Satisfiability Modulo Theories

Materials by Clark Barrett, Stanford University

CS357: October 2019
**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
The Satisfiability Revolution

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*
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

- 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*

Diagram:
- Arithmetics
- Arrays
- UF
- Core
- SAT Solver (DPLL)
- Assertions
  - Explanation
  - Conflicts
  - Lemmas
  - Propagation
SMT Solvers

Theory Solvers

- Decide $T$-satisfiability of a conjunction of theory literals
- Incremental
- Backtrackable
- Conflict Generation
- Theory Propagation
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 in DNF and check if any of its disjuncts is $T$-sat)

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

**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 in DNF and check if any of its disjuncts is \(T\)-sat)

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

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 in DNF and check if any of its disjuncts is \(T\)-sat)

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

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 in DNF and check if any of its disjuncts is \(T\)-sat)

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

Solution: Exploit propositional satisfiability technology.
**Def.** A formula is *(un)satisﬁable in* a theory \( T \), or \( T-\text{(un)satisﬁable} \), if there is a (no) model of \( T \) that satisﬁes it.

**Note:** The \( T \)-satisﬁability of quantiﬁer-free formulas is decidable iff the \( T \)-satisﬁability of conjunctions/sets of literals is decidable.

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

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

**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: **Barcelogic, Boolector, CVC4, MathSAT, Yices, Z3**

This talk 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: *Barcelogic, Boolector, CVC4, MathSAT, Yices, Z3*

This talk 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: **Barcelogic, Boolector, CVC4, MathSAT, Yices, Z3**

This talk will focus on the lazy approach
(Very) Lazy Approach for SMT – Example

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

**Theory** $T$: Equality with Uninterpreted Functions

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

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

**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

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

- 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 4, 1 \lor 3 \lor 4\} to SAT solver.
- SAT solver finds \{1, 2 \lor 3, 4, 1 \lor 2 \lor 4, 1 \lor 3 \lor 4\} unsat.
- Done: the original formula is unsatisfiable in UF.
(Very) Lazy Approach for SMT – Example

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

- 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 \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2, \overline{1} \lor 3 \lor 4\} \) to SAT solver.
  SAT solver finds \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4\} \) unsat.

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

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

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

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

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

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

- 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

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

- 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.
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
Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- Check $T$-satisfiability only of full propositional model
- Check $T$-satisfiability of partial assignment $M$ as it grows
- If $M$ is $T$-unsatisfiable, add $\neg M$ as a clause
  - If $M$ is $T$-unsatisfiable, identify a $T$-unsatisfiable subset $M_0$ of $M$ and add $\neg M_0$ as a clause
  - If $M$ is $T$-unsatisfiable, 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)
An Abstract Framework for Lazy SMT

Several variants and enhancements of lazy SMT solvers exist

They can be modeled abstractly and declaratively as transition systems

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

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

An abstract framework helps one:

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

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

• Modern SAT solvers are based on the DPLL procedure \[\text{DP60, DLL62}\]

• DPLL tries to build incrementally a satisfying truth assignment \(M\) for a CNF formula \(F\)

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

• If a wrong guess for a literal leads to an inconsistency, the procedure backtracks and tries the opposite value
An Abstract Framework for DPLL

States:

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

where

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

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

- \( M_i \) is \textit{decision level} \( i \) of \( M \)
- \( M[i] \overset{\text{def}}{=} M_0 \bullet \cdots \bullet M_i \)
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]

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

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

Extending the assignment

**Propagate**

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

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

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

**Decide**

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

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

**Note:** \( \text{Lit}(F) := \{ l \mid a \text{ literal of } F \} \cup \{ \bar{l} \mid l \text{ literal of } F \} \)
Extending the assignment

**Propagate**

\[
\begin{align*}
l_1 \lor \cdots \lor l_n \lor l & \in F \quad \overline{l_1}, \ldots, \overline{l_n} \in M \quad l, \overline{l} \notin M \\
M := M \cdot l
\end{align*}
\]

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

**Decide**

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

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

Repairing the assignment

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

Backtrack

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

Note: Last premise of Backtrack enforces chronological backtracking
Transition Rules for the Original DPLL

Repairing the assignment

**Fail**

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

**Backtrack**

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

**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 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 $G$
Replace **Backtrack** with

**Conflict**

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

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

**Explain**

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

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

**Backjump**

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

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

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

**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**

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

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

**Backjump**

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

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

**Note:** \( l \prec_M l' \) 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 \( M \models_p \lnot C \) 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**

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

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

**Backjump**

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

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

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

\[
\text{Fail} \quad C \neq \text{no} \quad \bullet \notin M
\]

\text{fail}
Modify $\text{Fail}$ to

$$\begin{align*}
\text{Fail} & \quad C \neq \text{no} \quad \bullet \notin M \\
\text{fail} & \quad \text{fail}
\end{align*}$$
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></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>no</td>
<td>Propagate</td>
</tr>
<tr>
<td>1 2</td>
<td>F</td>
<td>no</td>
<td>Propagate</td>
</tr>
<tr>
<td>1 2 3</td>
<td>F</td>
<td>no</td>
<td>Decide</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>F</td>
<td>no</td>
<td>Propagate</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>F</td>
<td>no</td>
<td>Decide</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>F</td>
<td>no</td>
<td>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 \overline{7} )</td>
<td>Conflict</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>F</td>
<td>( 1 \lor \overline{2} \lor \overline{5} \lor 6 )</td>
<td>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>( 1 \lor \overline{2} \lor \overline{5} )</td>
<td>Explain with ( \overline{5} \lor 6 )</td>
</tr>
<tr>
<td>1 2 5</td>
<td>F</td>
<td>no</td>
<td>Backjump</td>
</tr>
<tr>
<td>1 2 5 3</td>
<td>F</td>
<td>no</td>
<td>Decide</td>
</tr>
</tbody>
</table>
### Execution Example

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

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

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

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

...
## Execution Example

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

<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>2 \lor \overline{5} \lor 6 \lor \overline{7}</td>
<td>by <strong>Conflict</strong></td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>1 \lor 2 \lor \overline{5} \lor 6</td>
<td>by <strong>Explain with</strong> 1 \lor \overline{5} \lor 7</td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>1 \lor 2 \lor \overline{5}</td>
<td>by <strong>Explain with</strong> 5 \lor 6</td>
<td></td>
</tr>
<tr>
<td>1 2 5</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>
</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 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{5})</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 (\bullet) 3 4 5 6 7 (\overline{6})</td>
<td>(F)</td>
<td>(1 \lor \overline{2} \lor \overline{5} \lor 6)</td>
<td>by Explain with (\overline{1} \lor \overline{5} \lor 7)</td>
</tr>
<tr>
<td>1 2 (\bullet) 3 4 5 6 7 (\overline{7})</td>
<td>(F)</td>
<td>(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 (\bullet) 3</td>
<td>(F)</td>
<td>no</td>
<td>by Decide</td>
</tr>
</tbody>
</table>

\[ \cdots \]
### Execution Example

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

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

...
Execution Example

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( F )</td>
<td>no</td>
<td>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 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 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>( 1 \lor \overline{2} \lor 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>( 1 \lor \overline{2} \lor \overline{5} )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>( F )</td>
<td>( \overline{2} \lor 5 )</td>
<td>by Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
</tbody>
</table>
### Execution Example

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

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

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

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

...
Execution Example

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

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

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

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

by Backjump

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

...
Execution Example

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

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

**Learn**

\[
\frac{F \models_p C \quad C \not\in F}{F \leftarrow F \cup \{C\}}
\]

**Forget**

\[
\frac{C = \text{no} \quad F = G \cup \{C\} \quad G \models_p C}{F \leftarrow G}
\]

**Restart**

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

**Note:** Learn can be applied to any clause stored in C when C ≠ no
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

\textbf{Propagate, Decide,}

\textbf{Conflict, Explain, Backjump,}

\textbf{Learn, Forget, Restart}

\textit{Basic DPLL} \texttt{def} =

\{ \textbf{Propagate, Decide, Conflict, Explain, Backjump} \}

\textit{DPLL} \texttt{def} = \textit{Basic DPLL} + \{ \textbf{Learn, Forget, Restart} \}
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** (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 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.

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

**Proposition** (Completeness) For every exhausted execution starting with \( F = F_0 \) and ending with \( C = \text{no} \), the clause set \( F_0 \) is satisfied by \( M \).
The DPLL System – Strategies

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

- 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 \textbf{Learn} applications \textbf{and}
  - \textbf{Restart} less and less often

ensures termination

- A \textit{common basic strategy} applies the rules with the following priorities:
  1. If \( n > 0 \) conflicts have been found so far, increase \( n \) and apply \textbf{Restart}
  2. If a clause is falsified by \( M \), apply \textbf{Conflict}
  3. Keep applying \textbf{Explain} until \textbf{Backjump} is applicable
  4. Apply \textbf{Learn}
  5. Apply \textbf{Backjump}
  6. Apply \textbf{Propagate} to completion
  7. Apply \textbf{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
\[
C = \text{no } \quad l_1, \ldots, l_n \in M \quad l_1, \ldots, l_n \models_T \bot \\
C := \overline{l}_1 \lor \cdots \lor \overline{l}_n
\]

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

$T$-Explain
\[
C = l \lor D \quad \overline{l}_1, \ldots, \overline{l}_n \models_T \overline{l} \quad \overline{l}_1, \ldots, \overline{l}_n \prec_M \overline{l} \\
C := l_1 \lor \cdots \lor l_n \lor D
\]

Note: $\bot = $ empty clause

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

Fix a theory $T$

$T$-Conflict

$C = \text{no}$ \hspace{1cm} $l_1,\ldots,l_n \in M$ \hspace{1cm} $l_1,\ldots,l_n \models_T \bot$

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

$T$-Propagate

$l \in \text{Lit}(F)$ \hspace{1cm} $M \models_T l$ \hspace{1cm} $l, \bar{l} \notin M$

$M := M \upharpoonright l$

$T$-Explain

$C = l \lor D$ \hspace{1cm} $\bar{l}_1,\ldots,\bar{l}_n \models_T \bar{l}$ \hspace{1cm} $\bar{l}_1,\ldots,\bar{l}_n \prec_M \bar{l}$

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

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

Note: $\models_T$ decided by theory solver
Fix a theory $T$

**$T$-Conflict**

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

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

**$T$-Propagate**

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

$$M := M \ l$$

**$T$-Explain**

$$C = l \lor D \quad \overline{l}_1, \ldots, \overline{l}_n \models_T \overline{l} \quad \overline{l}_1, \ldots, \overline{l}_n \prec_M \overline{l}$$

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

**Note:** $\bot = $ 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><strong>by Propagate</strong>+</td>
</tr>
<tr>
<td>1 4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td><strong>by Decide</strong></td>
</tr>
<tr>
<td>1 4 \bullet 2</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td><strong>by T-Conflict</strong></td>
</tr>
<tr>
<td>1 4 \bullet 2</td>
<td>1, 2 \lor 3, 4</td>
<td>1 \lor 2 \lor 4</td>
<td><strong>by Learn</strong></td>
</tr>
<tr>
<td>1 4 \bullet 2</td>
<td>1, 2 \lor 3, 4, 1 \lor 2 \lor 4</td>
<td>no</td>
<td><strong>by Restart</strong></td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 \lor 3, 4, 1 \lor 2 \lor 4, 1 \lor 3 \lor 4</td>
<td>no</td>
<td><strong>by Propagate</strong>+</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 \lor 3, 4, 1 \lor 2 \lor 4, 1 \lor 3 \lor 4</td>
<td>1 \lor 3 \lor 4</td>
<td><strong>by T-Conflict, Learn</strong></td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></td>
<td><strong>by Fail</strong></td>
</tr>
</tbody>
</table>
Modeling the Very Lazy Theory Approach

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

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

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

\[ \begin{array}{c|c|c|c|c} 
  & M & F & C & \text{rule} \\
\hline
1 4 \bullet 2 & 1, \overline{2} \lor 3, 4 & \text{no} & \text{by } \text{Propagate}^+ \\\n1 4 \bullet 2 & 1, 2 \lor 3, 4 & \text{no} & \text{by } \text{Decide} \\\n1 4 \bullet 2 & 1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4 & \overline{1} \lor 2 \lor 4 & \text{by } \text{T-Conflict} \\\n1 4 \bullet 2 & 1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4 & \text{no} & \text{by } \text{Learn} \\\n1 4 2 3 & 1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4 & \overline{1} \lor 3 \lor 4 & \text{by } \text{T-Conflict, Learn} \\\n1 4 2 3 & 1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4 & \overline{1} \lor 3 \lor 4 & \text{by } \text{Fail} \\\n\end{array} \]
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 4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by <strong>Propagate</strong>+</td>
</tr>
<tr>
<td>1 4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by <strong>Decide</strong></td>
</tr>
<tr>
<td>1 4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by <strong>Learn</strong></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 <strong>T-Conflict</strong></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 <strong>Restart</strong></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 <strong>Propagate</strong>+</td>
</tr>
<tr>
<td>1 4</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4</td>
<td>no</td>
<td>by <strong>T-Conflict, Learn</strong></td>
</tr>
<tr>
<td>1 4</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4</td>
<td>no</td>
<td>by <strong>Fail</strong></td>
</tr>
</tbody>
</table>
Modeling the Very Lazy Theory Approach

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>Propagate+</td>
</tr>
<tr>
<td>2</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>Decide</td>
</tr>
<tr>
<td>4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>T-Conflict</td>
</tr>
<tr>
<td>14</td>
<td>1, 2 \lor 3, 4</td>
<td>1 \lor 2 \lor 4</td>
<td>Learn</td>
</tr>
<tr>
<td>1</td>
<td>1, 2 \lor 3, 4, 1 \lor 2 \lor 4</td>
<td>1 \lor 2 \lor 4</td>
<td>Restart</td>
</tr>
<tr>
<td>4</td>
<td>1, 2 \lor 3, 4, 1 \lor 2 \lor 4</td>
<td>1 \lor 2 \lor 4</td>
<td>by Propagate+</td>
</tr>
<tr>
<td>1</td>
<td>1, 2 \lor 3, 4, 1 \lor 2 \lor 4</td>
<td>1 \lor 2 \lor 4</td>
<td>T-Conflict, Learn</td>
</tr>
<tr>
<td>fail</td>
<td>1, 2 \lor 3, 4, 1 \lor 2 \lor 4, 1 \lor 3 \lor 4</td>
<td>1 \lor 3 \lor 4</td>
<td>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</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by Decide</td>
</tr>
<tr>
<td>1 4</td>
<td>1, 2 \lor 3, 4</td>
<td>no</td>
<td>by T-Conflict</td>
</tr>
<tr>
<td>1 4</td>
<td>1, 2 \lor 3, 4, 1 \lor 2 \lor 4</td>
<td>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, 1 \lor 3 \lor 4</td>
<td>1 \lor 3 \lor 4</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

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

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

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

\begin{tabular}{c|c|c|c|c|c}
M & F & C & rule & & \\
\hline
1 & 4 & \(1, \overline{2} \lor 3, 4\) & no & by Propagate$^+$ & \\
\hline
1 & 4 & \(1, \overline{2} \lor 3, 4\) & no & by Decide & \\
1 & 4 & \(1, \overline{2} \lor 3, 4\) & no & by T-Conflict & \\
1 & 4 & \(1, \overline{2} \lor 3, 4\) & \(\overline{1} \lor 2 \lor 4\) & by Learn & \\
1 & 4 & \(1, \overline{2} \lor 3, 4\) & \(\overline{1} \lor 2 \lor 4\) & by Restart & \\
1 & 4 & \(1, \overline{2} \lor 3, 4\) & \(\overline{1} \lor 2 \lor 4\) & by Propagate$^+$ & \\
fail & 1 & \(1, \overline{2} \lor 3, 4\) & \(\overline{1} \lor 3 \lor 4\) & by T-Conflict, Learn & \\
\end{tabular}
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 (\lor) 4</td>
<td>1, (\lor) 2 (\lor) 3, 4</td>
<td>1 (\lor) 2 (\lor) 3, 4</td>
<td>no by Propagate(^+)</td>
</tr>
<tr>
<td>1 (\lor) 4</td>
<td>1, (\lor) 2 (\lor) 3, 4</td>
<td>1 (\lor) 2 (\lor) 3, 4</td>
<td>no by Decide</td>
</tr>
<tr>
<td>1 (\lor) 4</td>
<td>1, (\lor) 2 (\lor) 3, 4</td>
<td>1 (\lor) 2 (\lor) 3, 4</td>
<td>(\overline{1} \land 2 \land 4) by (T)-Conflict</td>
</tr>
<tr>
<td>1 (\lor) 4</td>
<td>1, (\lor) 2 (\lor) 3, 4</td>
<td>1 (\lor) 2 (\lor) 3, 4</td>
<td>(\overline{1} \land 2 \land 4) by Learn</td>
</tr>
<tr>
<td>1 (\lor) 4</td>
<td>1, (\lor) 2 (\lor) 3, 4</td>
<td>1 (\lor) 2 (\lor) 3, 4</td>
<td>no by Restart</td>
</tr>
<tr>
<td>1 (\lor) 4</td>
<td>1, (\lor) 2 (\lor) 3, 4</td>
<td>1 (\lor) 2 (\lor) 3, 4</td>
<td>no by Propagate(^+)</td>
</tr>
<tr>
<td>1 (\lor) 4</td>
<td>1, (\lor) 2 (\lor) 3, 4</td>
<td>1 (\lor) 2 (\lor) 3, 4</td>
<td>(\overline{1} \land 2 \land 4) by (T)-Conflict, Learn</td>
</tr>
<tr>
<td>fail</td>
<td>1, (\lor) 2 (\lor) 3, 4</td>
<td>1 (\lor) 2 (\lor) 3, 4, (\overline{1} \land \overline{3} \land 4)</td>
<td>no by Fail</td>
</tr>
</tbody>
</table>
A Better Lazy Approach

The very lazy approach can be improved considerably with

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

- 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
A Better Lazy Approach

The very lazy approach can be improved considerably with

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

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

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

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

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

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

\[\begin{array}{cccc}
\text{M} & \text{F} & \text{C} & \text{rule} \\
1 & \overline{4} & 1, 2 \lor 3, 4 & \text{no} & \text{by Propagate}^+ \\
1 & \overline{4} \bullet \overline{2} & 1, 2 \lor 3, 4 & \text{no} & \text{by Decide} \\
1 & \overline{4} \bullet \overline{2} & 1, 2 \lor 3, 4 & \overline{1} \lor 2 & \text{by T-Conflict} \\
1 & \overline{4} & 1, 2 \lor 3, 4 & \text{no} & \text{by Backjump} \\
1 & \overline{4} & \overline{2} & 1, 2 \lor 3, 4 & \text{no} & \text{by Propagate} \\
1 & \overline{4} & \overline{2} & 1, 2 \lor 3, 4 & \overline{1} \lor 3 \lor 4 & \text{by T-Conflict} \\
\end{array}\]
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>( \overline{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>( \overline{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>
Lazy Approach – Strategies

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

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

**Note:** Depending on the cost of checking the \( T \)-satisfiability of \( M \), Step (2) can be applied with lower frequency or priority
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 \textbf{\textit{T-Conflict}} as the \textbf{\textit{only theory rule}}, the theory solver is used just to \textbf{validate} the choices of the SAT engine.

With \textbf{\textit{T-Propagate}} and \textbf{\textit{T-Explain}}, it can also be used to \textbf{guide} the engine’s search. [Tin02]

\textbf{\textit{T-Propagate}}

\[ \begin{array}{l}
 l \in \text{Lit}(F) \quad M \models_T l \quad l, \bar{l} \notin M \\
 M := M \cup l
\end{array} \]

\textbf{\textit{T-Explain}}

\[ \begin{array}{l}
 C = l \lor D \quad \bar{l}_1, \ldots, \bar{l}_n \models_T \bar{\bar{l}} \quad \bar{l}_1, \ldots, \bar{l}_n \prec_M \bar{l} \\
 C := l_1 \lor \cdots \lor l_n \lor D
\end{array} \]
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**

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

**$T$-Explain**

\[
\frac{C = l \lor D \quad \bar{l}_1, \ldots, \bar{l}_n \models_T \bar{l} \quad \bar{l}_1, \ldots, \bar{l}_n \prec_M \bar{l}}{C := l_1 \lor \cdots \lor l_n \lor D}
\]
Theory Propagation Example

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

1

2

3

4

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

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

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

Note: \( T \)-propagation eliminates search altogether in this case
no applications of Deci\(de \) are needed
Theory Propagation Example

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

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

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

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

<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, (\overline{2} \lor 3, 4)</td>
<td>no</td>
<td>by Propagate^+</td>
</tr>
<tr>
<td>1 4 2</td>
<td>1, (\overline{2} \lor 3, 4)</td>
<td>no</td>
<td>by T-Propagate ((1 \models_T 2))</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, (\overline{2} \lor 3, 4)</td>
<td>no</td>
<td>by T-Propagate ((1, 4 \models_T 3))</td>
</tr>
<tr>
<td>1 4 2 3 fail</td>
<td>1, (\overline{2} \lor 3, 4)</td>
<td>2 \lor 3</td>
<td>by Conflict</td>
</tr>
<tr>
<td>1 4 2 3 fail</td>
<td>1, (\overline{2} \lor 3, 4)</td>
<td>2 \lor 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 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>
<tr>
<td>fail</td>
<td></td>
<td></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

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

1

2

3

4

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

Note: \( T \)-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**

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

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

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

Updated terminology:

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

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

*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 $C = no$, $F_0$ 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 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 $M \models_p F_0$. 
The approach formalized so far can be implemented with a simple architecture named \( \text{DPLL}(T) \) \cite{ghn04,not06}

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

- 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 \textbf{DPLL}(T) [GHN+04, NOT06]

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

\textit{T}-solver:

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

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

$i = j)$ 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.

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

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

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

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

\[ T\text{-Learn} \]

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

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

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

\( T\text{-Learn} \)

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

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

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
Now we can relax the requirement on the theory solver:

*When $M \models_p F$, 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 $M$
Now we can relax the requirement on the theory solver:

When $M \models_p F$, 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 $M$
Now we can relax the requirement on the theory solver:

*When* $M \models_p F$, *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

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = y \cup z • y = \emptyset</td>
<td>F</td>
<td>by Propagate⁺</td>
</tr>
<tr>
<td>x = y \cup z • y = \emptyset x \neq z</td>
<td>F</td>
<td>by Decide</td>
</tr>
<tr>
<td>x = y \cup z • y = \emptyset 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>
</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

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

<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>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 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 \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:

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

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>\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 )</td>
<td>\text{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>\text{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>\text{by Propagate}</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>\text{by Propagate}</td>
</tr>
</tbody>
</table>

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

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

This enables an application of \text{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>(x = y \cup z)</td>
<td>by Propagate$^+$</td>
</tr>
<tr>
<td>(x = y \cup z \land y = \emptyset)</td>
<td>(x = y \cup z \land y = \emptyset)</td>
<td>by Decide</td>
</tr>
<tr>
<td>(x = y \cup z \land y = \emptyset)</td>
<td>(F)</td>
<td>by Propagate</td>
</tr>
<tr>
<td>(x = y \cup z \land y = \emptyset)</td>
<td>(x \neq z)</td>
<td>by T-Learn</td>
</tr>
</tbody>
</table>

\(x = y \cup z \land y = \emptyset\) \(x \neq z\) \(e \in x\) 
\(F\) \(F\) 
\(F\) \(F\) 
\(F\) \(F\) 
\(F\) \(F\) 
\(F\) \(F\) 
\(F\) \(F\) 

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

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ F, (x = z \lor e \in x \lor e \in z), (x = z \lor e \notin x \lor e \notin z) \]

by **Propagate**

by **Decide**

by **Propagate**

by **T-Learn**

\[ T \text{-solver can make the following deductions at this point:} \]

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

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>( x = y \cup z )</th>
<th>( F )</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = y \cup z \ \land \ y = \emptyset )</td>
<td>( F )</td>
<td>by Propagate +</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) )</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 \notin x \lor e \notin z) )</td>
<td>by Decide</td>
</tr>
</tbody>
</table>

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

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

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 \land 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 \land 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 \land 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>( x = y \cup z )</th>
<th>( F )</th>
<th>( x = y \cup z \land y = \emptyset )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = y \cup z \land y = \emptyset \land x \neq z )</td>
<td>( F )</td>
<td>( (x = z \lor e \in x \lor e \in z) )</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \land x \neq z \land e \in x )</td>
<td>( F )</td>
<td>( (x = z \lor e \notin x \lor e \notin z) )</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 )</td>
<td>( (x = z \lor e \notin x \lor e \notin z) )</td>
</tr>
</tbody>
</table>

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

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

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

\[ T \text{-solver can make the following deductions at this point:} \]

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

This enables an application of \( T \text{-Conflict} \) 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) ), (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 x \neq z \land e \in x )</td>
<td>( F, (x = z \lor e \in x \land e \in z) ), (x = z \lor e \notin x \land e \notin z ))</td>
<td>by 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 \land e \in z) ), (x = z \lor e \notin x \land e \notin z ))</td>
<td>by Propagate</td>
</tr>
</tbody>
</table>

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

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

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

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

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

<table>
<thead>
<tr>
<th>( \text{M} )</th>
<th>( \text{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 \neq z )</td>
<td>( F )</td>
<td>by Decide</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \neq z )</td>
<td>( F )</td>
<td>by Propagate</td>
</tr>
<tr>
<td>( x = y \cup z \land y = \emptyset \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>
</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

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


References


[TdH08] N. Tillmann and J. de Halleux. Pex-White Box Test Generation for .NET.
In B. Beckert and R. Hähnle, editors, 2nd International Conference on Tests and Proofs, TAP’08,

procedure.
In F. Baader and K. U. Schulz, editors, Frontiers of Combining Systems: Proceedings of the 1st
International Workshop (Munich, Germany), Applied Logic, pages 103–120. Kluwer Academic
Publishers, March 1996

In G. Ianni and S. Flesca, editors, Proceedings of the 8th European Conference on Logics in Artificial
Intelligence (Cosenza, Italy), volume 2424 of Lecture Notes in Artificial Intelligence. Springer, 2002

References
