Guessing Game

I’m thinking of a research area where:

• Algorithms have recently improved by *orders of magnitude*
• Computers solve tasks *better than* humans
• Computers solve tasks *without help* from humans
• *Big investments* are being made by the government and industry, including companies like:
  • Amazon, Apple, Facebook, Google, Intel, Microsoft
• It can be described using two letters; first one is A
I’m thinking of a research area where:

- Algorithms have recently improved by \textit{orders of magnitude}
- Computers solve tasks \textit{better than} humans
- Computers solve tasks \textit{without help} from humans
- \textit{Big investments} are being made by the government and industry, including companies like:
  - Amazon, Apple, Facebook, Google, Intel, Microsoft
- It can be described using two letters; first one is A
I’m thinking of a research area where:

- Algorithms have recently improved by *orders of magnitude*
- Computers solve tasks *better than* humans
- Computers solve tasks *without help* from humans
- *Big investments* are being made by the government and industry, including companies like:
  - Amazon, Apple, Facebook, Google, Intel, Microsoft
- It can be described using two letters; first one is A
Guessing Game

I’m thinking of a research area where:

• Algorithms have recently improved by *orders of magnitude*
• Computers solve tasks *better than* humans
• Computers solve tasks *without help* from humans
• *Big investments* are being made by the government and industry, including companies like:
  • Amazon, Apple, Facebook, Google, Intel, Microsoft
• It can be described using two letters; first one is A
I’m thinking of a research area where:

- Algorithms have recently improved by *orders of magnitude*
- Computers solve tasks *better than* humans
- Computers solve tasks *without help* from humans
- *Big investments* are being made by the government and industry, including companies like:
  - Amazon, Apple, Facebook, Google, Intel, Microsoft
- It can be described using two letters; first one is A
I’m thinking of a research area where:

- Algorithms have recently improved by *orders of magnitude*
- Computers solve tasks *better than* humans
- Computers solve tasks *without help* from humans
- *Big investments* are being made by the government and industry, including companies like:
  - Amazon, Apple, Facebook, Google, Intel, Microsoft
- It can be described using two letters; first one is A
Philosophers have long dreamed of machines that can reason. The pursuit of this dream has occupied some of the best minds and led both to great achievements and great disappointments.
~1700
Leibniz – mechanized human reasoning

1928
Hilbert
Entscheidungsproblem

1936
Church – lambda calculus
Turing – reduction halting problem

1954
Davis – decision procedure for Presburger arithmetic
Automated Reasoning: A Failure?

- At the turn of the century, automated reasoning was still considered by many to be impractical for most real-world applications.
- Interesting problems appeared to be beyond the reach of automated methods because of decidability and complexity barriers.
- The dream of Hilbert’s mechanized mathematics or Leibniz’s calculating machine was believed by many to be simply unattainable.
Princeton, c. 2000

- *Chaff SAT solver*: orders of magnitude faster than previous SAT solvers
- *Important observation*: many real-world problems do not exhibit worst-case theoretical performance

Palo Alto, c. 2001

- *Idea*: combine fast new SAT solvers with decision procedures for decidable first-order theories
- *SVC, CVC* solvers (Stanford); *ICS, Yices* solvers (SRI)
- *Satisfiability Modulo Theories* (SMT) was born
SMT solvers: *general-purpose* logic engines

- Given condition $X$, is it possible for $Y$ to happen
- $X$ and $Y$ are expressed in a *rich logical language*
  - First-order logic
  - Domain-specific reasoning
    - arithmetic, arrays, bit-vectors, data types, etc.

SMT solvers are *changing the way people solve problems*

- Instead of building a *special-purpose* solver
- *Translate* into a logical formula and use an SMT solver
- Not only easier, *often better*
SMT solvers: *general-purpose* logic engines

- Given condition $X$, is it possible for $Y$ to happen
- $X$ and $Y$ are expressed in a *rich logical language*
  - First-order logic
  - Domain-specific reasoning
    - arithmetic, arrays, bit-vectors, data types, etc.

SMT solvers are *changing the way people solve problems*

- Instead of building a *special-purpose* solver
- *Translate* into a logical formula and use an SMT solver
- Not only easier, *often better*
Automated Reasoning

Evolution of SMT solving

- Total time on QF_BV benchmarks (virtual best)
  - Average speedup: 11X
  - Unsolved (2010): 3100
    - All but 200 solved now
    - Over 2000 now solved in less than 1 second
Zelkova

Security Policy

(allow, principal : *, action : getObject, resource : cs240/*, condition : (StringEquals, aws:sourceVpc, vpc-111bbb222), (StringLike, s3:prefix, cs240/Exam*))

SMT Encoding

a = “getObject” \& r = “cs240/*” \& vpcExists \& vpc = “vpc-111bbb222” \& s3PrefixExists \& “cs240/Exam” prefixOf s3Prefix

SMT Solvers (cvc5 and z3)

- Strings and RegExp
- Bitvectors
- Arithmetic
Satisfiability Modulo Theories

Clark Barrett, Stanford University
CS 243, March 4, 2024
Acknowledgments: Many thanks to Cesare Tinelli and Albert Oliveras for contributing some of the material used in these slides.

Disclaimer: The literature on SMT and its applications is vast. The bibliographic references provided here are just a sample. Apologies to all authors whose work is not cited.
Introduction
Logic Basics: Syntax

Logical formulas are built out of variables, which, like in high school algebra, are placeholders for unknown values. The possible values for a variable are determined by its sort (like a type). Common sorts include Bool, Int, Real, Array, Bitvec, etc.

A term is either a variable or a function applied to zero or more terms.

A theory defines the set of sorts and functions allowed when building terms. We assume that every theory allows the equality symbol $\equiv$, which takes two terms of the same sort and returns a Bool.

Formulas are built by connecting atoms with Boolean operators like $\neg$ (negation), $\land$ (conjunction), $\lor$ (disjunction).

An atom is a term whose sort is Bool, and a literal is either an atom or the negation of an atom. A formula made up of a disjunction of literals is called a clause.
Logic Basics: Syntax

Logical formulas are built out of variables, which, like in high school algebra, are placeholders for unknown values. The possible values for a variable are determined by its sort (like a type). Common sorts include Bool, Int, Real, Array, Bitvec, etc.

A term is either a variable or a function applied to zero or more terms. A theory defines the set of sorts and functions allowed when building terms. We assume that every theory allows the equality symbol, which takes two terms of the same sort and returns a Bool.

Formulas are built by connecting atoms with Boolean operators like (negation), \(^\wedge\) (conjunction), \(^\vee\) (disjunction).

An atom is a term whose sort is Bool, and a literal is either an atom or the negation of an atom. A formula made up of a disjunction of literals is called a clause.
Logic Basics: Syntax

**Examples**

- $v, w, x, y$
  - variables
- $x = y$
  - an atom (the equality function applied to two variables, $x$ and $y$)
- $v + w$
  - a non-atom, the + function applied to two variables $v$ and $w$; could return Int or Real, depending on the theory being used
- $f(x)$
  - an *uninterpreted* function—a function allowed by the theory but not required to be any particular function
- $x = y \lor \neg x = y$
  - a clause, the disjunction of two literals
An *interpretation* $I$ (also called a *structure* or *model*) gives meaning to terms and formulas.

Each *sort* $\sigma$ is assigned a set $\sigma^I$, the *domain* for that sort.

Each variable $v$ is assigned a value $v^I$ in $\sigma^I$, where $\sigma$ is the sort of $v$.

Each function symbol $f$ is assigned to an actual function over the relevant domains. Equality is always assigned to the identity relation.

Given an interpretation $I$, every term $t$ can be assigned a value $t^I$ from the domain of its sort.

In addition to defining the syntax, a theory defines which interpretations are allowed.
A theory that allows no function symbols and has only the sort **Bool**, interpreted as the domain \{true, false\}, is equivalent to *propositional logic*.

A simple theory of arithmetic has sorts **Bool** and **Int** and has the following function symbols, defined in the expected way, shown with their sorts:

- 0, 1 : **Int**
- +, × : **Int** × **Int** → **Int**
- =, ≤ : **Int** × **Int** → **Bool**

Suppose \(I\) assigns \(x\) to 0 and \(y\) to 1:

- \((x + y)^I = 1\)
- \((x = y)^I = \text{false}\)
- \((x = y \lor \neg x = y)^I = \text{true}\)
Satisfiability

A formula $\phi$ is *satisfiable in* a theory $T$, or $T$-satisfiable, if there is an interpretation $I$ of $T$ such that $\phi^I = \text{true}$

A formula $\phi$ is *unsatisfiable in* a theory $T$, or $T$-unsatisfiable, if it is not satisfiable in $T$

The *propositional satisfiability* problem (or SAT problem) is to determine whether a formula $\phi$ is satisfiable in propositional logic.

The *satisfiability modulo theories* problem (or SMT problem) is to determine whether a formula $\phi$ is satisfiable in some given theory $T$
SMT Solvers

- Arithmetic
- Arrays
- UF
- Bit-Vectors

Core

- explanations
- conflicts
- lemmas
- propagations

assertions

SAT Solver

$DPLL$
SMT Solvers

SAT Solver
- Only sees *Boolean skeleton* of problem
- Builds partial model by assigning truth values to literals
- Sends these literals to the core as *assertions*
SMT Solvers

Core

• Sends each assertion to the appropriate theory
• Sends deduced literals to other theories/SAT solver
• Handles *theory combination*
Theory Solvers

- Decide $T$-satisfiability of a conjunction of theory literals
- Incremental
- Backtrackable
- Conflict Generation
- Theory Propagation
Theory Solvers
Given a theory $T$, a *Theory Solver* for $T$ takes as input a set $\Phi$ of literals and determines whether $\Phi$ is $T$-satisfiable.

$\Phi$ is $T$-satisfiable iff there is some model $M$ of $T$ such that each formula in $\Phi$ holds in $M$. 
Equality (\(=\)) with **Uninterpreted Functions** \([\text{NO80, BD94, NO07}]\)

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

- non-linear multiplication in arithmetic
- ALUs in circuits

**Example:** The formula

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

is **unsatisfiable**, but no arithmetic reasoning is needed if we **abstract** it to

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

it is **still** unsatisfiable
Theories of Interest: Arithmetic

Very useful, for obvious reasons

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

- **Bounds**: $x \Join k$ with $\Join \in \{<, >, \leq, \geq, =\}$ [BBC+05a]

- **Difference logic**: $x - y \Join k$, with $\Join \in \{<, >, \leq, \geq, =\}$ [NO05, WIGG05, CM06]

- **UTVPI**: $\pm x \pm y \Join k$, with $\Join \in \{<, >, \leq, \geq, =\}$ [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, JdM12]
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:

- $\forall a \forall i \forall v \ read(\ write(a, i, v), i) = v$
- $\forall a \forall i \forall j \forall v \ i \neq j \rightarrow \ read(\ write(a, i, v), j) = \ read(a, j)$

Sometimes also with extensionality:

- $\forall a \forall b \ (\forall i \ read(a, i) = \ read(b, i) \rightarrow a = b)$

Is the following set of literals satisfiable in this theory?

write(a, i, x) $\neq$ b, read(b, i) = y, read(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, HBJ+14]

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?

\[ a[1 : 0] \neq b[1 : 0] \land (a \mid b) = c \land c[0] = 0 \land a[1] + b[1] = 0 \]
We consider a simple example: difference logic.

In difference logic, we are interested in the satisfiability of a conjunction of arithmetic atoms.

Each atom is of the form $x - y \Join c$, where $x$ and $y$ are variables, $c$ is a numeric constant, and $\Join \in \{=, <, \leq, >, \geq\}$.

The variables can range over either the integers (QF_IDL) or the reals (QF_RDL).
The first step is to rewrite everything in terms of $\leq$: 
The first step is to rewrite everything in terms of $\leq$:

- $x - y = c \implies x - y \leq c \land x - y \geq c$
The first step is to rewrite everything in terms of $\leq$:

- $x - y = c \implies x - y \leq c \land x - y \geq c$
- $x - y \geq c \implies y - x \leq -c$
The first step is to rewrite everything in terms of $\leq$:

- $x - y = c \implies x - y \leq c \land x - y \geq c$
- $x - y \geq c \implies y - x \leq -c$
- $x - y > c \implies y - x < -c$
The first step is to rewrite everything in terms of $\leq$:

- $x - y = c \implies x - y \leq c \land x - y \geq c$
- $x - y \geq c \implies y - x \leq -c$
- $x - y > c \implies y - x < -c$
- $x - y < c \implies x - y \leq c - 1$ (integers)
The first step is to rewrite everything in terms of ≤:

- \( x - y = c \) \( \implies \) \( x - y \leq c \land x - y \geq c \)
- \( x - y \geq c \) \( \implies \) \( y - x \leq -c \)
- \( x - y > c \) \( \implies \) \( y - x < -c \)
- \( x - y < c \) \( \implies \) \( x - y \leq c - 1 \) (integers)
- \( x - y < c \) \( \implies \) \( x - y \leq c - \delta \) (reals)
Now we have a conjunction of literals, all of the form \( x - y \leq c \).

From these literals, we form a weighted directed graph with a vertex for each variable.

For each literal \( x - y \leq c \), there is an edge \( x \xrightarrow{c} y \).

The set of literals is satisfiable iff there is no cycle for which the sum of the weights on the edges is negative.

There are a number of efficient algorithms for detecting negative cycles in graphs.
Difference Logic Example

\[ x - y = 5 \land z - y \geq 2 \land z - x > 2 \land w - x = 2 \land z - w < 0 \]
\begin{align*}
  x - y &= 5 \\
  z - y &\geq 2 \\
  z - x &> 2 \\
  w - x &= 2 \\
  z - w &< 0
\end{align*}
\[ x - y = 5 \land z - y \geq 2 \land z - x > 2 \land w - x = 2 \land z - w < 0 \]
Difference Logic Example

\[
x - y = 5 \land z - y \geq 2 \land z - x > 2 \land w - x = 2 \land z - w < 0
\]

\[
\begin{align*}
x - y &= 5 & x - y &\leq 5 \land y - x \leq -5 \\
z - y &\geq 2 & y - z &\leq -2 \\
z - x &> 2 & x - z &\leq -3 \\
w - x &= 2 & w - x &\leq 2 \land x - w \leq -2 \\
z - w &< 0 & z - w &\leq -1
\end{align*}
\]
Difference Logic Example

![Diagram showing a cycle with numbers and arrows connecting them. The numbers are 5, -2, -3, -1, 2, -2, and -5. The cycle starts at 5 and moves through -2 to -3, -1, 2, -2, and back to 5.]

DPLL($T$): Combining $T$-Solvers with SAT
**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 into *disjunctive normal form* (DNF) and check if any of its disjuncts is \(T\)-satisfiable)

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

**Solution:** Exploit propositional satisfiability technology
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 into *disjunctive normal form* (DNF) and check if any of its disjuncts is $T$-satisfiable.)

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

Solution: Exploit propositional satisfiability technology.
**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 into *disjunctive normal form* (DNF) and check if any of its disjuncts is \(T\)-satisfiable)

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

**Solution:** Exploit propositional satisfiability technology.
**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 into *disjunctive normal form* (DNF) and check if any of its disjuncts is \(T\)-satisfiable)

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

**Solution:** Exploit propositional satisfiability technology.
**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 into *disjunctive normal form* (DNF) and check if any of its disjuncts is $T$-satisfiable)

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

**Solution:** Exploit propositional satisfiability technology.
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**

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: **cvc5, MathSAT, OpenSMT, SMTInterpol, Yices, Z3**

This lecture will focus on the lazy approach
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**

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: **cvc5, MathSAT, OpenSMT, SMTInterpol, Yices, Z3**

This lecture will focus on the lazy approach
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**

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: **cvc5, MathSAT, OpenSMT, SMTInterpol, Yices, Z3**

This lecture will focus on the lazy approach
\[ g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \]

**Theory \( T \): Equality with Uninterpreted Functions**

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., \( \bot \))
(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

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

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

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, 1 \lor 2 \lor 3 \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, 1 \lor 2, 1 \lor 3 \lor 4\} to SAT solver.
6. SAT solver finds \{1, 2 \lor 3, 4, 1 \lor 2, 1 \lor 3 \lor 4\} unsat.
   Done: the original formula is unsatisfiable in UF.
(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 \( \{1, \overline{2}, \overline{4}\} \).
3. Theory solver finds (concretization of) \( \{1, \overline{2}, \overline{4}\} \) unsat.
4. Send \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4\} \) to SAT solver.
5. SAT solver returns model \( \{1, 3, \overline{4}\} \).
6. Theory solver finds \( \{1, 3, \overline{4}\} \) unsat.
7. Send \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2, \overline{1} \lor 3 \lor 4\} \) to SAT solver.
8. SAT solver finds \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4\} \) unsat.

Done: the original formula is unsatisfiable in UF.
(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.
5. Send \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2, \overline{1} \lor 3 \lor 4\} \) to SAT solver.
6. SAT solver finds \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4\} \) unsat.

Done: the original formula is unsatisfiable in UF.
(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 \( \{1, 2 \lor 3, 4\} \) 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.
   SAT solver returns model \( \{1, 3, 4\} \).
   Theory solver finds \( \{1, 3, 4\} \) unsat.
4. Send \( \{1, 2 \lor 3, 4, \overline{1} \lor 2, \overline{1} \lor 3 \lor 4\} \) to SAT solver.
   SAT solver finds \( \{1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4\} \) unsat.

Done: the original formula is unsatisfiable in UF.
(Very) Lazy Approach for SMT – Example

\[ g(a) = c \quad \land \quad \begin{cases} f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d \\ 1 \quad \frac{1}{2} \quad 3 \quad \frac{4}{4} \end{cases} \]

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

Done: the original formula is unsatisfiable in UF.
(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.
- 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 UF.
(Very) Lazy Approach for SMT – Example

\[ \begin{align*}
  g(a) & = c \land \left( f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \right) \\
  & \quad \text{(1)} \\
  & \quad \text{(2)} \\
  & \quad \text{(3)} \\
  & \quad \text{(4)}
\end{align*} \]

- 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 UF.
(Very) Lazy Approach for SMT – Example

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

- 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 UF.
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, identify a $T$-unsatisfiable subset $M_0$ of $M$ and add $\neg M_0$ as a clause
- If $M$ is $T$-unsatisfiable, backtrack to some point where the assignment was still $T$-satisfiable
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, identify a $T$-unsatisfiable subset $M_0$ of $M$ and add $\neg M_0$ as a clause
  - If $M$ is $T$-unsatisfiable, backtrack to some point where the assignment was still $T$-satisfiable
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, backtrack to some point where the assignment was still $T$-satisfiable
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, backtrack to some point where the assignment was still $T$-satisfiable
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, backtrack to some point where the assignment was still $T$-satisfiable
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, backtrack to some point where the assignment was still $T$-satisfiable
Lazy Approach – Main Benefits

• Every tool does what it is good at:
  • SAT solver takes care of Boolean information
  • Theory solver takes care of theory information

• The theory solver works only with conjunctions of literals

• Modular approach:
  • SAT and theory solvers communicate via a simple API [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)
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)
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)
Several variants and enhancements of lazy SMT solvers exist.

They can be modeled abstractly and declaratively as transition systems.

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

The framework can be first developed for SAT and then extended to lazy SMT [NOT06, KG07].
Advantages of Abstract Framework

An abstract framework helps one:

• skip over implementation details and unimportant control aspects
• reason formally about solvers for SAT and SMT
• model advanced features such as non-chronological 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]
Advantages of Abstract Framework

An abstract framework helps one:

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

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

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

• DPLL tries to build incrementally a satisfying truth assignment $M$ for a CNF formula $F$

• $M$ is grown by
  • deducing the truth value of a literal from $M$ and $F$, or
  • guessing a truth value

• If a wrong guess for a literal leads to an inconsistency, the procedure backtracks and tries the opposite value
States:

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

where

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

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

- \( M_i \) is decision level \( i \) of \( M \)
- \( M^{[i]} \coloneqq M_0 \bullet \cdots \bullet M_i \)
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:

- \( \text{fail} \) 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]

\[
\begin{array}{c}
\frac{p_1 \quad \cdots \quad p_n}{[M := e_1] \quad [F := e_2]}
\end{array}
\]

updating $M$, $F$ or both when premises $p_1, \ldots, p_n$ all hold
Transition Rules for the Original DPLL

Extending the assignment

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

\[ M := M \setminus l \]

**Note:** When convenient, treat \( M \) as a set

**Note:** Clauses are treated modulo ACI of \( \lor \)

**Decide** \[ l \in \text{Lit}(F) \; l, \bar{l} \notin M \]

\[ M := M \setminus l \]

**Note:** \( \text{Lit}(F) \stackrel{\text{def}}{=} \{l \mid l \text{ literal of } F\} \cup \{\bar{l} \mid l \text{ literal of } F\} \)
Transition Rules for the Original DPLL

Extending the assignment

**Propagate**

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

**Note:** When convenient, treat \( M \) as a set

**Note:** Clauses are treated modulo ACI of \( \lor \)

**Decide**

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

**Note:** \( \text{Lit}(F) := \left\{ l \mid l \text{ literal of } F \right\} \cup \left\{ \bar{l} \mid l \text{ literal of } F \right\} \)
Transition Rules for the Original DPLL

Repairing the assignment

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

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

\text{Note:} \text{ Last premise of Backtrack enforces chronological backtracking}
Transition Rules for the Original DPLL

Repairing the assignment

**Fail**
\[
\begin{align*}
l_1 \lor \cdots \lor l_n &\in F \\
\bar{l}_1, \ldots, \bar{l}_n &\in M \\
\bullet &\notin M
\end{align*}
\]

\[
\text{fail}
\]

**Backtrack**
\[
\begin{align*}
l_1 \lor \cdots \lor l_n &\in F \\
\bar{l}_1, \ldots, \bar{l}_n &\in M \\
M &\equiv M \cdot l \\
\bullet &\notin N
\end{align*}
\]

\[
M := M \bar{l}
\]

**Note:** Last premise of **Backtrack** enforces **chronological** backtracking
To model conflict-driven backjumping and learning, add to states a third component $C$ whose value is either no or a conflict clause.

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

Initial state:

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

Expected final states:

- fail if $F_0$ is unsatisfiable
- $\langle M, G, \text{no} \rangle$ otherwise, where
  - $G$ is equivalent to $F_0$ and
  - $M$ satisfies $G$
To model conflict-driven backjumping and learning, add to states a third component \( C \) whose value is either \textbf{no} or a \textit{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 \( G \)
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**

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

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

**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 \( C \neq \text{no} \)

**Note:** \( \models_p \) denotes propositional entailment
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**

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

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

**Backjump**

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

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

**Note:** \( l <_M l' \) if \( l \) occurs before \( l' \) in \( M \)

\( \text{lev } l = i \) iff \( l \) occurs in decision level \( i \) of \( M \)

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

**Note:** \( \models_p \) denotes propositional entailment
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**

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

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

**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 \( C \neq \text{no} \)

**Note:** \( \models_p \) denotes propositional entailment
Modify \textbf{Fail} to

\[
\text{Fail} \quad \begin{array}{c}
C \neq \text{no} \\
\not\in M
\end{array} \\
\text{fail}
\]
Modify **Fail** to

\[
\text{Fail} \quad \text{\underline{C \not\equiv \text{no}}} \quad \bullet \not\in M \\
\underline{\text{fail}}
\]
Execution Example

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

<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>by Propagate</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 Decide</td>
</tr>
<tr>
<td>1 2 3</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>( F )</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 3 4 5</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 8</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 6 7 8</td>
<td>( F )</td>
<td>( \overline{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 6 7 8</td>
<td>( F )</td>
<td>( \overline{1} \lor \overline{2} \lor \overline{5} )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 5</td>
<td>( F )</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>1 2 5 3</td>
<td>( F )</td>
<td>no</td>
<td>by Decide</td>
</tr>
</tbody>
</table>
Execution Example

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

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

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

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

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>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>2 \lor 5 \lor 6 \lor 7</td>
<td>by Conflict</td>
</tr>
<tr>
<td>1 2 • 3 4 • 5 6 7</td>
<td>F</td>
<td>\overline{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 6 7</td>
<td>F</td>
<td>\overline{1} \lor \overline{2} \lor \overline{5}</td>
<td>by Explain with \overline{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 5 • 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>
$F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, \overline{2} \lor \overline{5} \lor 6 \lor \overline{7}\}$

<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>by <strong>Propagate</strong></td>
</tr>
<tr>
<td>1</td>
<td>$F$</td>
<td>no</td>
<td>by <strong>Propagate</strong></td>
</tr>
<tr>
<td>1 2</td>
<td>$F$</td>
<td>no</td>
<td>by <strong>Decide</strong></td>
</tr>
<tr>
<td>1 2 3</td>
<td>$F$</td>
<td>no</td>
<td>by <strong>Propagate</strong></td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>$F$</td>
<td>no</td>
<td>by <strong>Decide</strong></td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>$F$</td>
<td>no</td>
<td>by <strong>Propagate</strong></td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>$F$</td>
<td>no</td>
<td>by <strong>Propagate</strong></td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>$F$</td>
<td>no</td>
<td>by <strong>Propagate</strong></td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>$F$</td>
<td>no</td>
<td>by <strong>Conflict</strong></td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>$F$</td>
<td>$2 \lor \overline{5} \lor 6 \lor \overline{7}$</td>
<td>by <strong>Explain with</strong> $\overline{1} \lor \overline{5} \lor 7$</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>$F$</td>
<td>$\overline{1} \lor \overline{2} \lor 5 \lor 6$</td>
<td>by <strong>Explain with</strong> $\overline{5} \lor 6$</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>$F$</td>
<td>no</td>
<td>by <strong>Backjump</strong></td>
</tr>
<tr>
<td>1 2 5 3</td>
<td>$F$</td>
<td>no</td>
<td>by <strong>Decide</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
$F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, \overline{2} \lor \overline{5} \lor 6 \lor \overline{7}\}$

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

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

<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>by Propagate</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 Propagate</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 Propagate</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>\overline{2} \lor \overline{5} \lor 6 \lor 7</td>
<td>Conflict</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>(F)</td>
<td>\overline{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 6 7</td>
<td>(F)</td>
<td>\overline{1} \lor \overline{2} \lor \overline{5}</td>
<td>by Explain with (\overline{5} \lor 6)</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>(F)</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>1 2 5</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 5 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>
$F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, \overline{2} \lor \overline{5} \lor 6 \lor 7\}$

<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>by Propagate</td>
</tr>
<tr>
<td>1</td>
<td>$F$</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>$\overline{2} \lor 5 \lor 6 \lor 7$</td>
<td>no</td>
<td>by Conflict</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>$\overline{1} \lor \overline{2} \lor 5 \lor 6$</td>
<td>no</td>
<td>by Explain with $\overline{1} \lor 5 \lor 7$</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>$\overline{1} \lor 2 \lor 5$</td>
<td>no</td>
<td>by Explain with $5 \lor 6$</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 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>(\overline{2} \lor \overline{5} \lor 6 \lor 7)</td>
<td></td>
<td>by Conflict</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>(\overline{1} \lor \overline{2} \lor \overline{5} \lor 6)</td>
<td></td>
<td>by Explain with (\overline{1} \lor \overline{5} \lor 7)</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>(\overline{1} \lor \overline{2} \lor \overline{5})</td>
<td></td>
<td>by Explain with (\overline{5} \lor 6)</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>(\overline{3} \lor \overline{4} \lor \overline{5} \lor \overline{6} \lor \overline{7})</td>
<td></td>
<td>by Explain with (\overline{3} \lor \overline{4} \lor \overline{5} \lor \overline{6} \lor \overline{7})</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8 9</td>
<td>(\overline{1} \lor \overline{2} \lor \overline{3} \lor \overline{4} \lor \overline{5} \lor \overline{6} \lor \overline{7} \lor \overline{8} \lor \overline{9})</td>
<td></td>
<td>by Explain with (\overline{1} \lor \overline{2} \lor \overline{3} \lor \overline{4} \lor \overline{5} \lor \overline{6} \lor \overline{7} \lor \overline{8} \lor \overline{9})</td>
</tr>
</tbody>
</table>

\[ \ldots \]

46
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>by Propagate</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 Decide</td>
</tr>
<tr>
<td>1 2 3</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>(F)</td>
<td>no</td>
<td>by Propagate</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>(\overline{2} \lor \overline{5} \lor 6 \lor 7)</td>
<td>by Conflict</td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>(\overline{1} \lor \overline{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 3 4 5 6 7</td>
<td>(\overline{1} \lor \overline{2} \lor \overline{5})</td>
<td>by Explain with (\overline{5} \lor 6)</td>
<td></td>
</tr>
<tr>
<td>1 2 5 7</td>
<td>(F)</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>1 2 5 7 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>
Also add

\[
\begin{align*}
\text{Learn} & \quad \frac{F \models_p C \quad C \not\models F}{F := F \cup \{C\}} \\
\text{Forget} & \quad \frac{C = \text{no} \quad F = G \cup \{C\} \quad G \models_p C}{F := G}
\end{align*}
\]

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

**Note:** Learn can be applied to any clause stored in \(C\) when \(C \not\models \text{no}\)
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**

\[
\text{Basic DPLL} \overset{\text{def}}{=} \{ \text{Propagate, Decide, Conflict, Explain, Backjump} \}
\]

\[
\text{DPLL} \overset{\text{def}}{=} \text{Basic DPLL} + \{ \text{Learn, Forget, Restart} \}
\]
At the core, current CDCL SAT solvers are implementations of the transition system with rules

**Propagate, Decide,**

**Conflict, Explain, Backjump,**

**Learn, Forget, Restart**

\[
\text{Basic DPLL} \overset{\text{def}}{=} \{ \text{Propagate, Decide, Conflict, Explain, Backjump} \}
\]

\[
\text{DPLL} \overset{\text{def}}{=} \text{Basic DPLL} + \{ \text{Learn, Forget, Restart} \}
\]
Some terminology:

**Irreducible state:** state for which no Basic DPLL rules apply

**Execution:** sequence of transitions allowed by the rules and starting with $M = \emptyset$ 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 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 $M$. 
Some terminology:

*Irreducible state:* state for which no Basic DPLL rules apply

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

*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.*

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

**Proposition (Completeness)** For every exhausted execution starting
The Basic DPLL System – Correctness

Some terminology:

**Irreducible state**: state for which no Basic DPLL rules apply

**Execution**: sequence of transitions allowed by the rules and starting with $M = \emptyset$ 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.

**Proposition** *(Soundness)* For every exhausted execution starting with $F = F_0$ and ending with 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 $M$. 
Some terminology:

**Irreducible state:** state for which no Basic DPLL rules apply

**Execution:** sequence of transitions allowed by the rules and starting with \( M = \emptyset \) 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 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

• 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 – Strategies

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

ensures termination

• 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 – Strategies

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

ensures termination

• 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 – Strategies

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

ensures termination

• 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 – Strategies

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

ensures termination

• 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 – Strategies

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

ensures termination

• 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 – Strategies

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

ensures termination

• 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 – Strategies

• Applying
  • one Basic DPLL rule between each two Learn applications and
  • Restart less and less often
  
ensures termination

• 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
From SAT to SMT

Same 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 is 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

\[ \begin{align*}
C &= \text{no} \quad l_1, \ldots, l_n \in M \\
&\quad l_1, \ldots, l_n \models_T \bot \\
C &:= \bar{l}_1 \lor \cdots \lor \bar{l}_n 
\end{align*} \]

$T$-Propagate

\[ \begin{align*}
&l \in \text{Lit}(F) \\
&M \models_T l \\
&l, \bar{l} \notin M
\end{align*} \]

\[ M := M \downarrow \]

$T$-Explain

\[ \begin{align*}
&C = l \lor D \\
&\bar{l}_1, \ldots, \bar{l}_n \models_T \bar{l} \\
&\bar{l}_1, \ldots, \bar{l}_n \prec_M \bar{l}
\end{align*} \]

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

Note: $\bot = \text{empty clause}$

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

Fix a theory $T$

**$T$-Conflict**

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

$C := \bar{l}_1 \lor \cdots \lor \bar{l}_n$

**$T$-Propagate**

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

$M := M \: l$

**$T$-Explain**

$C = l \lor D$ \quad $\bar{l}_1, \ldots, \bar{l}_n \models_T \bar{l}$ \quad $\bar{l}_1, \ldots, \bar{l}_n <_M \bar{l}$

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

**Note:** $\bot = \text{empty clause}$

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

Fix a theory $T$

$T$-Conflict

\[ C = \text{no} \quad l_1, \ldots, l_n \in M \quad l_1, \ldots, l_n \models_T \bot \]

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

$T$-Propagate

\[ l \in \text{Lit}(F) \quad M \models_T l \quad l, \bar{l} \notin M \]

\[ M := M \ l \]

$T$-Explain

\[ C = l \lor D \quad \bar{l}_1, \ldots, \bar{l}_n \models_T \bar{l} \quad \bar{l}_1, \ldots, \bar{l}_n \prec_M \bar{l} \]

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

Note: $\bot = \text{empty clause}$

Note: $\models_T$ decided by theory solver
**T-Conflict** is enough to model the naive integration of SAT solvers and theory solvers seen in the earlier UF 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></td>
<td></td>
<td>no</td>
<td>by Propagate +</td>
</tr>
<tr>
<td>(\frac{1}{4})</td>
<td>(\frac{1}{2})</td>
<td>(1 \lor 2 \lor 3 \lor 4)</td>
<td>by Decide</td>
</tr>
<tr>
<td>(\frac{1}{4})</td>
<td>(\frac{3}{2})</td>
<td>no</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>(\frac{1}{4})</td>
<td>(\frac{3}{2})</td>
<td>(\frac{1}{2} \lor 2 \lor 4)</td>
<td>by Learn</td>
</tr>
<tr>
<td>(\frac{1}{4})</td>
<td>(\frac{3}{2})</td>
<td>(\frac{1}{2} \lor 2 \lor 4)</td>
<td>by Restart</td>
</tr>
<tr>
<td>(\frac{1}{4})</td>
<td>(\frac{3}{2})</td>
<td>no</td>
<td>by Propagate +</td>
</tr>
<tr>
<td>(\frac{1}{4})</td>
<td>(\frac{3}{2})</td>
<td>(\frac{1}{2} \lor 2 \lor 4)</td>
<td>by T-Conflict, Learn</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td>no</td>
<td>by Fail</td>
</tr>
</tbody>
</table>
Modeling the Very Lazy Theory Approach

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

<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>no</td>
<td>by Propagate +</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>no</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>no</td>
<td>by Learn</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>no</td>
<td>by Restart</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>no</td>
<td>by Propagate +</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>no</td>
<td>by T-Conflict, Learn</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td>by Fail</td>
</tr>
</tbody>
</table>
Modeling the Very Lazy Theory Approach

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

\[
\begin{array}{cccc}
\text{M} & \text{F} & \text{C} & \text{rule} \\
1 & 2 \lor 3, 4 & \text{no} & \\
1 & 4 & 1, 2 \lor 3, 4 & \text{no} & \text{by Propagate}^+ \\
1 & 4 \bullet 2 & 1, 2 \lor 3, 4 & \text{no} & \text{by Decide} \\
1 & 4 \bullet 2 & 1, 2 \lor 3, 4 & 1 \lor 2 \lor 4 & \text{by } T\text{-Conflict} \\
1 & 4 \bullet 2 & 1, 2 \lor 3, 4, 1 \lor 2 \lor 4 & \text{no} & \text{by Learn} \\
1 & 4 & 1, 2 \lor 3, 4, 1 \lor 2 \lor 4 & \text{no} & \text{by Restart} \\
1 & 4 & 2 & 3 & 1 \lor 2 \lor 4, 1 \lor 3 \lor 4 & \text{no} & \text{by Propagate}^+ \\
1 & 4 & 2 & 3 & 1 \lor 2 \lor 4, 1 \lor 3 \lor 4 & 1 \lor 3 \lor 4 & \text{by } T\text{-Conflict, Learn} \\
fail & & & \text{by Fail} \\
\end{array}
\]
Modeling the Very Lazy Theory Approach

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

<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>no</td>
<td>by Propagate+</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1, (\overline{2} \lor 3, \overline{4})</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1, (\overline{2} \lor 3, \overline{4})</td>
<td>by Learn</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1, (\overline{2} \lor 3, \overline{4})</td>
<td>by Restart</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1, (\overline{2} \lor 3, \overline{4})</td>
<td>by Propagate+</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1, (\overline{2} \lor 3, \overline{4})</td>
<td>by T-Conflict, Learn</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td>by Fail</td>
</tr>
</tbody>
</table>
Modeling the Very Lazy Theory Approach

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/4</td>
<td>1/2</td>
<td>no</td>
<td>by Propagate+</td>
</tr>
<tr>
<td>1/2</td>
<td>1/2</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1/2</td>
<td>1/2</td>
<td>no</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>1/2</td>
<td>1/2</td>
<td>no</td>
<td>by Learn</td>
</tr>
<tr>
<td>1/2</td>
<td>1/2</td>
<td>no</td>
<td>by Restart</td>
</tr>
<tr>
<td>1/4</td>
<td>1/2</td>
<td>no</td>
<td>by Propagate+</td>
</tr>
<tr>
<td>1/4</td>
<td>1/2</td>
<td>no</td>
<td>by T-Conflict, Learn</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td>fail</td>
<td>by Fail</td>
</tr>
</tbody>
</table>
Modeling the Very Lazy Theory Approach

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1\ 4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Propagate$^+$</td>
</tr>
<tr>
<td>1\ 4 • 2</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1\ 4 • 2</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>1\ 4 • 2</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4</td>
<td>yes</td>
<td>by Learn</td>
</tr>
<tr>
<td>1\ 4</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4</td>
<td>no</td>
<td>by Restart</td>
</tr>
<tr>
<td>1\ 4 2\ 3</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4</td>
<td>no</td>
<td>by Propagate$^+$</td>
</tr>
<tr>
<td>1\ 4 2\ 3</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 3 \lor 4</td>
<td>yes</td>
<td>by T-Conflict, Learn</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td>fail</td>
<td>by Fail</td>
</tr>
</tbody>
</table>
Modeling the Very Lazy Theory Approach

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

<table>
<thead>
<tr>
<th>(M)</th>
<th>(F)</th>
<th>(C)</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Propagate(^+)</td>
</tr>
<tr>
<td>1 4 * 2</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 4 * 2</td>
<td>1, 2 \lor 3, 4</td>
<td>(\overline{1} \lor 2 \lor 4)</td>
<td>by (T)-Conflict</td>
</tr>
<tr>
<td>1 4 * 2</td>
<td>1, 2 \lor 3, 4, (\overline{1} \lor 2 \lor 4)</td>
<td>(\overline{1} \lor 2 \lor 4)</td>
<td>by Learn</td>
</tr>
<tr>
<td>1 4</td>
<td>1, 2 \lor 3, 4, 1 \lor 2 \lor 4</td>
<td>no</td>
<td>by Restart</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 \lor 3, 4, 1 \lor 2 \lor 4</td>
<td>no</td>
<td>by Propagate(^+)</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 \lor 3, 4, 1 \lor 2 \lor 4, (\overline{1} \lor 3 \lor 4)</td>
<td>(\overline{1} \lor 3 \lor 4)</td>
<td>by (T)-Conflict, Learn</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td>by Fail</td>
</tr>
</tbody>
</table>
Modeling the Very Lazy Theory Approach

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>(1, \overline{2} \lor 3, \overline{4})</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\overline{1} \overline{4})</td>
<td>(1, \overline{2} \lor 3, \overline{4})</td>
<td>no</td>
<td>by Propagate^+</td>
<td></td>
</tr>
<tr>
<td>(1 \overline{4} \bullet 2)</td>
<td>(1, \overline{2} \lor 3, \overline{4})</td>
<td>no</td>
<td>by Decide</td>
<td></td>
</tr>
<tr>
<td>(1 \overline{4} \bullet 2)</td>
<td>(1, \overline{2} \lor 3, \overline{4})</td>
<td>(\overline{1} \lor 2 \lor 4)</td>
<td>by T-Conflict</td>
<td></td>
</tr>
<tr>
<td>(1 \overline{4} \bullet 2)</td>
<td>(1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4)</td>
<td>(\overline{1} \lor 2 \lor 4)</td>
<td>by Learn</td>
<td></td>
</tr>
<tr>
<td>(1 \overline{4})</td>
<td>(1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4)</td>
<td>no</td>
<td>by Restart</td>
<td></td>
</tr>
<tr>
<td>(\overline{1} \overline{4} 2 3)</td>
<td>(1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4)</td>
<td>no</td>
<td>by Propagate^+</td>
<td></td>
</tr>
<tr>
<td>(\overline{1} \overline{4} 2 3)</td>
<td>(1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4)</td>
<td>(\overline{1} \lor 3 \lor 4)</td>
<td>by T-Conflict, Learn</td>
<td></td>
</tr>
<tr>
<td>(\overline{1} \overline{4} 2 3)</td>
<td>(1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4)</td>
<td>(\overline{1} \lor 3 \lor 4)</td>
<td>by Fail</td>
<td></td>
</tr>
</tbody>
</table>
Modeling the Very Lazy Theory Approach

\[
\begin{align*}
\left\{\begin{array}{c}
g(a) = c \\
g(a) = d
\end{array}\right\} & \quad \land \quad \left\{\begin{array}{c}
f(g(a)) \neq f(c) \\
c \neq d
\end{array}\right\} \\
1 & \quad \land \quad 2 & \quad \lor \quad 3 & \quad \land \quad 4
\end{align*}
\]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/4</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by \textbf{Propagate}^+</td>
</tr>
<tr>
<td>1/4</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by \textbf{Propagate}^+</td>
</tr>
<tr>
<td>1/4</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by \textbf{Decide}</td>
</tr>
<tr>
<td>1/4</td>
<td>1, 2 ∨ 3, 4, \overline{1} ∨ 2 ∨ 4</td>
<td>\overline{1} ∨ 2 ∨ 4</td>
<td>by \textbf{T-Conflict}</td>
</tr>
<tr>
<td>1/4</td>
<td>1, 2 ∨ 3, 4, \overline{1} ∨ 2 ∨ 4</td>
<td>\overline{1} ∨ 2 ∨ 4</td>
<td>by \textbf{T-Conflict}</td>
</tr>
<tr>
<td>1/4</td>
<td>1, 2 ∨ 3, 4, \overline{1} ∨ 2 ∨ 4</td>
<td>no</td>
<td>by \textbf{Learn}</td>
</tr>
<tr>
<td>1/4</td>
<td>1, 2 ∨ 3, 4, \overline{1} ∨ 2 ∨ 4</td>
<td>no</td>
<td>by \textbf{Restart}</td>
</tr>
</tbody>
</table>

\textbf{fail}
Modeling the Very Lazy Theory Approach

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Propagate$^+$</td>
</tr>
<tr>
<td>1 4 \cdot 2</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 4 \cdot 2</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by (T)-Conflict</td>
</tr>
<tr>
<td>1 4 \cdot 2</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4</td>
<td>\overline{1} \lor 2 \lor 4</td>
<td>by Learn</td>
</tr>
<tr>
<td>1 4</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4</td>
<td>no</td>
<td>by Restart</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 \lor 3, 4, 1 \lor 2 \lor 4</td>
<td>no</td>
<td>by Propagate$^+$</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4</td>
<td>\overline{1} \lor 3 \lor 4</td>
<td>by (T)-Conflict, Learn</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td>by Fail</td>
</tr>
</tbody>
</table>
Modeling the Very Lazy Theory Approach

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 4</td>
<td>1, 3, 4</td>
<td>no</td>
<td>by \textbf{Propagate}^+</td>
</tr>
<tr>
<td>1, 4 3</td>
<td>1, 2, 3, 4</td>
<td>no</td>
<td>by \textbf{Decide}</td>
</tr>
<tr>
<td>1, 4 3</td>
<td>1, 2, 3, 4</td>
<td>no</td>
<td>by \textbf{T-Conflict}</td>
</tr>
<tr>
<td>1, 4 3</td>
<td>1, 2, 3, 4</td>
<td>no</td>
<td>by \textbf{Learn}</td>
</tr>
<tr>
<td>1, 4 3</td>
<td>1, 2, 3, 4</td>
<td>no</td>
<td>by \textbf{Restart}</td>
</tr>
<tr>
<td>1, 4 3</td>
<td>1, 2, 3, 4</td>
<td>no</td>
<td>by \textbf{Propagate}^+</td>
</tr>
</tbody>
</table>

fail
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

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
The very lazy approach can be improved considerably with

- An \textit{on-line} SAT engine,
  which can accept new input clauses on the fly

- an \textit{incremental and explicating} $T$-solver,
  which can
  \begin{enumerate}
    \item check the $T$-satisfiability of $M$ as it is extended and
    \item identify a small $T$-unsatisfiable subset of $M$ once $M$ becomes $T$-unsatisfiable
  \end{enumerate}
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

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
</table>
| 1  | 4  | 1, \(\overline{2}\lor 3, 4\) | no by **Propagate**  
| 1  | 4  | \(\overline{1}\lor 2\lor 3, 4\) | no by **Decide**  
| 1  | 4  | \(\overline{2}\lor 3, 4\) | \(\overline{1}\lor 2\) by **T-Conflict**  
| 1  | 4  | \(\overline{2}\lor 3, 4\) | no by **Backjump**  
| 1  | 4  | \(\overline{2}\lor 3, 4\) | no by **Propagate**  
| 1  | 4  | \(\overline{2}\lor 3, 4\) | \(\overline{2}\lor 3\lor 4\) by **T-Conflict**  
| fail | | \(\overline{2}\lor 3\lor 4\) | by **Fail** |
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>1, \overline{2} \lor 3, \overline{4}</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>\overline{4} \bullet 2</td>
<td>1, \overline{2} \lor 3, \overline{4}</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>\overline{4} \bullet 2</td>
<td>1, \overline{2} \lor 3, \overline{4}</td>
<td>\overline{1} \lor 2</td>
</tr>
<tr>
<td>1</td>
<td>4 2</td>
<td>1, \overline{2} \lor 3, \overline{4}</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>4 2 3</td>
<td>1, \overline{2} \lor 3, \overline{4}</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>4 2 3</td>
<td>1, \overline{2} \lor 3, \overline{4}</td>
<td>\overline{1} \lor 3 \lor 4</td>
</tr>
<tr>
<td></td>
<td>fail</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 \ 4</td>
<td>1, \ \bar{2} \lor 3, \ \bar{4}</td>
<td>no</td>
<td>by Propagate$^+$</td>
</tr>
<tr>
<td>1 \ \bar{4} \ \lor 2</td>
<td>1, \ \bar{2} \lor 3, \ \bar{4}</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>\bar{1} \ \bar{4} \ \lor 2</td>
<td>1, \ \bar{2} \lor 3, \ \bar{4}</td>
<td>\bar{1} \lor 2</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>\bar{1} \ 4 \ \lor 2</td>
<td>1, \ \bar{2} \lor 3, \ \bar{4}</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>\bar{1} \ 4 \ 2 \ \lor 3</td>
<td>1, \ \bar{2} \lor 3, \ \bar{4}</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>\bar{1} \ \bar{4} \ 2 \ \lor 3</td>
<td>1, \ \bar{2} \lor 3, \ \bar{4}</td>
<td>\bar{1} \lor \bar{3} \lor 4</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td>by Fail</td>
</tr>
</tbody>
</table>
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 (\bar{4})</td>
<td>1, (\bar{2} \lor 3, 4)</td>
<td>no</td>
<td>by Propagate +</td>
</tr>
<tr>
<td>1 (\bar{4} \oplus 2)</td>
<td>1, (\bar{2} \lor 3, 4)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 (\bar{4} \oplus 2)</td>
<td>1, (\bar{2} \lor 3, 4)</td>
<td>(\bar{1} \lor 2)</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>1 (\bar{4} 2)</td>
<td>1, (\bar{2} \lor 3, 4)</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>1 (\bar{4} 2 3)</td>
<td>1, (\bar{2} \lor 3, 4)</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 (\bar{4} 2 3)</td>
<td>1, (\bar{2} \lor 3, 4)</td>
<td>(\bar{1} \lor 3 \lor 4)</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td>by Fail</td>
</tr>
</tbody>
</table>
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 4</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by Propagate $^+$</td>
</tr>
<tr>
<td>1 4 • 2</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 4 • 2</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by $T$-Conflict</td>
</tr>
<tr>
<td>1 4 2</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 ∨ 3, 4</td>
<td>1 ∨ 2</td>
<td>by $T$-Conflict</td>
</tr>
<tr>
<td>fail</td>
<td>1, 2 ∨ 3, 4</td>
<td>1 ∨ 3 ∨ 4</td>
<td>by Fail</td>
</tr>
</tbody>
</table>
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>1, 2 \lor 3, 4</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1, 2 \lor 3, 4</td>
<td>\bar{1} \lor 2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1, 2 \lor 3, 4</td>
<td>\bar{1} \lor 3 \lor 4</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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>2 3 4</td>
<td>no</td>
<td>Propagate</td>
</tr>
<tr>
<td>1 4</td>
<td>2 3 4</td>
<td>no</td>
<td>Decide</td>
</tr>
<tr>
<td>1 4 2</td>
<td>2 3 4</td>
<td>1 2 3 4</td>
<td>T-Conflict</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1 2 3 4</td>
<td>no</td>
<td>Backjump</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1 2 3 4</td>
<td>no</td>
<td>Propagate</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1 2 3 4</td>
<td>T 3 4</td>
<td>T-Conflict</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1 2 3 4</td>
<td>T 3 4</td>
<td>Fail</td>
</tr>
</tbody>
</table>
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 \ 4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Propagate+</td>
</tr>
<tr>
<td>1 \ 4 \bullet \ 2</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 \ 4 \bullet \ 2</td>
<td>1, 2 \lor 3, 4</td>
<td>1 \lor 2</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>1 \ 4 \ 2</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>1 \ 4 \ 2 \ 3</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>1 \ 4 \ 2 \ 3</td>
<td>1, 2 \lor 3, 4</td>
<td>1 \lor 3 \lor 4</td>
<td>by T-Conflict</td>
</tr>
</tbody>
</table>

fail by Fail
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>1, 2 \lor 3, 4</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>4 \bullet 2</td>
<td>1, 2 \lor 3, 4</td>
<td>no by Propagate +</td>
</tr>
<tr>
<td>1</td>
<td>4 \bullet 2</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>1</td>
<td>4 2</td>
<td>1, 2 \lor 3, 4</td>
<td>no by Backjump</td>
</tr>
<tr>
<td>1</td>
<td>4 2 3</td>
<td>1, 2 \lor 3, 4</td>
<td>no by Propagate</td>
</tr>
<tr>
<td>1</td>
<td>4 2 3</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 3 \lor 4</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td>1, 2 \lor 3, 4</td>
<td>by Fail</td>
</tr>
</tbody>
</table>
Ignoring **Restart** (for simplicity), a **common strategy** is to apply the rules using the following priorities:

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

**Note:** Depending on the cost of checking the $T$-satisfiability of $M$, Step (2) can be applied with lower frequency or priority.
Ignoring **Restart** (for simplicity), a **common strategy** is to apply the rules using the following priorities:

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

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

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

With $T$-Propagate and $T$-Explain, it can also be used to guide the engine’s search [Tin02]

**$T$-Propagate**

\[
\frac{l \in \text{Lit}(F) \quad M \models_T l \quad l, \overline{l} \not\in M}{M := M \cup \{l\}}
\]

**$T$-Explain**

\[
\frac{C = l \lor D \quad \overline{l}_1, \ldots, \overline{l}_n \models_T \overline{l} \quad \overline{l}_1, \ldots, \overline{l}_n <_M \overline{l}}{C := l_1 \lor \ldots \lor l_n \lor D}
\]
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]

\[
\text{T-Propagate} \quad \frac{l \in \text{Lit}(F) \quad M \models_T l \quad l, \bar{l} \notin M}{M := M \ l}
\]

\[
\text{T-Explain} \quad \frac{C = l \lor D \quad \bar{l}_1, \ldots, \bar{l}_n \models_T \bar{l} \quad \bar{l}_1, \ldots, \bar{l}_n <_M \bar{l}}{C := l_1 \lor \cdots \lor l_n \lor D}
\]
Theory Propagation Example

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

\[ \begin{array}{cccc}
M & F & C & \text{rule} \\
1 & 1, 2 \lor 3, 4 & \text{no} & \text{by Propagate}^+ \\
1 4 & 1, 2 \lor 3, 4 & \text{no} & \text{by } T\text{-Propagate } (1 \models_T 2) \\
1 4 2 & 1, 2 \lor 3, 4 & \text{no} & \text{by } T\text{-Propagate } (1, 4 \models_T 3) \\
1 4 2 3 & 1, 2 \lor 3, 4 & \text{no} & \text{by Conflict} \\
1 4 2 3 & 1, 2 \lor 3, 4 & 2 \lor 3 & \text{by Fail} \\
\text{fail} & & & \\
\end{array} \]

\textbf{Note:} \, T\text{-propagation eliminates search altogether in this case} \\
\text{no applications of Decide are needed}
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 4</td>
<td>1, 2, 3, 4</td>
<td>no</td>
<td>by Propagate$^+$</td>
</tr>
<tr>
<td>1 4 2</td>
<td>1, 2, 3, 4</td>
<td>no</td>
<td>by $T$-Propagate $\left(1 \models_T 2\right)$</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2, 3, 4</td>
<td>no</td>
<td>by $T$-Propagate $\left(1, 4 \models_T 3\right)$</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2, 3, 4</td>
<td>2, 3</td>
<td>by Conflict</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td>2, 3</td>
<td>by Fail</td>
</tr>
</tbody>
</table>

**Note:** $T$-propagation eliminates search altogether in this case. No applications of *Decide* are needed.
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 4</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by <strong>Propagate</strong></td>
</tr>
<tr>
<td>1 4 2</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by <strong>T-Propagate</strong> (1</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by <strong>T-Propagate</strong> (1, 4</td>
</tr>
<tr>
<td>fail</td>
<td>1, 2 ∨ 3, 4</td>
<td>2 ∨ 3</td>
<td>by <strong>Conflict</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>by <strong>Fail</strong></td>
</tr>
</tbody>
</table>

**Note:** \( T \)-propagation eliminates search altogether in this case
no applications of **Decide** are needed
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>[1, 2 \lor 3, \bar{4}]</th>
<th>C</th>
<th>\text{rule}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4</td>
<td>no</td>
<td>by \textbf{Propagate}^+</td>
<td></td>
</tr>
<tr>
<td>1 4 2</td>
<td>no</td>
<td>by \textbf{T-Propagate} ((1 \models_T 2))</td>
<td></td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>no</td>
<td>by \textbf{T-Propagate} ((1, 4 \models_T 3))</td>
<td></td>
</tr>
<tr>
<td>fail</td>
<td>no</td>
<td>by \textbf{Conflict}</td>
<td></td>
</tr>
</tbody>
</table>

\textbf{Note:} \(T\)-propagation eliminates search altogether in this case.
no applications of \textbf{Decide} are needed
Theory Propagation Example

\[
\begin{align*}
g(a) = c \quad & \wedge \quad f(g(a)) \neq f(c) \quad \vee \quad g(a) = d \quad \wedge \quad c \neq d \\
1 \quad & \quad 2 \quad & \quad 3 \quad & \quad 4
\end{align*}
\]

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

Note: \text{T}-propagation eliminates search altogether in this case
no applications of \text{Decide} are needed
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 4</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by Propagate+</td>
</tr>
<tr>
<td>1 4 2</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by T-Propagate (1 ⊢T 2)</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 ∨ 3, 4</td>
<td>no</td>
<td>by T-Propagate (1, 4 ⊢T 3)</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 ∨ 3, 4</td>
<td>2 ∨ 3</td>
<td>by Conflict</td>
</tr>
</tbody>
</table>

Note: \( T \)-propagation eliminates search altogether in this case
no applications of Decide are needed
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>4</td>
<td>1, (\bar{2}\lor 3, \bar{4})</td>
<td>no by Propagate+</td>
</tr>
<tr>
<td>1 2</td>
<td>4</td>
<td>1, (\bar{2}\lor 3, \bar{4})</td>
<td>no by \text{T-Propagate} ((1 \models_\text{T} 2))</td>
</tr>
<tr>
<td>1</td>
<td>4 2</td>
<td>1, (\bar{2}\lor 3, \bar{4})</td>
<td>no by \text{T-Propagate} ((1, 4 \models_\text{T} 3))</td>
</tr>
<tr>
<td>1</td>
<td>4 2 3</td>
<td>1, (\bar{2}\lor 3, \bar{4})</td>
<td>(\bar{2}\lor 3) by Conflict</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td>by Fail</td>
</tr>
</tbody>
</table>

**Note:** \text{T}-propagation eliminates search altogether in this case
no applications of Decide are needed
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 $\overset{\text{def}}{=} (1) + (2) + (3)$
At the core, current lazy SMT solvers are implementations of the transition system with rules

1. Propagate, Decide, Conflict, Explain, Backjump, Fail

2. $T$-Conflict, $T$-Propagate, $T$-Explain

3. Learn, Forget, Restart

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

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

Updated terminology:

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

*Execution:* sequence of transitions allowed by the rules and starting with $M = \varnothing$ 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 $M \models_p F_0$. 
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 \( M = \emptyset \) 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.

**Proposition** (Soundness) For every exhausted execution starting with \( F = F_0 \) and ending with fail, the clause set \( F_0 \) is \( T \)-unsatisfiable.
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 $M = \emptyset$ 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 $T$-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 $M \models_p F_0$. 
DPLL($T$) Architecture

The approach formalized so far can be implemented with a simple architecture named DPLL($T$) \cite{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 $\text{DPLL}(T)$ [GHN$^+$04, NOT06]

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

$\text{DPLL}(X)$:

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

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

$T$-solver:

- Checks the $T$-satisfiability of conjunctions of literals
- Computes theory propagations
- Produces explanations of $T$-unsatisfiability/propagation
- Must be incremental and backtrackable
Reasoning by Cases in Theory Solvers

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

**Example:** $T = \text{the theory of arrays}.$

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

$i = j$) 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.

**Conclusion:** $M$ is $T$-unsatisfiable
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.

**Conclusion:** $M$ is $T$-unsatisfiable
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.

**Conclusion:** $M$ is $T$-unsatisfiable
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.

**Conclusion:** $M$ is $T$-unsatisfiable
Reasoning by Cases in Theory Solvers

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

**Example:** $T$ = the theory of arrays.

$$M = \{ \underbrace{r(w(a, i, x), j) \neq x}_1, \underbrace{r(w(a, i, x), j) \neq r(a, j)}_2 \}$$

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

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

**Conclusion:** $M$ is $T$-unsatisfiable
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 [BNOT06]

**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
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 [BNOT06]

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
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 [BNOT06]

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
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 [*BNOT06*].

**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.
**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:

\[ s = s' \land i = j \rightarrow s = t, \quad s = s' \land i \neq j \rightarrow s = r(a, j) \]
**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:

\[
s = s' \land i = j \rightarrow s = t, \quad s = s' \land i \neq j \rightarrow s = r(a, j)
\]"
**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*:

\[ s = s' \land i = j \rightarrow s = t, \quad s = s' \land i \neq j \rightarrow s = r(a, j) \]"
To model the generation of theory lemmas for case splits, add the rule

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

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

where \( L_S \) is a finite set of literals dependent on the initial set of clauses (see [BNOT06] for a formal definition of \( L_S \))

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

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

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

where \( L_S \) is a finite set of literals dependent on the initial set of clauses (see [BNOT06] for a formal definition of \( L_S \)).

**Note:** 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 $M \models_p F$, it must either

- determine whether $M \models_T \bot$
- 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 $M$

**Note:** In practice, to determine if $M \models_T \bot$, the $T$-solver only needs a small subset of $L_S$ to be defined in $M$
Now we can relax the requirement on the theory solver:

\textit{When } \mathcal{M} \models_{p} \mathcal{F}, \text{ it must either}

- \textit{determine whether } \mathcal{M} \models_{T} \perp \text{ or}
- \textit{generate a new clause by } \text{T-Learn}\text{ containing at least one literal of } L_{S} \text{ undefined in } \mathcal{M}

The \text{T}-solver is \textit{required} to determine whether \( \mathcal{M} \models_{T} \perp \) \textit{only} if all literals in \( L_{S} \) are defined in \( \mathcal{M} \)

\textit{Note: In practice, to determine if } \mathcal{M} \models_{T} \perp, \text{ the } T\text{-solver only needs a small subset of } L_{S} \text{ to be defined in } \mathcal{M} \)
Now we can relax the requirement on the theory solver:

When $M \models \mathcal{F}$, 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 $M$

**Note:** 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

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

\[
\begin{array}{ccc}
\text{M} & \text{F} & \text{rule} \\
\hline
x = y \cup z & F & \text{by Propagate}\,^+ \\
x = y \cup z \cdot y = \emptyset \land x \neq z & F & \text{by Decide} \\
x = y \cup z \cdot y = \emptyset \land x \neq z & F & \text{by Propagate} \\
x = y \cup z \cdot y = \emptyset \land e \in x & F, (x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z) & \text{by Propagate} \\
x = y \cup z \cdot y = \emptyset \land e \in x \land e \notin z & F, (x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z) & \text{by Propagate} \\
\end{array}
\]

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

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

This enables an application of \( T \)-Conflict with clause

\[ x \neq y \cup z \lor y \neq \emptyset \lor x = z \lor e \notin x \lor e \in z \]
Example — Theory of Finite Sets

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

<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 e \in x )</td>
<td>( F ) ((x = z \lor e \in x \lor e \in z)), ((x = z \lor e \notin x \lor e \notin z))</td>
<td>by Propagate ( T )-Learn</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land e \in x \land e \notin z )</td>
<td>( F ) ((x = z \lor e \in x \lor e \in z)), ((x = z \lor e \notin x \lor e \notin z))</td>
<td>by Decide</td>
</tr>
</tbody>
</table>

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

\[ e \in x \implies e \in y \cup z \implies e \in y \implies e \in \emptyset \implies \bot \]

This enables an application of \( T \)-Conflict with clause

\[ x \neq y \cup z \lor y \neq \emptyset \lor x = z \lor e \notin x \lor e \in z \]
Example — Theory of Finite Sets

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

<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><strong>by Propagate</strong>+</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset )</td>
<td>( F )</td>
<td><strong>by Decide</strong></td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land x \neq z )</td>
<td>( F )</td>
<td><strong>by Propagate</strong></td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land e \in x )</td>
<td>( F )</td>
<td><strong>by Decide</strong></td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land e \in x \land e \notin z )</td>
<td>( F )</td>
<td><strong>by Propagate</strong></td>
</tr>
</tbody>
</table>

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

\[ e \in x \quad \rightarrow \quad e \in y \cup z \quad \rightarrow \quad e \in y \quad \rightarrow \quad e \in \emptyset \quad \rightarrow \quad \bot \]

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

\[ x \neq y \cup z \lor y \neq \emptyset \lor x = z \lor e \notin x \lor e \in z \]
Example — Theory of Finite Sets

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

<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>( \text{by Propagate}^+ )</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset )</td>
<td>( F )</td>
<td>( \text{by Decide} )</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land x \neq z )</td>
<td>( F, (x = z \lor e \in x \lor e \notin z) )</td>
<td>( \text{by Propagate} )</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land z \notin e \in x )</td>
<td>( F, (x = z \lor e \in x \lor e \notin z) )</td>
<td>( \text{by Decide} )</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land z \notin e \in x )</td>
<td>( F, (x = z \lor e \in x \lor e \notin z) )</td>
<td>( \text{by Propagate} )</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

\[ x \neq y \cup z \lor y \neq \emptyset \lor x = z \lor e \notin x \lor e \in z \]
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><strong>by Propagate</strong></td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset )</td>
<td>( F )</td>
<td><strong>by Decide</strong></td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land x \neq z )</td>
<td>( F, (x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z) )</td>
<td><strong>by Propagate</strong></td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land e \in x )</td>
<td>( F, (x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z) )</td>
<td><strong>by Propagate</strong></td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land e \in x \land e \notin z )</td>
<td>( F, (x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z) )</td>
<td><strong>by Propagate</strong></td>
</tr>
</tbody>
</table>

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

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

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

\[ x \neq y \cup z \lor y \neq \emptyset \lor x = z \lor e \notin x \lor e \in z \]
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 ), ((x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z))</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 \lor e \in z), (x = z \lor e \notin x \lor e \notin z))</td>
<td>by Decide</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land x \neq z \land e \notin x )</td>
<td>( F ), ((x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z))</td>
<td>by Propagate</td>
</tr>
</tbody>
</table>

'T-solver can make the following deductions at this point:

\[ e \in x \implies e \in y \cup z \implies e \in y \implies e \in \emptyset \implies \bot \]

This enables an application of T-Conflict with clause

\[ x \neq y \cup z \lor y \neq \emptyset \lor x = z \lor e \notin x \lor e \in z \]
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 \lor e \in z) ), ( (x = z \lor e \notin x \lor 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 \lor e \in z) ), ( (x = z \lor e \notin x \lor e \notin z) )</td>
<td>by Decide</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 \lor e \in z) ), ( (x = z \lor e \notin x \lor e \notin z) )</td>
<td>by Propagate</td>
</tr>
</tbody>
</table>

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

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

This enables an application of \( T \)-Conflict with clause

\[ x \neq y \cup z \lor y \neq \emptyset \lor x = z \lor e \notin x \lor e \in z \]
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, (x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z))</td>
<td>by Propagate by T-Learn</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 \lor e \in z), (x = z \lor e \notin x \lor e \notin z))</td>
<td>by Decide</td>
</tr>
<tr>
<td>(x = y \cup z \land y = \emptyset \land x \neq z \land e \notin x \land e \notin z)</td>
<td>(F, (x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z))</td>
<td>by Propagate</td>
</tr>
</tbody>
</table>

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

\[ e \in x \implies e \in y \cup z \implies e \in y \implies e \in \emptyset \implies \bot \]

This enables an application of T-Conflict with clause

\[ x \neq y \cup z \lor y \neq \emptyset \lor x = z \lor e \notin x \lor e \in z \]
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 \lor x \neq z)</td>
<td>(F)</td>
<td>by Propagate</td>
</tr>
<tr>
<td>(x = x \cup z \land y = \emptyset \lor x \neq z \land e \in e)</td>
<td>(F, (x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z))</td>
<td>by (T)-Learn</td>
</tr>
<tr>
<td>(x = x \cup z \land y = \emptyset \lor x \neq z \land e \in e)</td>
<td>(F, (x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z))</td>
<td>by Decide</td>
</tr>
<tr>
<td>(x = x \cup z \land y = \emptyset \lor x \neq z \land e \in e)</td>
<td>(F, (x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z))</td>
<td>by Propagate</td>
</tr>
</tbody>
</table>

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

\[ e \in x \implies e \in y \cup z \implies e \in y \implies e \in \emptyset \implies \bot \]

This enables an application of \(T\)-Conflict with clause

\[ x \neq y \cup z \lor y \neq \emptyset \lor x = z \lor e \notin x \lor e \in z \]
Example — Theory of Finite Sets

\[ F : \ x = y \cup z \ \land \ y \not\subseteq \emptyset \lor x \not\subseteq z \]

<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 \not\subseteq z)</td>
<td>(F)</td>
<td>by Propagate</td>
</tr>
<tr>
<td>(x = y \cup z \land y = \emptyset \land x \not\subseteq z)</td>
<td>(F) (\land e \in x \land e \not\in z)</td>
<td>by T-Learn</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

\[ x \neq y \cup z \lor y \neq \emptyset \lor x = z \lor e \notin x \lor e \in z \]
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, (x = z \lor e \in x \land e \in z), )</td>
<td>by Propagate</td>
</tr>
<tr>
<td></td>
<td>( (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 e \in x )</td>
<td>( F, (x = z \lor e \in x \land e \in z), )</td>
<td>by Decide</td>
</tr>
<tr>
<td></td>
<td>( (x = z \lor e \notin x \land e \notin z) )</td>
<td>by Propagate</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land e \in x \land e \notin z )</td>
<td>( F, (x = z \lor e \in x \land e \in z), )</td>
<td>by Propagate</td>
</tr>
<tr>
<td></td>
<td>( (x = z \lor e \notin x \land e \notin z) )</td>
<td>by $T$-Learn</td>
</tr>
</tbody>
</table>

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

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

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

\[
x \neq y \cup z \lor y \neq \emptyset \lor x = z \lor e \notin x \lor e \in z
\]
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
Conclusion
SMT Applications

- Program Analysis and Verification!
- Security (e.g., checking access policies at AWS)
- Hardware configuration and verification
- Verification of smart contracts for blockchains
- Checking database integrity constraints
- Network configuration checking
- Program synthesis
- And many more...
Some Research Directions

• New theories and theory solvers, often adapted to an application domain
• Independently checkable proofs for unsatisfiable formulas
• Better algorithms for reasoning about quantifiers
• Extensions to program synthesis
• Better decision heuristics
• Better algorithms and performance
• Theory combination mechanisms
• Parallel and distributed solving
• Using SMT to check AI-generated code
Try cvc5!

About cvc5

cvc5 is an efficient open-source automatic theorem prover for Satisfiability Modulo Theories (SMT) problems. It can be used to prove the satisfiability (or, dually, the validity) of first-order formulas with respect to (combinations of) a variety of useful background theories. It further provides a Syntax-Guided Synthesis (SyGuS) engine to synthesize functions with respect to background theories and their combinations.

cvc5 is the successor of CVC4 and is intended to be an open and extensible SMT engine. It can be used as a stand-alone tool or as a library, with essentially no limit on its use for research or commercial purposes (see license). To contribute to cvc5, please refer to our contribution guidelines.

cvc5 is a joint project led by Stanford University and the University of Iowa.

Technical Support

For bug reports, please use the cvc5 issue tracker.

If you have a question, a feature request, or if you would like to contribute in some way, please use the discussions feature on the cvc5 GitHub repository.

Guidelines For Fuzzing cvc5

The development team of cvc5 is committed to ensuring that its core usage model (without experimental options) is extremely robust. At the same time, our team is small and we have to set priorities, including prioritizing user bugs over fuzzer bugs.

When applying fuzzing techniques to cvc5, we ask you to follow these guidelines.

https://cvc5.github.io/
Suggested Readings

All available from my website: http://theory.stanford.edu/~barrett/pubs


In *5th International Conference on Theory and Applications of Satisfiability Testing, SAT’02*, pages 244–251, 2002

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory.

References


   *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, October 1979


[NO05] Robert Nieuwenhuis and Albert Oliveras. DPLL(T) with Exhaustive Theory Propagation and its Application to Difference Logic.


[RRZ05] Silvio Ranise, Christophe Ringeissen, and Calogero G. Zarba. Combining data structures with
nonstably infinite theories using many-sorted logic.
In B. Gramlich, editor, Proceedings of the Workshop on Frontiers of Combining Systems, volume
3717 of Lecture Notes in Computer Science, pages 48–64. Springer, 2005

of Arrays.
In 16th Annual IEEE Symposium on Logic in Computer Science, LICS’01, pages 29–37. IEEE
Computer Society, 2001

[Sha02] Natarajan Shankar. Little engines of proof.
In Lars-Henrik Eriksson and Peter A. Lindsay, editors, FME 2002: Formal Methods - Getting IT
Right, Proceedings of the International Symposium of Formal Methods Europe (Copenhagen,

[SLB03] Sanjit A. Seshia, Shuvendu K. Lahiri, and Randal E. Bryant. A hybrid SAT-based decision procedure
for separation logic with uninterpreted functions.

[SSB02] O. Strichman, S. A. Seshia, and R. E. Bryant. Deciding Separation Formulas with SAT.
In E. Brinksma and K. G. Larsen, editors, 14th International Conference on Computer Aided
Verification, CAV’02, volume 2404 of Lecture Notes in Computer Science, pages 209–222. Springer,
2002
[TdH08] N. Tillmann and J. de Halleux. Pex-White Box Test Generation for .NET.


Combining Theories
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 ($T_{UF}$)

**Question:** Given solvers for each theory, can we combine them modularly into one for $T_{LA} \cup T_A \cup T_{UF}$?

Under certain conditions, we can do it with the Nelson-Oppen combination method [NO79, Opp80].
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 \( (T_{UF}) \)

Question: Given solvers for each theory, can we combine them modularly into one for \( T_{LA} \cup T_A \cup T_{UF} \)?

Under certain conditions, we can do it with the Nelson-Oppen combination method [NO79, Opp80]
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 ($T_{UF}$)

**Question:** Given solvers for each theory, can we combine them modularly into one for $T_{LA} \cup T_A \cup T_{UF}$?

Under certain conditions, we can do it with the Nelson-Oppen combination method [NO79, Opp80]
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 \((T_{UF})\)

**Question:** Given solvers for each theory, can we combine them modularly into one for \(T_{LA} \cup T_A \cup T_{UF}\)?

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

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

**First step:** *purify* literals so that each belongs to a single theory
Consider the following set of literals over $T_{LRA} \cup T_{UF}$ ($T_{LRA}$, linear real arithmetic):

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

**First step:** *purify* literals so that each belongs to a single theory
Consider the following set of literals over $T_{LRA} \cup T_{UF}$ ($T_{LRA}$, linear real arithmetic):

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

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

\[
\begin{align*}
  f(f(x) - f(y)) &= a &\Rightarrow& f(e_1) &= a \\
  e_1 &= f(x) - f(y) &\Rightarrow& f(e_1) &= a \\
  e_1 &= e_2 - e_3 \\
  e_2 &= f(x) \\
  e_3 &= f(y)
\end{align*}
\]
Consider the following set of literals over $T_{LRA} \cup T_{UF}$ ($T_{LRA}$, linear real arithmetic):

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

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

\[
\begin{align*}
  f(0) &> a + 2 \quad \Longrightarrow \quad f(e_4) &> a + 2 \quad \Longrightarrow \quad f(e_4) &= e_5 \\
  e_4 &= 0 \\
  e_5 &> a + 2
\end{align*}
\]
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>$L_1$</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>

$L_1 \models_{UF} e_2 = e_3 \quad L_2 \models_{LRA} e_1 = e_4$

$L_1 \models_{UF} a = e_5$

**Third step:** check for satisfiability locally

Report unsatisfiable
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>

$L_1 \models_{UF} e_2 = e_3 \quad L_2 \models_{LRA} e_1 = e_4$

$L_1 \models_{UF} a = e_5$

**Third step:** check for satisfiability locally

Report unsatisfiable
Motivating Example (Convex Case)

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

\[
\begin{array}{c|c}
L_1 & L_2 \\
\hline
f(e_1) = a & e_2 - e_3 = e_1 \\
f(x) = e_2 & e_4 = 0 \\
f(y) = e_3 & e_5 > a + 2 \\
f(e_4) = e_5 & e_2 = e_3 \\
x = y & a = e_5 \\
e_1 = e_4 & \\
\end{array}
\]

$L_1 \models_{UF} e_2 = e_3$  
$L_2 \models_{LRA} e_1 = e_4$  
$L_1 \models_{UF} a = e_5$

**Third step:** check for satisfiability locally

Report unsatisfiable.
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>

$L_1 \models_{UF} e_2 = e_3$  $L_2 \models_{LRA} e_1 = e_4$

$L_1 \models_{UF} a = e_5$

**Third step:** check for satisfiability locally

$L_1 \models_{UF} \bot$  Report unsatisfiable
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>

$L_1 \models_{UF} e_2 = e_3 \quad L_2 \models_{LRA} e_1 = e_4$

$L_1 \models_{UF} a = e_5$

**Third step:** check for satisfiability locally

$L_1 \models_{UF} L_2 \quad \text{Report unsatisfiable}$
Motivating Example (Convex Case)

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

\[
\begin{array}{c|c}
\text{L}_1 & \text{L}_2 \\
\hline
f(e_1) = a & e_2 - e_3 = e_1 \\
f(x) = e_2 & e_4 = 0 \\
f(y) = e_3 & e_5 > a + 2 \\
f(e_4) = e_5 & e_2 = e_3 \\
x = y & a = e_5 \\
e_1 = e_4 & \\
\end{array}
\]

\(L_1 \models_{UF} e_2 = e_3 \quad L_2 \models_{LRA} e_1 = e_4 \)

\(L_1 \models_{UF} a = e_5 \)

**Third step:** check for satisfiability locally

\(L_1 \models_{UF} \perp \quad \text{Report unsatisfiable} \)

\(L_2 \models_{LRA} \perp \)
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>

$L_1 \models_{UF} e_2 = e_3 \quad L_2 \models_{LRA} e_1 = e_4$

$L_1 \models_{UF} a = e_5$

**Third step:** check for satisfiability locally

$L_1 \models_{UF} \bot$ \quad Report unsatisfiable

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

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

\[
\begin{array}{c|c|c}
L_1 & e_2 - e_3 = e_1 \\
\hline
f(e_1) = a & e_4 = 0 \\
f(x) = e_2 & e_5 > a + 2 \\
f(y) = e_3 & e_2 = e_3 \\
f(e_4) = e_5 & a = e_5 \\
x = y & \\
e_1 = e_4 & \\
\end{array}
\]

$L_1 \models_{UF} e_2 = e_3 \quad L_2 \models_{LRA} e_1 = e_4$

$L_1 \models_{UF} a = e_5$

**Third step:** check for satisfiability locally

$L_1 \models_{UF} \bot$

$L_2 \models_{LRA} \bot$

Report unsatisfiable
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>

$L_1 \models_{UF} e_2 = e_3 \quad L_2 \models_{LRA} e_1 = e_4$

$L_1 \models_{UF} a = e_5$

**Third step:** check for satisfiability locally

$L_1 \models_{UF} \perp$

$L_2 \models_{LRA} \perp$  Report unsatisfiable
Motivating Example (Non-convex Case)

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

\[
\begin{align*}
1 & \leq x \leq 2 \\
f(1) & = a \\
f(2) & = f(1) + 3 \\
a & = b + 2
\end{align*}
\]

**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 $T_{\text{LIA}} \cup T_{\text{UF}}$ ($T_{\text{LIA}}$, linear integer arithmetic):

\[
\begin{align*}
1 \leq x & \leq 2 \\
f(1) & = a \\
f(2) & = f(1) + 3 \\
a & = b + 2
\end{align*}
\]

**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 $T_{\text{LIA}} \cup T_{\text{UF}}$ ($T_{\text{LIA}}$, linear integer arithmetic):

$$1 \leq x \leq 2$$

$$f(1) = a$$

$$f(2) = f(1) + 3$$

$$a = b + 2$$

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

$$f(1) = a \implies f(e_1) = a$$

$$e_1 = 1$$
Consider the following unsatisfiable set of literals over $T_{\text{LIA}} \cup T_{\text{UF}}$ ($T_{\text{LIA}}$, linear integer arithmetic):

\[
1 \leq x \leq 2 \\
f(1) = a \\
f(2) = f(1) + 3 \\
a = b + 2
\]

**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>$x = e_1$</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>$L_1$</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>
<tr>
<td>$x = e_1$</td>
<td></td>
</tr>
</tbody>
</table>

No more entailed equalities, but $L_1 \models_{\text{LIA}} x = e_1 \lor x = e_2$
Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

\[
\begin{array}{cc}
L_1 & L_2 \\
1 \leq x & f(e_1) = a \\
x \leq 2 & f(x) = b \\
e_1 = 1 & f(e_2) = e_3 \\
a = b + 2 & f(e_1) = e_4 \\
e_2 = 2 & x = e_1 \\
e_3 = e_4 + 3 & \\
a = e_4 & \\
x = e_1 & 
\end{array}
\]

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>(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>

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>( 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>

\( L_2 \models_{UF} a = b \), which entails \( \perp \) when sent to \( L_1 \)
**Motivating Example (Non-convex Case)**

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

<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>$x = e_2$</td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
<tr>
<td>$x = e_2$</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 ≤ $x$</td>
<td></td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x ≤ 2$</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></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>

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>$L_1$</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>
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>

\( L_2 \models_{UF} e_3 = b \), which entails \( \perp \) when sent to \( L_1 \)
The Nelson-Oppen Method

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

We consider only input problems of the form

\[
L_1 \cup L_2
\]

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

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

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

Note: Because of purification, there is no loss of generality in considering only ground $(\Sigma_i \cup C)$-literals
The Nelson-Oppen Method

Bare-bones, 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
   \[ c = d \in A \text{ or } c \neq d \in A \] for all \( c, d \in C \)

2. If \( L_i \cup A \) is \( T_i \)-unsatisfiable for \( i = 1 \) or \( i = 2 \), return unsat

3. Otherwise, return sat
Bare-bones, non-deterministic, non-incremental version

[Opp80, Rin96, TH96]:

**Input:** \( L_1 \cup L_2 \) with \( L_i \) finite set of ground \((\sum_i \cup C)\)-litersals

**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

3. Otherwise, return sat
The Nelson-Oppen Method

Bare-bones, 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

3. Otherwise, return sat
Bare-bones, 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

3. Otherwise, return sat
The Nelson-Oppen Method

Bare-bones, 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)

**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.
**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.
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.
**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

$$L \models_T s_1 = t_1 \lor \cdots \lor s_n = t_n \quad \Longrightarrow \quad L \models_T s_i = t_i \text{ for some } i$$

**Note:** 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 $T$.

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

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

**Note:** 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 $T$.

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

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

**Note:** 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 $T$.

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

How can we integrate all of them \textit{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
Let $T_1, \ldots, T_n$ be theories with respective solvers $S_1, \ldots, S_n$

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

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

1. Extend DPLL($T$) to DPLL($T_1, \ldots, T_n$)

2. Lift Nelson-Oppen to the DPLL($X_1, \ldots, X_n$) level

3. Build a DPLL($T_1, \ldots, T_n$) solver
Modeling DPLL($T_1, \ldots, T_n$) Abstractly

- Let $n = 2$, for simplicity
- Let $T_i$ be of signature $\Sigma_i$ for $i = 1, 2$, with $\Sigma_1 \cap \Sigma_2 = \emptyset$
- Let $C$ be a set of free constants
- Assume wlog that each input literal has signature $(\Sigma_1 \cup C)$ or $(\Sigma_2 \cup C)$ (no mixed literals)
- Let $M|_i \overset{\text{def}}{=} \{ (\Sigma_i \cup C)\text{-literals of } M \text{ and their complement} \}$
- Let $I(M) \overset{\text{def}}{=} \{ c = d \mid c, d \text{ occur in } C, M|_1 \text{ and } M|_2 \} \cup \{ c \neq d \mid c, d \text{ occur in } C, M|_1 \text{ and } M|_2 \}$

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

\[
\text{Decide} \quad \begin{align*}
   l \in \text{Lit}(F) \cup \text{I}(M) \\
   l, \bar{l} \not\in M
\end{align*}
\]

\[
M := M \cdot l
\]

Only change: decide on interface equalities as well

\[
\text{T-Propagate} \quad \begin{align*}
   l \in \text{Lit}(F) \cup \text{I}(M) \\
   i \in \{1, 2\} \\
   M \models_{T_i} l \\
   l, \bar{l} \not\in M
\end{align*}
\]

\[
M := M \cdot l
\]

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

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

**Decide**

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

**Only change**: decide on interface equalities as well

**$T$-Propagate**

\[
\begin{align*}
    l &\in \text{Lit}(F) \cup \text{I}(M) \quad i \in \{1, 2\} \quad M \models_{T_i} l \\
    M &:= M \cdot l
\end{align*}
\]

**Only change**: propagate interface equalities as well, but reason locally in each $T_i$
Propagate, Conflict, Explain, Backjump, Fail (unchanged)

**Decide**

\[
l \in \text{Lit}(F) \cup I(M) \quad l, \bar{l} \notin M
\]

\[
M := M \cdot l
\]

Only change: decide on interface equalities as well

**\(T\)-Propagate**

\[
l \in \text{Lit}(F) \cup I(M) \quad i \in \{1, 2\} \quad M \models_{T_i} l \quad l, \bar{l} \notin M
\]

\[
M := M \cdot l
\]

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

**T-Conflict**

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

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

**T-Explain**

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

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

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

**I-Learn**

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

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

**New rule**: for entailed disjunctions of interface literals
Abstract DPLL Modulo Multiple Theories

**T-Conflict**

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

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

**T-Explain**

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

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

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

**I-Learn**

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

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

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

\[
F := \begin{align*}
\begin{array}{l}
\text{0} & f(e_1) = a \land e_2 - e_3 = e_1 \\
\text{1} & f(x) = e_2 \land e_4 = 0 \\
\text{2} & f(y) = e_3 \land e_5 > a + 2 \\
\text{3} & f(e_4) = e_5 \\
\text{4} & x = y \\
\text{5} & e_2 = e_3 \\
\text{6} & e_1 = e_4 \\
\text{7} & a = e_5 \\
\end{array}
\end{align*}
\]

<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>by Propagate +</td>
<td></td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>F</td>
<td>no</td>
<td>by T-Propagate (1, 2, 4 \models_{UF} 8)</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8</td>
<td>F</td>
<td>no</td>
<td>by T-Propagate (5, 6, 8 \models_{LRA} 9)</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 (0, 3, 9 \models_{UF} 10)</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>F</td>
<td>7 \lor 10</td>
<td>by T-Conflict (7, 10 \models_{LRA} \bot)</td>
</tr>
<tr>
<td>fail</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>by Fail</td>
<td></td>
</tr>
</tbody>
</table>
Example — Convex Theories

\[
F := \begin{align*}
0 & f(e_1) = a \\
1 & f(x) = e_2 \\
2 & f(y) = e_3 \\
3 & f(e_4) = e_5 \\
4 & x = y \\
5 & e_2 - e_3 = e_1 \\
6 & e_4 = 0 \\
7 & e_5 > a + 2 \\
8 & e_2 = e_3 \\
9 & e_1 = e_4 \\
10 & a = e_5
\end{align*}
\]

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

\[ F := \begin{align*}
0 & \quad f(e_1) = a \\
1 & \quad f(x) = e_2 \\
2 & \quad f(y) = e_3 \\
3 & \quad f(e_4) = e_5 \\
4 & \quad x = y \\
5 & \quad e_2 - e_3 = e_1 \\
6 & \quad e_4 = 0 \\
7 & \quad e_5 > a + 2 \\
8 & \quad e_2 = e_3 \\
9 & \quad e_1 = e_4 \\
10 & \quad a = e_5
\end{align*} \]

<table>
<thead>
<tr>
<th></th>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5 6 7</td>
<td>F</td>
<td>no</td>
<td>by Propagate⁺</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>no</td>
<td>by T-Propagate (1, 2, 4 \models_{UF} 8)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>F</td>
<td>no</td>
<td>by T-Propagate (5, 6, 8 \models_{LRA} 9)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>F</td>
<td>no</td>
<td>by T-Propagate (0, 3, 9 \models_{UF} 10)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>F</td>
<td>7 ∨ 10</td>
<td>\text{fail}</td>
<td>by T-Conflict (7, 10 \models_{LRA} ⊥)</td>
</tr>
<tr>
<td>10</td>
<td>F</td>
<td></td>
<td></td>
<td>by Fail</td>
</tr>
</tbody>
</table>
Example — Convex Theories

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

\[ e_2 = e_3 \\
8. \\
\]

\[ e_1 = e_4 \\
9. \\
\]

\[ a = e_5 \\
10. \]

<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>by Propagate +</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>no</td>
<td>by T-Conflict (7, 10 \models LRA ⊥)</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>yes</td>
<td>by Fail</td>
</tr>
</tbody>
</table>


### Example — Convex Theories

Given a set of formulas $F := f(e_1) = a \land f(x) = e_2 \land f(y) = e_3 \land f(e_4) = e_5 \land x = y \land e_2 - e_3 = e_1 \land e_4 = 0 \land e_5 > a + 2$,

<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>by Propagate$^+$</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8</td>
<td>F</td>
<td>no</td>
<td>by $T$-Propagate $(1, 2, 4 \models_{UF} 8)$</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 $(5, 6, 8 \models_{LRA} 9)$</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 $(0, 3, 9 \models_{UF} 10)$</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 fail</td>
<td>F</td>
<td>7 \lor 10</td>
<td>by $T$-Conflict $(7, 10 \models_{LRA} \bot)$</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>F</td>
<td>no</td>
<td>by Fail</td>
</tr>
</tbody>
</table>


Example — Convex Theories

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Example — Convex Theories

\[ F := \begin{align*}
0 & : f(e_1) = a \\
1 & : f(x) = e_2 \\
2 & : f(y) = e_3 \\
3 & : f(e_4) = e_5 \\
4 & : x = y \\
5 & : e_2 - e_3 = e_1 \\
6 & : e_4 = 0 \\
7 & : e_5 > a + 2 \\
8 & : e_2 = e_3 \\
9 & : e_1 = e_4 \\
10 & : a = e_5
\end{align*} \]

<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 no</td>
<td>by Propagate $+$</td>
<td></td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8</td>
<td>F no</td>
<td>by T-Propagate (1, 2, 4 $\models_{\text{UF}}$ 8)</td>
<td></td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>F no</td>
<td>by T-Propagate (5, 6, 8 $\models_{\text{LRA}}$ 9)</td>
<td></td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>F no</td>
<td>by T-Propagate (0, 3, 9 $\models_{\text{UF}}$ 10)</td>
<td></td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>$\overline{F}$ $\lor$ $\overline{10}$</td>
<td>by T-Conflict (7, 10 $\models_{\text{LRA}}$ $\bot$)</td>
<td></td>
</tr>
</tbody>
</table>
Example — Convex Theories

\[
F := \begin{align*}
\frac{0}{2} & f(e_1) = a \land e_2 - e_3 = e_1 \\
\frac{1}{6} & f(x) = e_2 \land e_4 = 0 \\
\frac{2}{7} & f(y) = e_3 \land e_5 > a + 2 \\
\frac{3}{4} & f(e_4) = e_5 \land x = y
\end{align*}
\]

\[
\begin{align*}
\frac{5}{8} & e_2 = e_3 \\
\frac{6}{9} & e_1 = e_4 \\
\frac{7}{10} & a = e_5
\end{align*}
\]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>no</td>
<td>by Propagate$^+$</td>
<td></td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>F</td>
<td>no</td>
<td>by $T$-Propagate $\langle 1, 2, 4 \rangle \models_{UF} 8$</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8</td>
<td>F</td>
<td>no</td>
<td>by $T$-Propagate $\langle 5, 6, 8 \rangle \models_{LRA} 9$</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 $\langle 0, 3, 9 \rangle \models_{UF} 10$</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>$\overline{7} \lor \overline{10}$</td>
<td>by $T$-Conflict $\langle 7, 10 \rangle \models_{LRA} \bot$</td>
<td>by Fail</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Example — Non-convex Theories

\[ F := \begin{align*}
    f(e_1) &= a & f(x) &= b & f(e_2) &= e_3 & f(e_1) &= e_4 \\
    1 \leq x & \leq 2 & e_1 &= 1 & a &= b + 2 \\
    4 & 5 & 6 & 7 & 8 & 9
\end{align*} \]

\[ \begin{array}{l}
    a = e_4 \\
    x = e_1 \\
    x = e_2 \\
    a = b
\end{array} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ... 9</td>
<td>F</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>0 ... 9 10</td>
<td>F</td>
<td>no</td>
<td>by T-Propagate (0, 11 \models_{UF} 10)</td>
</tr>
<tr>
<td>0 ... 9 10 11</td>
<td>F, \frac{4}{5} \vee \frac{5}{11} \vee 12</td>
<td>no</td>
<td>by I-Learn (\models_{LIA} \frac{4}{5} \vee 11 \vee 12)</td>
</tr>
<tr>
<td>0 ... 9 10 11 13</td>
<td>F, \frac{4}{5} \vee \frac{5}{11} \vee 12</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>0 ... 9 10 11 13</td>
<td>F, \frac{4}{5} \vee \frac{5}{11} \vee 12</td>
<td>no</td>
<td>by T-Propagate (0, 1, 11 \models_{UF} 13)</td>
</tr>
<tr>
<td>0 ... 9 10 11 13 11</td>
<td>F, \frac{4}{5} \vee \frac{5}{11} \vee 12</td>
<td>no</td>
<td>by T-Conflict (7, 13 \models_{UF} \bot)</td>
</tr>
<tr>
<td>0 ... 9 10 11 12</td>
<td>F, \frac{4}{5} \vee \frac{5}{11} \vee 12</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>0 ... 9 10 11 12</td>
<td>F, \frac{4}{5} \vee \frac{5}{11} \vee 12</td>
<td>no</td>
<td>by T-Propagate (0, 1, 13 \models_{UF} 11)</td>
</tr>
<tr>
<td>0 ... 9 10 11 12</td>
<td>F, \frac{4}{5} \vee \frac{5}{11} \vee 12</td>
<td>no</td>
<td>by Propagate</td>
</tr>
<tr>
<td>... fail ...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(exercise)
Example — Non-convex Theories

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

$$\begin{align*}
& a = e_4 \\
& x = e_1 \\
& x = e_2 \\
& a = b \\
\end{align*}$$

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>10</td>
<td>no</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>10</td>
<td>by Propagate$^+$</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>10</td>
<td>by T-Propagate (0, 3 $\models_{UF}$ 10)</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>10</td>
<td>by I-Learn ($\models_{LIA}$ 4 $\lor$ 5 $\lor$ 11 $\lor$ 12)</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>10</td>
<td>by Decide</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>10</td>
<td>by T-Propagate (0, 1, 11 $\models_{UF}$ 13)</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>10</td>
<td>by T-Conflict (7, 13 $\models_{UF}$ ⊥)</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>10</td>
<td>by Backjump</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>10</td>
<td>by T-Propagate (0, 1, 13 $\models_{UF}$ 11)</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>10</td>
<td>by Propagate (exercise)</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>10</td>
<td>by Fail</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>10</td>
<td>by Propagate</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>10</td>
<td>by Fail</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
<td>10</td>
<td>by Propagate (exercise)</td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>10</td>
<td>by Fail</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example — Non-convex Theories

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

\[ \begin{array}{c}
a = e_4 \\
x = e_1 \\
x = e_2 \\
a = b
\end{array} \]

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

\(\ldots\)
Example — Non-convex Theories

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

\[
\begin{array}{cccc}
M & F & C & \text{rule} \\
\hline
0 \ldots 9 & F & \text{no} & \\
0 \ldots 9 \ 10 & F & \text{no} & \text{by Propagate}^+ \\
0 \ldots 9 \ 10 \ 11 & F, 4 \lor 5 \lor 11 \lor 12 & \text{no} & \text{by } T\text{-Propagate } (0, 3 \models_{UF} 10) \\
0 \ldots 9 \ 10 \ 11 \ 13 & F, 4 \lor 5 \lor 11 \lor 12 & \text{no} & \text{by 1-Learn } (\models_{LIA} 4 \lor 5 \lor 11 \lor 12) \\
0 \ldots 9 \ 10 \ 11 \ 13 & F, 4 \lor 5 \lor 11 \lor 12 & \text{no} & \text{by Decide} \\
0 \ldots 9 \ 10 \ 11 \ 13 & F, 4 \lor 5 \lor 11 \lor 12 & \text{no} & \text{by } T\text{-Propagate } (0, 1, 11 \models_{UF} 13) \\
0 \ldots 9 \ 10 \ 11 \ 13 \ 12 & F, 4 \lor 5 \lor 11 \lor 12 & \text{no} & \text{by } T\text{-Conflict } (7, 13 \models_{UF} 1) \\
0 \ldots 9 \ 10 \ 11 \ 12 & F, 4 \lor 5 \lor 11 \lor 12 & \text{no} & \text{by Backjump} \\
0 \ldots 9 \ 10 \ 11 \ 12 & F, 4 \lor 5 \lor 11 \lor 12 & \text{no} & \text{by } T\text{-Propagate } (0, 1, 13 \models_{UF} 11) \\
0 \ldots 9 \ 10 \ 11 \ 12 & F, 4 \lor 5 \lor 11 \lor 12 & \text{no} & \text{by Propagate} \\
0 \ldots 9 \ 10 \ 11 \ 12 & F, 4 \lor 5 \lor 11 \lor 12 & \text{no} & \text{by Propagate} \\
0 \ldots 9 \ 10 \ 11 \ 12 & F, 4 \lor 5 \lor 11 \lor 12 & \text{no} & \text{by Propagate} \\
0 \ldots 9 \ 10 \ 11 \ 12 & F, 4 \lor 5 \lor 11 \lor 12 & \text{no} & \text{by Propagate} \\
\ldots & \text{fail} & \ldots & \text{by Fail} \\
\end{array}
\]
Example — Non-convex Theories

\[ F := \begin{cases} 0 & \text{f(e_1) = a} \\ 1 & \text{f(x) = b} \\ 2 & \text{f(e_2) = e_3} \\ 3 & \text{f(e_1) = e_4} \\ 1 ≤ x & \text{x ≤ 2} \\ 4 & \text{e_1 = 1} \\ 5 & \text{a = b + 2} \\ 6 & \text{e_2 = 2} \\ 7 & \text{e_3 = e_4 + 3} \end{cases} \]

\[
\begin{align*}
\text{a} &= \text{e}_4 \\
\text{x} &= \text{e}_1 \\
\text{x} &= \text{e}_2 \\
\text{a} &= \text{b}
\end{align*}
\]

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

...
**Example — Non-convex Theories**

\[ F := \begin{array}{l}
  f(e_1) = a \land f(x) = b \land f(e_2) = e_3 \land f(e_1) = e_4 \land \\
  1 \leq x \land x \leq 2 \land e_1 = 1 \land a = b + 2 \land e_2 = 2 \land e_3 = e_4 + 3
\end{array} \]

\[ a = e_4 \quad x = e_1 \quad x = e_2 \quad a = b \]

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

*(exercise)*
Example — Non-convex Theories

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

\[ a = e_4 \quad x = e_1 \quad x = e_2 \quad a = b \]

<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>by Propagate</td>
</tr>
<tr>
<td>0</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate ((0, 3) \models_{UF} 10)</td>
</tr>
<tr>
<td>0</td>
<td>( F )</td>
<td>no</td>
<td>by T-Propagate ((0, 1, 11) \models_{UF} 13)</td>
</tr>
<tr>
<td>0</td>
<td>( F ) &amp; 4 \lor 5 \lor 11 \lor 12</td>
<td>no</td>
<td>by I-Learn ((\models_{LIA} 4 \lor 5 \lor 11 \lor 12)</td>
</tr>
<tr>
<td>0</td>
<td>( F ) &amp; 4 \lor 5 \lor 11 \lor 12</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>0</td>
<td>( F ) &amp; 4 \lor 5 \lor 11 \lor 12</td>
<td>no</td>
<td>by Propagate ((0, 1, 11) \models_{UF} 13)</td>
</tr>
<tr>
<td>0</td>
<td>( F ) &amp; 4 \lor 5 \lor 11 \lor 12</td>
<td>no</td>
<td>by T-Conflict ((7, 13) \models_{UF} \bot)</td>
</tr>
<tr>
<td>0</td>
<td>( F ) &amp; 4 \lor 5 \lor 11 \lor 12</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>0</td>
<td>( F ) &amp; 4 \lor 5 \lor 11 \lor 12</td>
<td>no</td>
<td>by T-Propagate ((0, 1, 13) \models_{UF} \bot)</td>
</tr>
<tr>
<td>0</td>
<td>( F ) &amp; 4 \lor 5 \lor 11 \lor 12</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>0</td>
<td>( F ) &amp; 4 \lor 5 \lor 11 \lor 12</td>
<td>...</td>
<td>by Fail</td>
</tr>
</tbody>
</table>
Example — Non-convex Theories

\[ F := \begin{align*} 
 0 &: f(e_1) = a & 1 &: f(x) = b & 2 &: f(e_2) = e_3 & 3 &: f(e_1) = e_4 \\
1 &\leq x & 2 &\leq x & e_1 = 1 & a = b + 2 & e_2 = 2 & e_3 = e_4 + 3 \\
4 & & 5 & & 6 & & 7 & & 8 & & 9 \\
& & & & \end{align*} \]

\[ \begin{array}{c}
  a = e_4 \\
  x = e_1 \\
  x = e_2 \\
  a = b \\
 10 & 11 & 12 & 13 \\
\end{array} \]

<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>0</td>
<td>F</td>
<td>no</td>
<td>Propagate (+)</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>\text{T-Propagate} ((0, \ 3 \models_{UF} 10))</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>\text{I-Learn} ((\models_{LIA} 4 \lor 5 \lor 11 \lor 12))</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>Decide</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>\text{T-Propagate} ((0, \ 1, \ 11 \models_{UF} 13))</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>\text{T-Conflict} ((7, \ 13 \models_{UF} \bot))</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>Backjump</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>\text{T-Propagate} ((0, \ 1, \ 13 \models_{UF} 11))</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>Propagate ((exercise))</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>Fail</td>
</tr>
<tr>
<td>\ldots</td>
<td>fail \ldots</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>
Example — Non-convex Theories

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

\[ a = e_4 \quad x = e_1 \quad x = e_2 \quad a = b \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ... 9</td>
<td>F</td>
<td>no</td>
<td>by Propagate ( ^{+} )</td>
</tr>
<tr>
<td>0 ... 9 10</td>
<td>F</td>
<td>no</td>
<td>by ( T )-Propagate ( (0, 3 \models_{\text{UF}} 10) )</td>
</tr>
<tr>
<td>0 ... 9 10 11</td>
<td>F, ( 4 \lor 5 \lor 11 \lor 12 )</td>
<td>no</td>
<td>by ( T )-Propagate ( (0, 1, 11 \models_{\text{UF}} 13) )</td>
</tr>
<tr>
<td>0 ... 9 10 11 13</td>
<td>7 \lor 13</td>
<td>no</td>
<td>by ( T )-Conflict ( (7, 13 \models_{\text{UF}} \bot) )</td>
</tr>
<tr>
<td>0 ... 9 10 11 13</td>
<td>F, ( 4 \lor 5 \lor 11 \lor 12 )</td>
<td>no</td>
<td>by Propagate (exercise)</td>
</tr>
<tr>
<td>0 ... 9 10 11 12</td>
<td>F, ( 4 \lor 5 \lor 11 \lor 12 )</td>
<td>no</td>
<td>by Fail</td>
</tr>
<tr>
<td>... fail ...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example — Non-convex Theories

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 \ldots 9</td>
<td>0</td>
<td>no</td>
<td>by Propagate $^{\top}$</td>
</tr>
<tr>
<td>0 \ldots 9 10</td>
<td>0</td>
<td>no</td>
<td>by $T$-Propagate $(0, 3 \models_{\text{UF}} 10)$</td>
</tr>
<tr>
<td>0 \ldots 9 10 \bullet 11</td>
<td>0</td>
<td>no</td>
<td>by I-Learn $(\models_{\text{LIA}} 4 \lor 5 \lor 11 \lor 12)$</td>
</tr>
<tr>
<td>0 \ldots 9 10 \bullet 11 13</td>
<td>0</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>0 \ldots 9 10 \bullet 11 13</td>
<td>0</td>
<td>no</td>
<td>by $T$-Propagate $(0, 1, 11 \models_{\text{UF}} 13)$</td>
</tr>
<tr>
<td>0 \ldots 9 10 \bullet 11 13</td>
<td>0</td>
<td>no</td>
<td>by $T$-Conflict $(7, 13 \models_{\text{UF}} \bot)$</td>
</tr>
<tr>
<td>0 \ldots 9 10 \bullet 11 13</td>
<td>0</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>0 \ldots 9 10 \bullet 11 13</td>
<td>0</td>
<td>no</td>
<td>by $T$-Propagate $(0, 1, 13 \models_{\text{UF}} 11)$</td>
</tr>
</tbody>
</table>

... fail ...

Exercise
Example — Non-convex Theories

\[ F := f(e_1) = a \land f(x) = b \land e_1 = 1 \land a = b + 2 \land e_2 = 2 \land e_3 = e_4 + 3 \]

\[
\begin{align*}
0 &:= a = e_4 \\
1 &:= x = e_1 \\
2 &:= x = e_2 \\
3 &:= a = b
\end{align*}
\]

<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>by Propagate (^+)</td>
</tr>
<tr>
<td>0 ... 9</td>
<td>F</td>
<td>no</td>
<td>by (T)-Propagate ((0, 3, \models_{UF} 10))</td>
</tr>
<tr>
<td>0 ... 9 10</td>
<td>F</td>
<td>no</td>
<td>by I-Learn ((\models_{LIA} 4 \land 5 \land 11 \land 12))</td>
</tr>
<tr>
<td>0 ... 9 10 11</td>
<td>F, (4 \lor 5 \lor 11 \lor 12)</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>0 ... 9 10 11 13</td>
<td>F, (4 \lor 5 \lor 11 \lor 12)</td>
<td>no</td>
<td>by (T)-Propagate ((0, 1, 11, \models_{UF} 13))</td>
</tr>
<tr>
<td>0 ... 9 10 11 13 12</td>
<td>F, (4 \lor 5 \lor 11 \lor 12)</td>
<td>7 \lor (\overline{13})</td>
<td>by (T)-Conflict ((7, 13, \models_{UF} \bot))</td>
</tr>
<tr>
<td>0 ... 9 10 11 12</td>
<td>F, (4 \lor 5 \lor 11 \lor 12)</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>0 ... 9 10 11 12</td>
<td>F, (4 \lor 5 \lor 11 \lor 12)</td>
<td>no</td>
<td>by (T)-Propagate ((0, 1, \overline{13}, \models_{UF} \overline{11}))</td>
</tr>
<tr>
<td>... fail ...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(exercise)
Example — Non-convex Theories

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ... 9</td>
<td>( F )</td>
<td>no</td>
<td>by Propagate ( ^+ )</td>
</tr>
<tr>
<td>0 ... 9 10</td>
<td>( F )</td>
<td>no</td>
<td>by T-Propagate ( (0, 3 \models_{UF} 10) )</td>
</tr>
<tr>
<td>0 ... 9 10 ( \bullet ) 11</td>
<td>( F, 4 \lor 5 \lor 11 \lor 12 )</td>
<td>no</td>
<td>by ( I )-Learn ( (\models_{LIA} 4 \lor 5 \lor 11 \lor 12) )</td>
</tr>
<tr>
<td>0 ... 9 10 ( \bullet ) 11 13</td>
<td>( F, 4 \lor 5 \lor 11 \lor 12 )</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>0 ... 9 10 ( \bullet ) 11 13</td>
<td>( F, 4 \lor 5 \lor 11 \lor 12 )</td>
<td>no</td>
<td>by T-Propagate ( (0, 1, 11 \models_{UF} 13) )</td>
</tr>
<tr>
<td>0 ... 9 10 ( \overline{13} ) 11 13</td>
<td>( F, 4 \lor 5 \lor 11 \lor 12 )</td>
<td>( 7 \lor \overline{13} )</td>
<td>by T-Conflict ( (7, 13 \models_{UF} \bot) )</td>
</tr>
<tr>
<td>0 ... 9 10 ( \overline{13} ) 11 12</td>
<td>( F, 4 \lor 5 \lor 11 \lor 12 )</td>
<td>no</td>
<td>by Backjump</td>
</tr>
<tr>
<td>0 ... 9 10 ( \overline{13} ) ( \overline{11} ) 12</td>
<td>( F, 4 \lor 5 \lor 11 \lor 12 )</td>
<td>no</td>
<td>by Propagate ( (exercise) )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>fail</td>
<td>( \ldots )</td>
<td>by Fail</td>
</tr>
</tbody>
</table>
Example — Non-convex Theories

\[ F := (e_1) = a \land (x) = b \land (e_2) = e_3 \land a = b + 2 \land e_2 = 2 \land e_3 = e_4 + 3 \]

\[ a = e_4 \quad x = e_1 \quad x = e_2 \quad a = b \]

<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>Propagate^+</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>T-Propagate (0, 3 \models_{UF} 10)</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>I-Learn (\models_{LIA} 4 \lor 5 \lor 11 \lor 12)</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>Decide</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>T-Propagate (0, 1, 11 \models_{UF} 13)</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>7 \lor 13 by T-Conflict (7, 13 \models_{UF} \bot)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>T-Propagate (0, 1, 13 \models_{UF} \bot)</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>Propagate (exercise)</td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td>no</td>
<td>Fail</td>
</tr>
</tbody>
</table>