begin{align*}
1 & \leq x \leq 2 \\
f(1) & = a \\
f(x) & = b \\
a & = b + 2
\end{align*}
\]
Motivating Example (Non-convex Case)

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

\[
1 \leq x \leq 2 \\
f(1) = a \\
f(x) = b \\
a = b + 2
\]

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 $T_{LIA} \cup T_{EUF}$:

\[
\begin{align*}
1 & \leq x & \leq 2 \\
 f(1) & = a \\
 f(x) & = b \\
 a & = b + 2
\end{align*}
\]

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

\[
\begin{align*}
f(1) = a & \implies f(e_1) = a \\
e_1 & = 1
\end{align*}
\]
Motivating Example (Non-convex Case)

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

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

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

$$f(2) = f(1) + 3 \implies e_2 = 2$$

$$f(e_2) = e_3$$

$$f(e_1) = e_4$$

$$e_3 = e_4 + 3$$
Motivating Example (Non-convex Case)

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

<table>
<thead>
<tr>
<th>L_1</th>
<th>L_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>
Motivating Example (Non-convex Case)

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

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

No more entailed equalities, but $L_1 \models_{LIA} x = e_1 \lor x = e_2$
Motivating Example (Non-convex Case)

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

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

Consider each case of $x = e_1 \lor x = e_2$ separately
Motivating Example (Non-convex Case)

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

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

Case 1) $x = e_1$
Motivating Example (Non-convex Case)

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

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td>$x = e_1$</td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
<tr>
<td>$x = e_1$</td>
<td></td>
</tr>
</tbody>
</table>
Motivating Example (Non-convex Case)

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

<table>
<thead>
<tr>
<th></th>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td></td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td></td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td></td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td></td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
<td>$x = e_1$</td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = e_1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$L_2 \models_{EUF} a = b$, which entails $\bot$ when sent to $L_1$
Motivating Example (Non-convex Case)

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

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>
Motivating Example (Non-convex Case)

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

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

Case 2) $x = e_2$
Motivating Example (Non-convex Case)

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

<table>
<thead>
<tr>
<th></th>
<th>( L_1 )</th>
<th>( L_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1 \leq x )</td>
<td>( f(e_1) = a )</td>
<td></td>
</tr>
<tr>
<td>( x \leq 2 )</td>
<td></td>
<td>( f(x) = b )</td>
</tr>
<tr>
<td>( e_1 = 1 )</td>
<td></td>
<td>( f(e_2) = e_3 )</td>
</tr>
<tr>
<td>( a = b + 2 )</td>
<td>( f(e_1) = e_4 )</td>
<td></td>
</tr>
<tr>
<td>( e_2 = 2 )</td>
<td></td>
<td>( x = e_2 )</td>
</tr>
<tr>
<td>( e_3 = e_4 + 3 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( a = e_4 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x = e_2 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Motivating Example (Non-convex Case)

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

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td>$x = e_2$</td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
<tr>
<td>$x = e_2$</td>
<td></td>
</tr>
</tbody>
</table>

$L_2 \models_{EUF} e_3 = b$, which entails $\bot$ when sent to $L_1$
The Nelson-Oppen Method

• For $i = 1, 2$, let $T_i$ be a first-order theory of signature $\Sigma_i$ (set of function and predicate symbols in $T_i$ other than $=$)
• Let $T = T_1 \cup T_2$
• Let $C$ 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 $\Sigma_i$ (set of function and predicate symbols in $T_i$ other than $=$)

• Let $T = T_1 \cup T_2$

• Let $C$ 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) $(\Sigma_i \cup C)$-literals
The Nelson-Oppen Method

• For \( i = 1, 2 \), let \( T_i \) be a first-order theory of signature \( \Sigma_i \) (set of function and predicate symbols in \( T_i \) other than \( = \))

• Let \( T = T_1 \cup T_2 \)

• Let \( C \) 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) \((\Sigma_i \cup C)\)-literals

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

Output: sat or unsat

1. Guess an arrangement \( A \), i.e., a set of equalities and disequalities over \( C \) such that

\[ c = d \in A \quad \text{or} \quad c \neq d \in A \quad \text{for all} \quad c, d \in C \]
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 \((\Sigma_i \cup C)\)-literals

Output: sat or unsat

1. Guess an arrangement \( A \), i.e., a set of equalities and disequalities over \( C \) such that
   \[ c = d \in A \text{ or } c \neq d \in A \text{ for all } c, d \in C \]

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 \((\Sigma_i \cup C)\)-literals
Output: sat or unsat

1. Guess an arrangement \( A \), i.e., a set of equalities and disequalities over \( C \) such that

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

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
\[ L \models_T s_1 = t_1 \lor \cdots \lor s_n = t_n \implies L \models_T s_i = t_i \text{ for some } i \]

**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 $\text{DPLL}(T_1, \ldots, T_n)$ Abstractly

- Let $n = 2$, for simplicity
- Let $T_i$ be of signature $\Sigma_i$ 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 $M|_i \overset{\text{def}}{=} \{(\Sigma_i \cup \mathcal{C})\text{-literals of } M \text{ and their complement}\}$
- Let $I(M) \overset{\text{def}}{=} \{c = d \mid c, d \text{ occur in } \mathcal{C}, M|_1 \text{ and } M|_2\} \cup \{c \neq d \mid c, d \text{ occur in } \mathcal{C}, M|_1 \text{ and } M|_2\}$
  
  (*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

\[
\begin{align*}
    l \in \text{Lit}(F) \cup I(M) & \quad l, \bar{l} \notin M \\
    M := M \bullet l
\end{align*}
\]

Only change: decide on interface equalities as well
Abstract DPLL Modulo Multiple Theories

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

\[ \text{Decide} \quad l \in \text{Lit}(F) \cup \text{I}(M) \quad l, \overline{l} \notin M \]
\[ M := M \cdot l \]

Only change: decide on interface equalities as well

\[ \text{T-Propagate} \]
\[ l \in \text{Lit}(F) \cup \text{I}(M) \quad i \in \{1, 2\} \quad M \models_{T_i} l \quad l, \overline{l} \notin M \]
\[ M := M \ l \]

Only change: propagate interface equalities as well, but reason locally in each \( T_i \)
Abstract DPLL Modulo Multiple Theories

**T-Conflict**

\[ C = \text{no} \quad l_1, \ldots, l_n \in M \quad l_1, \ldots, l_n \models_{T_i} \bot \quad i \in \{1, 2\} \]

\[ C := \overline{l}_1 \lor \cdots \lor \overline{l}_n \]

**T-Explain**

\[ C = l \lor D \quad \overline{l}_1, \ldots, \overline{l}_n \models_{T_i} \overline{l} \quad i \in \{1, 2\} \quad \overline{l}_1, \ldots, \overline{l}_n \preceq_M \overline{l} \]

\[ C := l_1 \lor \cdots \lor l_n \lor D \]

Only change: reason locally in each \( T_i \)
Abstract DPLL Modulo Multiple Theories

**T-Conflict**

\[ C = \text{no} \quad l_1, \ldots, l_n \in M \quad l_1, \ldots, l_n \models_{T_i} \bot \quad i \in \{1, 2\} \]

\[ C := \overline{l}_1 \lor \cdots \lor \overline{l}_n \]

**T-Explain**

\[ C = l \lor D \quad \overline{l}_1, \ldots, \overline{l}_n \models_{T_i} \overline{l} \quad i \in \{1, 2\} \quad \overline{l}_1, \ldots, \overline{l}_n \prec_M \overline{l} \]

\[ C := l_1 \lor \cdots \lor l_n \lor D \]

Only change: reason locally in each \( T_i \)

**I-Learn**

\[ \models_{T_i} l_1 \lor \cdots \lor l_n \quad l_1, \ldots, l_n \in M |_{i \cup \text{I(M)}} \quad i \in \{1, 2\} \]

\[ F := F \cup \{l_1 \lor \cdots \lor l_n\} \]

New rule: for entailed disjunctions of interface literals
Example — Convex Theories

\[ F := \begin{align*}
  & f(e_1) = a \land f(x) = e_2 \land f(y) = e_3 \land f(e_4) = e_5 \land x = y \land \\
  & e_2 - e_3 = e_1 \land e_4 = 0 \land e_5 > a + 2 \\
  & e_2 = e_3 \land e_1 = e_4 \land a = e_5
\end{align*} \]
Example — Convex Theories

\[ F := f(e_1) = a \land f(x) = e_2 \land f(y) = e_3 \land f(e_4) = e_5 \land x = y \land \\
\begin{align*}
e_2 - e_3 &= e_1 \\
e_4 &= 0 \\
e_5 &= a + 2 \\
e_2 &= e_3 \\
e_1 &= e_4 \\
a &= e_5
\end{align*} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>F</td>
<td>no</td>
<td>//</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8</td>
<td>F</td>
<td>no</td>
<td>by Propagate$^+$</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>F</td>
<td>no</td>
<td>by T-Propagate (1, 2, 4 \models_{\text{EUF}} 8)</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>F</td>
<td>no</td>
<td>by T-Propagate (5, 6, 8 \models_{\text{LRA}} 9)</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>F</td>
<td>\overline{7} \lor \overline{10}</td>
<td>by T-Conflict (7, 10 \models_{\text{LRA}} \bot)</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>F</td>
<td>fail</td>
<td>by Fail</td>
</tr>
</tbody>
</table>
Example — Non-convex Theories

\[ F := \begin{align*}
&\text{0: } f(e_1) = a \land f(x) = b \land f(e_2) = e_3 \land f(e_1) = e_4 \\
&\text{1: } 1 \leq x \land x \leq 2 \land e_1 = 1 \land a = b + 2 \land e_2 = 2 \land e_3 = e_4 + 3 \\
&\text{4: } a = e_4 \\
&\text{5: } x = e_1 \\
&\text{6: } x = e_2 \\
&\text{7: } a = b \\
&\text{10: } a = e_4 \\
&\text{11: } x = e_1 \\
&\text{12: } x = e_2 \\
&\text{13: } a = b
\end{align*} \]
Example — Non-convex Theories

\[ F := \begin{align*}
0 & : f(e_1) = a \land f(x) = b \land f(e_2) = e_3 \land f(e_1) = e_4 \\
1 & : 1 \leq x \land x \leq 2 \land e_1 = 1 \land a = b + 2 \land e_2 = 2 \land e_3 = e_4 + 3 \\
2 & : a = e_4 \land x = e_1 \\
3 & : x = e_2 \\
4 & : a = b
\end{align*} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(F)</td>
<td>no</td>
<td>//</td>
</tr>
<tr>
<td>1</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate(^+)</td>
</tr>
<tr>
<td>2</td>
<td>(F)</td>
<td>no</td>
<td>by (T)-Propagate ((0, 3 \models_{\text{EUF}} 10))</td>
</tr>
<tr>
<td>3</td>
<td>(F)</td>
<td>no</td>
<td>by I-Learn ((\models_{\text{LIA}} 4 \lor 5 \lor 11 \lor 12))</td>
</tr>
<tr>
<td>4</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>5</td>
<td>(F)</td>
<td>no</td>
<td>by (T)-Propagate ((0, 1, 11 \models_{\text{EUF}} 13))</td>
</tr>
<tr>
<td>6</td>
<td>(F)</td>
<td>no</td>
<td>by (T)-Conflict ((7, 13 \models_{\text{EUF}} \bot))</td>
</tr>
<tr>
<td>7</td>
<td>(F)</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>8</td>
<td>(F)</td>
<td>no</td>
<td>by (T)-Propagate ((0, 1, 13 \models_{\text{EUF}} 11))</td>
</tr>
<tr>
<td>9</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate (exercise)</td>
</tr>
<tr>
<td>10</td>
<td>(F)</td>
<td>no</td>
<td>by Fail</td>
</tr>
<tr>
<td>11</td>
<td>(F)</td>
<td>no</td>
<td>by (T)-Propagate ((0, 1, 11 \models_{\text{EUF}} 11))</td>
</tr>
<tr>
<td>12</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate (exercise)</td>
</tr>
<tr>
<td>13</td>
<td>(F)</td>
<td>no</td>
<td>by Fail</td>
</tr>
</tbody>
</table>

Sat/SMT Summer School 2012

Foundations of Lazy SMT and DPLL(\(T\)) – p.73/86
Suggested Readings


References


References


References


References


References


References


References


References


References


References


References


References
