CS:4980 Topics in Computer Science II

Introduction to Automated Reasoning

Combining Theory Solvers with SAT solvers

Cesare Tinelli

Spring 2024

The University of Iowa

Credits

These slides are based on slides originally developed by **Cesare Tinelli** at the University of Iowa, and by **Clark Barrett**, **Caroline Trippel**, and **Andrew (Haoze) Wu** at Stanford University. Adapted by permission.

Theory of Uninterpreted Functions: $\mathcal{T}_{EUF}$

Recall: Given a signature Σ, the most general theory consists of the class of all Σ-interpretations

This family of theories parameterized by the signature is known as the theory of *Equality with Uninterpreted Functions (EUF)* or the *empty theory*

QF_UF (conjunctions of $\mathcal{T}_{EUF}$-literals) can be decided with a satisfiability proof system

The proof system can be implemented efficiently by a *congruence closure* procedure

Example: $f(a) \doteq a \land g(a) \neq f(a)$

Note: For simplicity, we only consider equality over one sort

Theory of Uninterpreted Functions: $\mathcal{T}_{EUF}$

Recall: Given a signature Σ, the most general theory consists of the class of all Σ-interpretations

This family of theories parameterized by the signature is known as the theory of *Equality with Uninterpreted Functions (EUF)* or the *empty theory*

QF_UF (conjunctions of $\mathcal{T}_{EUF}$-literals) can be decided with a satisfiability proof system

The proof system can be implemented efficiently by a *congruence closure* procedure

Example: $f(a) \doteq a \;\wedge\; g(a) \not\doteq f(a)$

Note: For simplicity, we only consider equality over one sort

Theory of Uninterpreted Functions: $\mathcal{T}_{EUF}$

Recall: Given a signature Σ, the most general theory consists of the class of all Σ-interpretations

This family of theories parameterized by the signature is known as the theory of *Equality with Uninterpreted Functions (EUF)* or the *empty theory*

QF_UF (conjunctions of $\mathcal{T}_{EUF}$-literals) can be decided with a satisfiability proof system

The proof system can be implemented efficiently by a *congruence closure* procedure

Example: $f(a) \doteq a \ \wedge \ g(a) \not\doteq f(a)$

Note: For simplicity, we only consider equality over one sort

Theory of Uninterpreted Functions: $\mathcal{T}_{EUF}$

Recall: Given a signature Σ, the most general theory consists of the class of all Σ-interpretations

This family of theories parameterized by the signature is known as the theory of *Equality with Uninterpreted Functions (EUF)* or the *empty theory*

QF_UF (conjunctions of $\mathcal{T}_{EUF}$-literals) can be decided with a satisfiability proof system

The proof system can be implemented efficiently by a *congruence closure* procedure

Example: $f(a) \doteq a \wedge g(a) \not\doteq f(a)$

Note: For simplicity, we only consider equality over one sort

Congruence Closure: Definitions

Consider a set S and a binary relation $R \subseteq S \times S$

R is an *equivalence relation* if it is reflexive, symmetric, and transitive

R is a *congruence relation* if

- it is an equivalence relation and
- for every n-ary function $f : S^n \to S$, if $R(a_i, b_i)$ holds for all $a_1, \ldots, a_n, y_1, \ldots, y_n \in S$, then $R(f(a_1, \ldots, a_n), f(a_1, \ldots, a_n))$ holds as well

Is equality an congruence relation? Yes!

Congruence Closure: Definitions

Consider a set S and a binary relation $R \subseteq S \times S$

R is an *equivalence relation* if it is reflexive, symmetric, and transitive

R is a *congruence relation* if

- it is an equivalence relation and
- for every n-ary function $f : S^n \to S$, if $R(a_i, b_i)$ holds for all $a_1, \ldots a_n, y_1, \ldots, y_n \in S$, then $R(f(a_1, \ldots, a_n), f(a_1, \ldots, a_n))$ holds as well

Is equality an congruence relation? Yes!

Congruence Closure: Definitions

Consider a set S and a binary relation $R \subseteq S \times S$

R is an *equivalence relation* if it is reflexive, symmetric, and transitive

R is a *congruence relation* if

- it is an equivalence relation and
- for every n-ary function $f : S^n \to S$, if $R(a_i, b_i)$ holds for all $a_1, \ldots a_n, y_1, \ldots, y_n \in S$, then $R(f(a_1, \ldots, a_n), f(a_1, \ldots, a_n))$ holds as well

Is equality an congruence relation? Yes!

Congruence Closure: Definitions

Consider a set S and a binary relation $R \subseteq S \times S$

R is an *equivalence relation* if it is reflexive, symmetric, and transitive

R is a *congruence relation* if

- it is an equivalence relation and
- for every n-ary function $f : S^n \rightarrow S$, if $R(a_i, b_i)$ holds for all $a_1, \ldots a_n, y_1, \ldots, y_n \in S$, then $R(f(a_1, \ldots, a_n), f(a_1, \ldots, a_n))$ holds as well

Is equality an congruence relation? Yes!

Congruence Closure: Definitions

Consider a set S and a binary relation $R \subseteq S \times S$

R is an *equivalence relation* if it is reflexive, symmetric, and transitive

R is a *congruence relation* if

- it is an equivalence relation and
- for every n-ary function $f : S^n \to S$, if $R(a_i, b_i)$ holds for all $a_1, \ldots a_n, y_1, \ldots, y_n \in S$, then $R(f(a_1, \ldots, a_n), f(a_1, \ldots, a_n))$ holds as well

Is equality an congruence relation? Yes!

Congruence Closure: Definitions

Consider a set S and a binary relation $R \subseteq S \times S$

The *equivalence closure* R^E of R is the smallest relation that

- contains R
- is a equivalent relation

The *congruence closure* R^C of R is the smallest relation that

- contains R
- is a congruence relation

Congruence Closure: Definitions

Consider a set S and a binary relation $R \subseteq S \times S$

The *equivalence closure* R^E of R is the smallest relation that

- contains R
- is a equivalent relation

The *congruence closure* R^C of R is the smallest relation that

- contains R
- is a congruence relation

Congruence Closure: Definitions

Consider a set S and a binary relation $R \subseteq S \times S$

The *equivalence closure* R^E of R is the smallest relation that

- contains R
- is a equivalent relation

The *congruence closure* R^C of R is the smallest relation that

- contains R
- is a congruence relation

Congruence Closure Algorithm

Given a Σ-formula α, its *subterm set* S_α consists of the subterms of α that do not contain $\doteq$

Example: $\alpha := f(f(a)) \doteq a \land f(f(f(a))) \doteq a \land g(a) \neq g(f(a))$

$$S_\alpha := \{ a, f(a), f(f(a)), f(f(f(a))), g(a), g(f(a)) \}$$

High-level idea:

1. Partition the literals into a set of equalities E and a set of inequalities D
2. Construct the congruence closure E^c of E over S_α
3. α is unsatisfiable iff there exists $t_1 \neq t_2 \in D$ and $(t_1, t_2) \in E^c$

Congruence Closure Algorithm

Given a Σ-formula α, its *subterm set* S_α consists of the subterms of α that do not contain $\doteq$

Example: $\alpha := \; f(f(a)) \doteq a \;\wedge\; f(f(f(a))) \doteq a \;\wedge\; g(a) \not\doteq g(f(a))$

$$S_\alpha := \{\, a,\; f(a),\; f(f(a)),\; f(f(f(a))),\; g(a),\; g(f(a)) \,\}$$

High-level idea:

1. Partition the literals into a set of equalities E and a set of inequalities D
2. Construct the congruence closure E^c of E over S_α
3. α is unsatisfiable iff there exists $t_1 \not\doteq t_2 \in D$ and $(t_1, t_2) \in E^c$

Congruence Closure Algorithm

Given a Σ-formula α, its *subterm set* S_α consists of the subterms of α that do not contain $\doteq$

Example: $\alpha := \ f(f(a)) \doteq a \ \wedge \ f(f(f(a))) \doteq a \ \wedge \ g(a) \not\doteq g(f(a))$

$$S_\alpha :\doteq \{\, a,\ f(a),\ f(f(a)),\ f(f(f(a))),\ g(a),\ g(f(a)) \,\}$$

High-level idea:

1. Partition the literals into a set of equalities E and a set of inequalities D
2. Construct the congruence closure E^c of E over S_α
3. α is unsatisfiable iff there exists $t_1 \not\doteq t_2 \in D$ and $(t_1, t_2) \in E^c$

Congruence Closure Algorithm

Given a Σ-formula α, its *subterm set* S_α consists of the subterms of α that do not contain $\doteq$

Example: $\alpha := \; f(f(a)) \doteq a \wedge f(f(f(a))) \doteq a \wedge g(a) \not\doteq g(f(a))$

$S_\alpha := \{ a, f(a), f(f(a)), f(f(f(a))), g(a), g(f(a)) \}$

High-level idea:

1. Partition the literals into a set of equalities E and a set of inequalities D
2. Construct the congruence closure E^C of E over S_α
3. α is unsatisfiable iff there exists $t_1 \not\doteq t_2 \in D$ and $(t_1, t_2) \in E^C$

Congruence Closure: Algorithm

$$\alpha = f(f(a)) \doteq a \land f(f(f(a))) \doteq a \land g(a) \not\doteq g(f(a))$$

$$S_\alpha = \{ a, f(a), f(f(a)), f(f(f(a))), g(a), g(f(a)) \}$$

Step 1: place each subterm of α into its own *congruence class*:

$$\{ a \}, \{ f(a) \}, \{ f(f(a)) \}, \{ f(f(f(a))) \}, \{ g(a) \}, \{ g(f(a)) \}$$

Congruence Closure: Algorithm

$$\alpha = f(f(a)) \doteq a \land f(f(f(a))) \doteq a \land g(a) \not\doteq g(f(a))$$

$$S_\alpha = \{ a, f(a), f(f(a)), f(f(f(a))), g(a), g(f(a)) \}$$

Step 1: place each subterm of α into its own *congruence class*:

$$\{ a \}, \{ f(a) \}, \{ f(f(a)) \}, \{ f(f(f(a))) \}, \{ g(a) \}, \{ g(f(a)) \}$$

Congruence Closure: Algorithm

Step 2: For each positive literal $t_1 \doteq t_2$ in α

- *merge* the congruence classes for t_1 and t_2
- *propagate* the resulting congruences

$$\alpha = f(f(a)) \doteq a \wedge f(f(f(a))) \doteq a \wedge g(a) \not\doteq g(f(a))$$

$$\{\, a \,\}, \{\, f(a) \,\}, \{\, f(f(a)) \,\}, \{\, f(f(f(a))) \,\}, \{\, g(a) \,\}, \{\, g(f(a)) \,\}$$

Congruence Closure: Algorithm

Step 2: For each positive literal $t_1 \doteq t_2$ in α

- *merge* the congruence classes for t_1 and t_2
- *propagate* the resulting congruences

$$\alpha = f(f(a)) \doteq a \wedge f(f(f(a))) \doteq a \wedge g(a) \not\doteq g(f(a))$$

$$\{\, a, f(f(a)) \,\}, \{\, f(a) \,\}, \{\, f(f(f(a))) \,\}, \{\, g(a) \,\}, \{\, g(f(a)) \,\}$$

Congruence Closure: Algorithm

Step 2: For each positive literal $t_1 \doteq t_2$ in α

- *merge* the congruence classes for t_1 and t_2
- *propagate* the resulting congruences

$$\alpha = f(f(a)) \doteq a \wedge f(f(f(a))) \doteq a \wedge g(a) \not\doteq g(f(a))$$

$$\{ a, f(f(a)) \}, \{ f(a), f(f(f(a))) \}, \{ g(a) \}, \{ g(f(a)) \}$$

Congruence Closure: Algorithm

Step 2: For each positive literal $t_1 \doteq t_2$ in α

- *merge* the congruence classes for t_1 and t_2
- *propagate* the resulting congruences

$$\alpha = f(f(a)) \doteq a \land f(f(f(a))) \doteq a \land g(a) \not\doteq g(f(a))$$

$$\{\, a, f(a), f(f(a)), f(f(f(a))) \,\}, \{\, g(a) \,\}, \{\, g(f(a)) \,\}$$

Congruence Closure: Algorithm

Step 2: For each positive literal $t_1 \doteq t_2$ in α

- *merge* the congruence classes for t_1 and t_2
- *propagate* the resulting congruences

$$\alpha = f(f(a)) \doteq a \wedge f(f(f(a))) \doteq a \wedge g(a) \not\doteq g(f(a))$$

$$\{\, a, f(a), f(f(a)), f(f(f(a))) \,\}, \{\, g(a), g(f(a)) \,\}$$

Congruence Closure: Algorithm

$$\alpha = f(f(a)) \doteq a \land f(f(f(a))) \doteq a \land g(a) \not\doteq g(f(a))$$

$$\{\, a, f(a), f(f(a)), f(f(f(a))) \,\}, \{\, g(a), g(f(a)) \,\}$$

Step 3: α is $\mathcal{T}_{EUF}$-unsatisfiable iff it contains a negative literal $t_1 \not\doteq t_2$, with t_1 and t_2 in the same congruence class

Note: This algorithm can be implemented efficiently with a *union-find* data structure (CC. Chap. 9.1-9.3)

Congruence Closure: Algorithm

$$\alpha = f(f(a)) \doteq a \land f(f(f(a))) \doteq a \land g(a) \not\doteq g(f(a))$$

$$\{\, a, f(a), f(f(a)), f(f(f(a))) \,\}, \{\, g(a), g(f(a)) \,\}$$

Step 3: α is $\mathcal{T}_{EUF}$-unsatisfiable iff it contains a negative literal $t_1 \not\doteq t_2$, with t_1 and t_2 in the same congruence class

Note: This algorithm can be implemented efficiently with a *union-find* data structure (CC. Chap. 9.1-9.3)

Congruence Closure: still an active research problem

Downey, et al. “Variations on the common subexpressions problem”, 1980.

Nieuwenhuis and Oliveras, “Proof-Producing Congruence Closure”, 2005.

Willsey, et al. “egg: Fast and extensible equality saturation”, 2021.

What if we have disjunctions?

The congruence closure checks the satisfiability of conjunctions of $\mathcal{T}_{EUF}$-literals

What about

$$g(a) \doteq c \;\wedge\; (f(g(a)) \not\doteq f(c) \;\vee\; g(a) \doteq d) \;\wedge\; c \not\doteq d$$

Theorem 1

For all theories $\mathcal{T}$, the $\mathcal{T}$-satisfiability of quantifier-free formulas is decidable iff the $\mathcal{T}$-satisfiability of conjunctions/sets of literals is decidable.

Proof.

Convert the formula to DNF and check if any of its disjuncts is $\mathcal{T}$-satisfiable. □

Recall: the DNF conversion is very inefficient!

A better solution: exploit propositional satisfiability technology

What if we have disjunctions?

The congruence closure checks the satisfiability of conjunctions of $\mathcal{T}_{EUF}$-literals

What about

$$g(a) \doteq c \;\wedge\; (f(g(a)) \not\doteq f(c) \;\vee\; g(a) \doteq d) \;\wedge\; c \not\doteq d$$

Theorem 1

For all theories $\mathcal{T}$, the $\mathcal{T}$-satisfiability of quantifier-free formulas is decidable iff the $\mathcal{T}$-satisfiability of conjunctions/sets of literals is decidable.

Proof.

Convert the formula to DNF and check if any of its disjuncts is $\mathcal{T}$-satisfiable. □

Recall: the DNF conversion is very inefficient!

A better solution: exploit propositional satisfiability technology

What if we have disjunctions?

The congruence closure checks the satisfiability of conjunctions of $\mathcal{T}_{EUF}$-literals

What about

$$g(a) \doteq c \ \land \ (f(g(a)) \not\doteq f(c) \ \lor \ g(a) \doteq d) \ \land \ c \not\doteq d$$

Theorem 1

For all theories $\mathcal{T}$, the $\mathcal{T}$-satisfiability of quantifier-free formulas is decidable iff the $\mathcal{T}$-satisfiability of conjunctions/sets of literals is decidable.

Proof.

Convert the formula to DNF and check if any of its disjuncts is $\mathcal{T}$-satisfiable. □

Recall: the DNF conversion is very inefficient!

A better solution: exploit propositional satisfiability technology

What if we have disjunctions?

The congruence closure checks the satisfiability of conjunctions of $\mathcal{T}_{EUF}$-literals

What about

$$g(a) \doteq c \;\land\; (f(g(a)) \not\doteq f(c) \;\lor\; g(a) \doteq d) \;\land\; c \not\doteq d$$

Theorem 1

For all theories $\mathcal{T}$, the $\mathcal{T}$-satisfiability of quantifier-free formulas is decidable iff the $\mathcal{T}$-satisfiability of conjunctions/sets of literals is decidable.

Proof.

Convert the formula to DNF and check if any of its disjuncts is $\mathcal{T}$-satisfiable. □

Recall: the DNF conversion is very inefficient!

A better solution: exploit propositional satisfiability technology

What if we have disjunctions?

The congruence closure checks the satisfiability of conjunctions of $\mathcal{T}_{EUF}$-literals

What about

$$g(a) \doteq c \;\wedge\; (f(g(a)) \not\doteq f(c) \;\vee\; g(a) \doteq d) \;\wedge\; c \not\doteq d$$

> **Theorem 1**
> *For all theories $\mathcal{T}$, the $\mathcal{T}$-satisfiability of quantifier-free formulas is decidable iff the $\mathcal{T}$-satisfiability of conjunctions/sets of literals is decidable.*

Proof.

Convert the formula to DNF and check if any of its disjuncts is $\mathcal{T}$-satisfiable. □

Recall: the DNF conversion is very inefficient!

A better solution: exploit propositional satisfiability technology

Lifting SAT Technology to SMT

Two main approaches:

1. *Eager*
   - translate into an equisatisfiable propositional formula
   - feed it to any SAT solver
2. *Lazy*
   - abstract the input formula to a propositional one
   - feed it to a (CDCL-based) SAT solver
   - use a theory decision procedure to refine the formula and guide the SAT solver
   - Notable systems: Bitwuzla, cvc5, MathSAT, Yices, Z3

Lifting SAT Technology to SMT

Two main approaches:

1. *Eager*
    - translate into an equisatisfiable propositional formula
    - feed it to any SAT solver
2. *Lazy*
    - abstract the input formula to a propositional one
    - feed it to a (CDCL-based) SAT solver
    - use a theory decision procedure to refine the formula and guide the SAT solver
    - Notable systems: Bitwuzla, cvc5, MathSAT, Yices, Z3

Lazy Approach for SMT

Given a quantifier-free Σ-formula φ, for each atomic formula α in φ, we associate a unique propositional variable $e(\alpha)$

The *Boolean skeleton* of a formula φ is a propositional logic formula, where each atomic formula α in φ is replaced with $e(\alpha)$

Example:

$$\varphi := x < 0 \lor (x + y < 1 \land \neg(x < 0)) \Rightarrow y < 0$$

Let $e(x < 0) = p_1$, $e(x + y < 1) = p_2$, $e(y < 0) = p_3$

What is the Boolean skeleton of φ? $p_1 \lor (p_2 \land \neg p_1) \Rightarrow p_3$

Lazy Approach for SMT

Given a quantifier-free Σ-formula φ, for each atomic formula α in φ, we associate a unique propositional variable $e(\alpha)$

The *Boolean skeleton* of a formula φ is a propositional logic formula, where each atomic formula α in φ is replaced with $e(\alpha)$

Example:

$$\varphi := \; x < 0 \;\vee\; (x + y < 1 \wedge \neg(x < 0)) \Rightarrow y < 0$$

Let $e(x < 0) = p_1, \; e(x + y < 1) = p_2, \; e(y < 0) = p_3$

What is the Boolean skeleton of φ? $p_1 \vee (p_2 \wedge \neg p_1) \Rightarrow p_3$

Lazy Approach for SMT

Given a quantifier-free Σ-formula φ, for each atomic formula α in φ, we associate a unique propositional variable $e(\alpha)$

The *Boolean skeleton* of a formula φ is a propositional logic formula, where each atomic formula α in φ is replaced with $e(\alpha)$

Example:

$$\varphi := \; x < 0 \;\vee\; (x + y < 1 \wedge \neg(x < 0)) \Rightarrow y < 0$$

Let $e(x < 0) = p_1,\; e(x + y < 1) = p_2,\; e(y < 0) = p_3$

What is the Boolean skeleton of φ? $p_1 \vee (p_2 \wedge \neg p_1) \Rightarrow p_3$

(Very) Lazy Approach for SMT – Example

$$g(a) \doteq c \;\wedge\; (f(g(a)) \not\doteq f(c) \;\vee\; g(a) \doteq d) \;\wedge\; c \not\doteq d$$

Simplest setting:

- Off-line SAT solver
- Non-incremental *theory solver* for conjunctions of equalities and disequalities
- Theory atoms (e.g., $g(a) \doteq c$) *abstracted* to propositional atoms (e.g., 1)

(Very) Lazy Approach for SMT – Example

$$g(a) \doteq c \;\wedge\; (f(g(a)) \not\doteq f(c) \;\vee\; g(a) \doteq d) \;\wedge\; c \not\doteq d$$

Simplest setting:

- Off-line SAT solver
- Non-incremental *theory solver* for conjunctions of equalities and disequalities
- Theory atoms (e.g., $g(a) \doteq c$) *abstracted* to propositional atoms (e.g., 1)

(Very) Lazy Approach for SMT – Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\overline{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\overline{4}}$$

(Very) Lazy Approach for SMT – Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\bar{4}}$$

- Send $\{1,\ \bar{2} \vee 3,\ \bar{4}\}$ to SAT solver
- SAT solver returns model $\{1,\ \bar{2},\ \bar{4}\}$
- Theory solver finds (concretization of) $\{1,\ \bar{2},\ \bar{4}\}$ unsat in $\mathcal{T}_{EUF}$ (meaning that $\bar{1} \vee 2 \vee 4$ is valid in $\mathcal{T}_{EUF}$)
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $\{1,\ 3,\ \bar{4}\}$
- Theory solver finds $\{1,\ 3,\ \bar{4}\}$ unsat
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ to SAT solver
- SAT solver finds $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ unsat

(Very) Lazy Approach for SMT – Example

$$\underbrace{g(a) \doteq c}_{1} \;\wedge\; \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \;\wedge\; \underbrace{c \not\doteq d}_{\bar{4}}$$

- Send $\{1,\ \bar{2} \vee 3,\ \bar{4}\}$ to SAT solver
- SAT solver returns model $\{1,\ \bar{2},\ \bar{4}\}$
- Theory solver finds (concretization of) $\{1,\ \bar{2},\ \bar{4}\}$ unsat in $\mathcal{T}_{EUF}$ (meaning that $\bar{1} \vee 2 \vee 4$ is valid in $\mathcal{T}_{EUF}$)
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $\{1,\ 3,\ \bar{4}\}$
- Theory solver finds $\{1,\ 3,\ \bar{4}\}$ unsat
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ to SAT solver
- SAT solver finds $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ unsat

(Very) Lazy Approach for SMT – Example

$$\underbrace{g(a) \doteq c}_{1} \;\wedge\; \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \;\wedge\; \underbrace{c \not\doteq d}_{\bar{4}}$$

- Send $\{1,\ \bar{2} \vee 3,\ \bar{4}\}$ to SAT solver
- SAT solver returns model $\{1,\ \bar{2},\ \bar{4}\}$
- Theory solver finds (concretization of) $\{1,\ \bar{2},\ \bar{4}\}$ unsat in $\mathcal{T}_{EUF}$ (meaning that $\bar{1} \vee 2 \vee 4$ is valid in $\mathcal{T}_{EUF}$)
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $\{1,\ 3,\ \bar{4}\}$
- Theory solver finds $\{1,\ 3,\ \bar{4}\}$ unsat
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ to SAT solver
- SAT solver finds $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ unsat

(Very) Lazy Approach for SMT – Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \mathrel{\dot{\neq}} f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \mathrel{\dot{\neq}} d}_{\bar{4}}$$

- Send $\{1,\ \bar{2} \vee 3,\ \bar{4}\}$ to SAT solver
- SAT solver returns model $\{1,\ \bar{2},\ \bar{4}\}$
- Theory solver finds (concretization of) $\{1,\ \bar{2},\ \bar{4}\}$ unsat in $\mathcal{T}_{EUF}$ (meaning that $\bar{1} \vee 2 \vee 4$ is valid in $\mathcal{T}_{EUF}$)
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $\{1,\ 3,\ \bar{4}\}$
- Theory solver finds $\{1,\ 3,\ \bar{4}\}$ unsat
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ to SAT solver
- SAT solver finds $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ unsat

(Very) Lazy Approach for SMT – Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \mathrel{\dot{\neq}} f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \mathrel{\dot{\neq}} d}_{\bar{4}}$$

- Send $\{1,\ \bar{2} \vee 3,\ \bar{4}\}$ to SAT solver
- SAT solver returns model $\{1,\ \bar{2},\ \bar{4}\}$
- Theory solver finds (concretization of) $\{1,\ \bar{2},\ \bar{4}\}$ unsat in $\mathcal{T}_{EUF}$ (meaning that $\bar{1} \vee 2 \vee 4$ is valid in $\mathcal{T}_{EUF}$)
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $\{1,\ 3,\ \bar{4}\}$
- Theory solver finds $\{1,\ 3,\ \bar{4}\}$ unsat
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ to SAT solver
- SAT solver finds $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ unsat

(Very) Lazy Approach for SMT – Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

- Send $\{1,\ \bar{2} \vee 3,\ \bar{4}\}$ to SAT solver
- SAT solver returns model $\{1,\ \bar{2},\ \bar{4}\}$
- Theory solver finds (concretization of) $\{1,\ \bar{2},\ \bar{4}\}$ unsat in $\mathcal{T}_{EUF}$ (meaning that $\bar{1} \vee 2 \vee 4$ is valid in $\mathcal{T}_{EUF}$)
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $\{1,\ 3,\ \bar{4}\}$
- Theory solver finds $\{1,\ 3,\ \bar{4}\}$ unsat
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ to SAT solver
- SAT solver finds $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ unsat

(Very) Lazy Approach for SMT – Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

- Send $\{1,\ \bar{2} \vee 3,\ \bar{4}\}$ to SAT solver
- SAT solver returns model $\{1,\ \bar{2},\ \bar{4}\}$
- Theory solver finds (concretization of) $\{1,\ \bar{2},\ \bar{4}\}$ unsat in $\mathcal{T}_{EUF}$ (meaning that $\bar{1} \vee 2 \vee 4$ is valid in $\mathcal{T}_{EUF}$)
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $\{1,\ 3,\ \bar{4}\}$
- Theory solver finds $\{1,\ 3,\ \bar{4}\}$ unsat
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ to SAT solver
- SAT solver finds $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ unsat

(Very) Lazy Approach for SMT – Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

- Send $\{1,\ \bar{2} \vee 3,\ \bar{4}\}$ to SAT solver
- SAT solver returns model $\{1,\ \bar{2},\ \bar{4}\}$
- Theory solver finds (concretization of) $\{1,\ \bar{2},\ \bar{4}\}$ unsat in $\mathcal{T}_{EUF}$ (meaning that $\bar{1} \vee 2 \vee 4$ is valid in $\mathcal{T}_{EUF}$)
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $\{1,\ 3,\ \bar{4}\}$
- Theory solver finds $\{1,\ 3,\ \bar{4}\}$ unsat
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ to SAT solver
- SAT solver finds $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ unsat

(Very) Lazy Approach for SMT – Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

- Send $\{1,\ \bar{2} \vee 3,\ \bar{4}\}$ to SAT solver
- SAT solver returns model $\{1,\ \bar{2},\ \bar{4}\}$
- Theory solver finds (concretization of) $\{1,\ \bar{2},\ \bar{4}\}$ unsat in $\mathcal{T}_{EUF}$ (meaning that $\bar{1} \vee 2 \vee 4$ is valid in $\mathcal{T}_{EUF}$)
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $\{1,\ 3,\ \bar{4}\}$
- Theory solver finds $\{1,\ 3,\ \bar{4}\}$ unsat
- Send $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ to SAT solver
- SAT solver finds $\{1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4\}$ unsat

Done! The original formula is unsatisfiable in $\mathcal{T}_{EUF}$

Eager Approach for SMT – Example

$$f(b) \doteq a \quad \vee \quad f(a) \not\doteq a$$

Step 1: Eliminate all function applications (Ackermann's encoding)

- introduce a constant symbol f_x to replace function application $f(x)$
- for each pair of introduced variables f_x, f_y, add the formula $x \doteq y \Rightarrow f_x \doteq f_y$

$$f(b) \Rightarrow f_b \qquad f(a) \Rightarrow f_a$$

$$(f_b \doteq a \ \vee \ f_a \not\doteq a) \ \wedge \ (a \doteq b \Rightarrow f_a \doteq f_b)$$

Now, atomic formulas are equalities between constants/variables

Eager Approach for SMT – Example

$$f(b) \doteq a \;\lor\; f(a) \not\doteq a$$

Step 1: Eliminate all function applications (Ackermann's encoding)

- introduce a constant symbol f_x to replace function application $f(x)$
- for each pair of introduced variables f_x, f_y, add the formula $x \doteq y \Rightarrow f_x \doteq f_y$

$$f(b) \Rightarrow f_b \qquad f(a) \Rightarrow f_a$$

$$(f_b \doteq a \;\lor\; f_a \not\doteq a) \;\land\; (a \doteq b \Rightarrow f_a \doteq f_b)$$

Now, atomic formulas are equalities between constants/variables

Eager Approach for SMT – Example

$$f(b) \doteq a \;\lor\; f(a) \not\doteq a$$

Step 1: Eliminate all function applications (Ackermann's encoding)

- introduce a constant symbol f_x to replace function application $f(x)$
- for each pair of introduced variables f_x, f_y, add the formula $x \doteq y \Rightarrow f_x \doteq f_y$

$$f(b) \Rightarrow f_b \qquad f(a) \Rightarrow f_a$$

$$(f_b \doteq a \;\lor\; f_a \not\doteq a) \;\land\; (a \doteq b \Rightarrow f_a \doteq f_b)$$

Now, atomic formulas are equalities between constants/variables

Eager Approach for SMT – Example

Rename f_b as c and f_a as d:

$$(f_b \doteq a \ \vee \ f_a \not\doteq a) \ \wedge \ (a \doteq b \Rightarrow f_a \doteq f_b)$$

becomes

$$(c \doteq a \ \vee \ d \not\doteq a) \ \wedge \ (a \doteq b \Rightarrow d \doteq c)$$

Step 2: Eliminate all equalities

- replace each pair of constants x, y with a unique propositional variable $p_{x,y}$
- add facts about reflexivity, symmetry, transitivity

$$(p_{c,a} \ \vee \ \neg p_{d,a}) \ \wedge \ (p_{a,b} \Rightarrow p_{d,c})$$
$$\wedge \ p_{a,a} \wedge p_{b,b} \wedge p_{c,c} \wedge p_{d,d} \wedge \ (p_{a,b} \Leftrightarrow p_{b,a}) \wedge (p_{a,c} \Leftrightarrow p_{c,a}) \wedge (p_{a,d} \Leftrightarrow p_{d,a}) \wedge \cdots$$
$$\wedge \ ((p_{a,b} \wedge p_{b,c}) \Rightarrow p_{a,c}) \wedge ((p_{a,c} \wedge p_{c,d}) \Rightarrow p_{a,d}) \wedge \cdots$$

The resulting propositional formula is equisatisfiable with the original T_{EUF}-formula

Note: Not all the transitivity cases are needed

Eager Approach for SMT – Example

Rename f_b as c and f_a as d:

$$(f_b \doteq a \;\vee\; f_a \not\doteq a) \;\wedge\; (a \doteq b \Rightarrow f_a \doteq f_b)$$

becomes

$$(c \doteq a \;\vee\; d \not\doteq a) \;\wedge\; (a \doteq b \Rightarrow d \doteq c)$$

Step 2: Eliminate all equalities

- replace each pair of constants x, y with a unique propositional variable $p_{x,y}$
- add facts about reflexivity, symmetry, transitivity

$$(p_{c,a} \;\vee\; \neg p_{d,a}) \;\wedge\; (p_{a,b} \Rightarrow p_{d,c})$$

$$\wedge\; p_{a,a} \wedge p_{b,b} \wedge p_{c,c} \wedge p_{d,d} \wedge\; (p_{a,b} \Leftrightarrow p_{b,a}) \wedge (p_{a,c} \Leftrightarrow p_{c,a}) \wedge (p_{a,d} \Leftrightarrow p_{d,a}) \wedge \cdots$$
$$\wedge\; ((p_{a,b} \wedge p_{b,c}) \Rightarrow p_{a,c}) \wedge ((p_{a,c} \wedge p_{c,d}) \Rightarrow p_{a,d}) \wedge \cdots$$

The resulting propositional formula is equisatisfiable with the original T_{EUF}-formula

Note: Not all the transitivity cases are needed

Eager Approach for SMT – Example

Rename f_b as c and f_a as d:

$$(f_b \doteq a \ \lor \ f_a \not\doteq a) \ \land \ (a \doteq b \Rightarrow f_a \doteq f_b)$$

becomes

$$(c \doteq a \ \lor \ d \not\doteq a) \ \land \ (a \doteq b \Rightarrow d \doteq c)$$

Step 2: Eliminate all equalities

- replace each pair of constants x, y with a unique propositional variable $p_{x,y}$
- add facts about reflexivity, symmetry, transitivity

$$\begin{gathered}(p_{c,a} \ \lor \ \neg p_{d,a}) \ \land \ (p_{a,b} \Rightarrow p_{d,c}) \\ \land \ p_{a,a} \land p_{b,b} \land p_{c,c} \land p_{d,d} \land \ (p_{a,b} \Leftrightarrow p_{b,a}) \land (p_{a,c} \Leftrightarrow p_{c,a}) \land (p_{a,d} \Leftrightarrow p_{d,a}) \land \cdots \\ \land \ ((p_{a,b} \land p_{b,c}) \Rightarrow p_{a,c}) \land ((p_{a,c} \land p_{c,d}) \Rightarrow p_{a,d}) \land \cdots\end{gathered}$$

The resulting propositional formula is equisatisfiable with the original T_{EUF}-formula

Note: Not all the transitivity cases are needed

Eager Approach for SMT – Example

Rename f_b as c and f_a as d:

$$(f_b \doteq a \ \lor \ f_a \not\doteq a) \ \land \ (a \doteq b \Rightarrow f_a \doteq f_b)$$

becomes

$$(c \doteq a \ \lor \ d \not\doteq a) \ \land \ (a \doteq b \Rightarrow d \doteq c)$$

Step 2: Eliminate all equalities

- replace each pair of constants x, y with a unique propositional variable $p_{x,y}$
- add facts about reflexivity, symmetry, transitivity

$$\begin{gathered}(p_{c,a} \ \lor \ \neg p_{d,a}) \ \land \ (p_{a,b} \Rightarrow p_{d,c}) \\ \land \ p_{a,a} \land p_{b,b} \land p_{c,c} \land p_{d,d} \land \ (p_{a,b} \Leftrightarrow p_{b,a}) \land (p_{a,c} \Leftrightarrow p_{c,a}) \land (p_{a,d} \Leftrightarrow p_{d,a}) \land \cdots \\ \land ((p_{a,b} \land p_{b,c}) \Rightarrow p_{a,c}) \land ((p_{a,c} \land p_{c,d}) \Rightarrow p_{a,d}) \land \cdots\end{gathered}$$

The resulting propositional formula is equisatisfiable with the original T_{EUF}-formula

Note: Not all the transitivity cases are needed

Discussion: eager vs. lazy approach

Eager

- translate into an equisatisfiable propositional formula
- feed it to any SAT solver

Lazy

- abstract the input formula to a propositional one
- feed it to a (CDCL-based) SAT solver
- use a theory decision procedure to refine the formula and guide the SAT solver

What are the pros and cons of the two approaches?

Discussion: eager vs. lazy approach

Eager

- translate into an equisatisfiable propositional formula
- feed it to any SAT solver

Lazy

- abstract the input formula to a propositional one
- feed it to a (CDCL-based) SAT solver
- use a theory decision procedure to refine the formula and guide the SAT solver

What are the pros and cons of the two approaches?

Discussion: eager vs. lazy approach

- **Eager**
  - Can always use the best SAT solver off the shelf
  - Requires care in encoding
  - Tends not to scale well
- **Lazy**
  - Theory-specific reasoning
  - Designing new theory solvers can be challenging
  - Might require extension of a SAT solver for more efficiency interplay with theory solver

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- Check $\mathcal{T}$-satisfiability only of full propositional model

  Check $\mathcal{T}$-satisfiability of partial assignment M as it grows

- If M is $\mathcal{T}$-unsatisfiable, add $\neg M$ as a clause

  If M is $\mathcal{T}$-unsatisfiable, identify a $\mathcal{T}$-unsatisfiable subset M_0 of M and add $\neg M_0$ as a clause

- If M is $\mathcal{T}$-unsatisfiable, add clause and restart

  If M is $\mathcal{T}$-unsatisfiable, backtrack to some point where the assignment was still $\mathcal{T}$-satisfiable

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- Check $\mathcal{T}$-satisfiability only of full propositional model

  Check $\mathcal{T}$-satisfiability of partial assignment M as it grows

- If M is $\mathcal{T}$-unsatisfiable, add $\neg M$ as a clause

  If M is $\mathcal{T}$-unsatisfiable, identify a $\mathcal{T}$-unsatisfiable subset M_0 of M and add $\neg M_0$ as a clause

- If M is $\mathcal{T}$-unsatisfiable, add clause and restart

  If M is $\mathcal{T}$-unsatisfiable, backtrack to some point where the assignment was still $\mathcal{T}$-satisfiable

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- Check $\mathcal{T}$-satisfiability only of full propositional model

  Check $\mathcal{T}$-satisfiability of partial assignment M as it grows

- If M is $\mathcal{T}$-unsatisfiable, add $\neg M$ as a clause

  If M is $\mathcal{T}$-unsatisfiable, identify a $\mathcal{T}$-unsatisfiable subset M_0 of M and add $\neg M_0$ as a clause

- If M is $\mathcal{T}$-unsatisfiable, add clause and restart

  If M is $\mathcal{T}$-unsatisfiable, backtrack to some point where the assignment was still $\mathcal{T}$-satisfiable

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- Check $\mathcal{T}$-satisfiability only of full propositional model

  Check $\mathcal{T}$-satisfiability of partial assignment M as it grows

- If M is $\mathcal{T}$-unsatisfiable, add $\neg M$ as a clause

  If M is $\mathcal{T}$-unsatisfiable, identify a $\mathcal{T}$-unsatisfiable subset M_0 of M and add $\neg M_0$ as a clause

- If M is $\mathcal{T}$-unsatisfiable, add clause and restart

  If M is $\mathcal{T}$-unsatisfiable, backtrack to some point where the assignment was still $\mathcal{T}$-satisfiable

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- Check $\mathcal{T}$-satisfiability only of full propositional model

  Check $\mathcal{T}$-satisfiability of partial assignment M as it grows

- If M is $\mathcal{T}$-unsatisfiable, add $\neg M$ as a clause

  If M is $\mathcal{T}$-unsatisfiable, identify a $\mathcal{T}$-unsatisfiable subset M_0 of M and add $\neg M_0$ as a clause

- If M is $\mathcal{T}$-unsatisfiable, add clause and restart

  If M is $\mathcal{T}$-unsatisfiable, backtrack to some point where the assignment was still $\mathcal{T}$-satisfiable

Lazy Approach – Enhancements

Several enhancements are possible to increase efficiency:

- Check $\mathcal{T}$-satisfiability only of full propositional model

  Check $\mathcal{T}$-satisfiability of partial assignment M as it grows

- If M is $\mathcal{T}$-unsatisfiable, add $\neg M$ as a clause

  If M is $\mathcal{T}$-unsatisfiable, identify a $\mathcal{T}$-unsatisfiable subset M_0 of M and add $\neg M_0$ as a clause

- If M is $\mathcal{T}$-unsatisfiable, add clause and restart

  If M is $\mathcal{T}$-unsatisfiable, backtrack to some point where the assignment was still $\mathcal{T}$-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
- 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 low effort

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
- 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 low effort

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
- 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 low effort

An Abstract Framework for Lazy SMT

Several variants and enhancements of lazy SMT solvers exist

They can be modeled a satisfiability proof system like those for Abstract DPLL and Abstract CDCL

Review: Abstract DPLL

States:

$$\text{UNSAT} \qquad \langle M, \Delta \rangle$$

where

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

Review: Abstract DPLL

States:

$$\textsc{UNSAT} \qquad \langle M, \Delta \rangle$$

where

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

Note: When convenient, we treat M as a set

Provided M contains no complementary literals it determines the assignment

$$v_M(p) = \begin{cases} \text{true} & \text{if } p \in M \\ \text{false} & \text{if } \neg p \in M \\ \text{undef} & \text{otherwise} \end{cases}$$

Review: Abstract DPLL

States:

$$\text{UNSAT} \qquad \langle M, \Delta \rangle$$

where

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

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

- M_i is *decision level i* of M
- $M^{[i]}$ denotes the subsequence $M_0 \bullet \cdots \bullet M_i$, from decision level 0 to decision level i

Review: Abstract DPLL

States:

$\textsc{unsat}$ $\langle M, \Delta \rangle$

Initial state:

- $\langle (), \Delta_0 \rangle$, where Δ_0 is to be checked for satisfiability

Review: Abstract DPLL

States:

$\textsc{unsat}$ $\qquad$ $\langle M, \Delta \rangle$

Initial state:

- $\langle (), \Delta_0 \rangle$, where Δ_0 is to be checked for satisfiability

Expected final states:

- $\textsc{unsat}$ if Δ_0 is unsatisfiable
- $\langle M, \Delta_n \rangle$ otherwise, where Δ_n is equisatisfiable with Δ_0 and satisfied by M

Review: Abstract CDCL

States:

$$\textsf{UNSAT} \qquad\qquad \langle M, \Delta, C \rangle$$

where

- M is a *sequence of literals* and *decision points* $\bullet$ (denoting a partial truth assignment)
- Δ is a *set of clauses* denoting a CNF formula
- C is either `no` or a *conflict clause*

Initial state:

- $\langle (), \Delta_0, \texttt{no} \rangle$, where Δ_0 is to be checked for satisfiability

Expected final states:

- $\textsf{UNSAT}$ if Δ_0 is unsatisfiable
- $\langle M, \Delta_n, \texttt{no} \rangle$ otherwise, where Δ_n is equisatisfiable with Δ_0 and satisfied by M

Review: Abstract CDCL

States:

$$\text{UNSAT} \qquad\qquad \langle M, \Delta, C \rangle$$

where

- M is a *sequence of literals* and *decision points* $\bullet$ (denoting a partial truth assignment)
- Δ is a *set of clauses* denoting a CNF formula
- C is either $\texttt{no}$ or a *conflict clause*

Initial state:

- $\langle (), \Delta_0, \texttt{no} \rangle$, where Δ_0 is to be checked for satisfiability

Expected final states:

- UNSAT if Δ_0 is unsatisfiable
- $\langle M, \Delta_n, \texttt{no} \rangle$ otherwise, where Δ_n is equisatisfiable with Δ_0 and satisfied by M

Review: Abstract CDCL

States:

$$\textsc{unsat} \qquad \langle M, \Delta, C \rangle$$

where

- M is a *sequence of literals* and *decision points* $\bullet$ (denoting a partial truth assignment)
- Δ is a *set of clauses* denoting a CNF formula
- C is either $\texttt{no}$ or a *conflict clause*

Initial state:

- $\langle (), \Delta_0, \texttt{no} \rangle$, where Δ_0 is to be checked for satisfiability

Expected final states:

- UNSAT if Δ_0 is unsatisfiable
- $\langle M, \Delta_n, \texttt{no} \rangle$ otherwise, where Δ_n is equisatisfiable with Δ_0 and satisfied by M

Review: CDCL proof rules

$$\textsc{Propagate}\ \frac{\{l_1,\ldots,l_n,l\} \in \Delta \qquad \bar{l}_1,\ldots,\bar{l}_n \in \mathsf{M} \qquad l,\bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\, l}$$

$$\textsc{Decide}\ \frac{l \in \mathtt{Lits}(\Delta) \qquad l,\bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M} \bullet l}$$

$$\textsc{Explain}\ \frac{\mathsf{C} = \{l\} \cup D \quad \{l_1,\ldots,l_n,\bar{l}\} \in \Delta \quad \bar{l}_1,\ldots,\bar{l}_n,\bar{l} \in \mathsf{M} \quad \bar{l}_1,\ldots,\bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := \{l_1,\ldots,l_n\} \cup D}$$

$$\textsc{Fail}\ \frac{\mathsf{C} \neq \mathtt{no} \qquad \bullet \notin \mathsf{M}}{\mathsf{UNSAT}}$$

$$\textsc{Backjump}\ \frac{\mathsf{C} = \{l_1,\ldots,l_n,l\} \qquad \mathtt{lev}(\bar{l}_1),\ldots,\mathtt{lev}(\bar{l}_n) \leq i < \mathtt{lev}(\bar{l})}{\mathsf{M} := \mathsf{M}^{[i]}\, l \qquad \mathsf{C} := \mathtt{no}}$$

$$\textsc{Restart}\ \frac{}{\mathsf{M} := \mathsf{M}^{[0]} \qquad C := \mathtt{no}}$$

$$\textsc{Conflict}\ \frac{\mathsf{C} = \mathtt{no} \qquad \{l_1,\ldots,l_n\} \in \Delta \qquad \bar{l}_1,\ldots,\bar{l}_n \in \mathsf{M}}{\mathsf{C} := \{l_1,\ldots,l_n\}}$$

$$\textsc{Learn}\ \frac{D \text{ is a clause} \qquad \Delta \models D \qquad D \notin \Delta}{\Delta := \Delta \cup \{D\}}$$

$$\textsc{Forget}\ \frac{\mathsf{C} = \mathtt{no} \qquad \Delta = \Delta' \cup \{C\} \qquad \Delta' \models C}{\Delta := \Delta'}$$

We are going to extend this abstract framework to lazy SMT

Review: CDCL proof rules

Propagate
$$\frac{\{l_1,\ldots,l_n,l\}\in\Delta \qquad \bar{l}_1,\ldots,\bar{l}_n\in \mathsf{M} \qquad l,\bar{l}\notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\; l}$$

Decide
$$\frac{l\in \mathtt{Lits}(\Delta) \qquad l,\bar{l}\notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\bullet l}$$

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

Fail
$$\frac{\mathsf{C}\neq \mathtt{no} \qquad \bullet\notin \mathsf{M}}{\mathsf{UNSAT}}$$

Backjump
$$\frac{\mathsf{C}=\{l_1,\ldots,l_n,l\} \qquad \mathtt{lev}(\bar{l}_1),\ldots,\mathtt{lev}(\bar{l}_n)\leq i<\mathtt{lev}(\bar{l})}{\mathsf{M} := \mathsf{M}^{[i]}\, l \qquad \mathsf{C} := \mathtt{no}}$$

Restart
$$\frac{}{\mathsf{M} := \mathsf{M}^{[0]} \qquad \mathsf{C} := \mathtt{no}}$$

Conflict
$$\frac{\mathsf{C}=\mathtt{no} \qquad \{l_1,\ldots,l_n\}\in\Delta \qquad \bar{l}_1,\ldots,\bar{l}_n\in \mathsf{M}}{\mathsf{C} := \{l_1,\ldots,l_n\}}$$

Learn
$$\frac{D \text{ is a clause} \qquad \Delta\models D \qquad D\notin\Delta}{\Delta := \Delta\cup\{D\}}$$

Forget
$$\frac{\mathsf{C}=\mathtt{no} \qquad \Delta=\Delta'\cup\{C\} \qquad \Delta'\models C}{\Delta := \Delta'}$$

We are going to extend this abstract framework to lazy SMT

From SAT to SMT

Same state components and transitions as in Abstract CDCL except that

- Δ contains quantifier-free clauses in some theory $\mathcal{T}$
- M is a sequence of theory literals (i.e., atomic formulas or their negations) and decision points
- CDCL Rules operate on the Boolean skeleton of Δ, given by a mapping from theory literals to propositional literals
- The proofs system is augmented with SMT-specific rules: $\mathcal{T}$-**Conflict**, $\mathcal{T}$-**Propagate** and $\mathcal{T}$-**Explain**
- Invariant: either $\mathsf{C} \neq \mathtt{no}$ or $\Delta \models_{\mathcal{T}} \mathsf{C}$ and $\mathsf{M} \models_{\mathrm{p}} \neg\mathsf{C}$

SMT-level Rules

At SAT level:

$$\textsc{Conflict}\ \frac{\mathsf{C} = \texttt{no} \qquad \{l_1, \ldots, l_n\} \in \Delta \qquad \overline{l}_1, \ldots, \overline{l}_n \in \mathsf{M}}{\mathsf{C} := \{l_1, \ldots, l_n\}}$$

At SMT level:

$$\mathcal{T}\text{-}\textsc{Conflict}\ \frac{\mathsf{C} = \texttt{no} \qquad \overline{l}_1 \land \cdots \land \overline{l}_n \models_{\mathcal{T}} \bot \qquad \overline{l}_1, \ldots, \overline{l}_n \in \mathsf{M}}{\mathsf{C} := \{l_1, \ldots, l_n\}}$$

If a set of literals in M are unsatisfiable in $\mathcal{T}$, make their negation a conflict clause

SMT-level Rules

At SAT level:

$$\textsc{Propagate}\ \frac{\{l_1, \ldots, l_n, l\} \in \Delta \qquad \overline{l}_1, \ldots, \overline{l}_n \in \mathsf{M} \qquad l, \overline{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\, l}$$

At SMT level:

$$\mathcal{T}\text{-}\textsc{Propagate}\ \frac{l \in \texttt{Lits}(\Delta) \qquad \mathsf{M} \models_{\mathcal{T}} l \qquad l, \overline{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\, l}$$

If M entails some literal l in $\mathcal{T}$, extend it with l

SMT-level Rules

At SAT level:

$$\textsc{Explain}\ \frac{\mathsf{C} = \{l\} \cup D \quad \{l_1, \ldots, l_n, \overline{l}\} \in \Delta \quad \overline{l}_1, \ldots, \overline{l}_n, \overline{l} \in \mathsf{M} \quad \overline{l}_1, \ldots, \overline{l}_n \prec_{\mathsf{M}} \overline{l}}{\mathsf{C} := \{l_1, \ldots, l_n\} \cup D}$$

At SMT level:

$$\mathcal{T}\textsc{-Explain}\ \frac{\mathsf{C} = \{l\} \cup D \quad \overline{l}_1 \wedge \cdots \wedge \overline{l}_n \models_{\mathcal{T}} \overline{l} \quad \overline{l}_1, \ldots, \overline{l}_n \prec_{\mathsf{M}} \overline{l}}{\mathsf{C} := \{l_1, \cdots, l_n\} \cup D}$$

If the complement $\overline{l}$ of a literal in the conflict clause is entailed in $\mathcal{T}$ by some literals $\overline{l}_1, \ldots, \overline{l}_n$ at lower decision levels, derive a new conflict clause by resolution with $\{l_1, \ldots, l_n, \overline{l}\}$

CDCL Modulo Theories proof rules

Decide
$$\frac{l \in \texttt{Lits}(\Delta) \qquad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M} \bullet l}$$

Fail
$$\frac{\mathsf{C} \neq \texttt{no} \qquad \bullet \notin \mathsf{M}}{\mathsf{UNSAT}}$$

Restart
$$\frac{}{\mathsf{M} := \mathsf{M}^{[0]} \qquad C := \texttt{no}}$$

Learn
$$\frac{D \text{ is a clause} \qquad \Delta \models D \qquad D \notin \Delta}{\Delta := \Delta \cup \{D\}}$$

Forget
$$\frac{\mathsf{C} = \texttt{no} \qquad \Delta = \Delta' \cup \{C\} \qquad \Delta' \models C}{\Delta := \Delta'}$$

$\mathcal{T}$-Propagate
$$\frac{l \in \texttt{Lits}(\Delta) \qquad \mathsf{M} \models_{\mathcal{T}} l \qquad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\ l}$$

$\mathcal{T}$-Conflict
$$\frac{\mathsf{C} = \texttt{no} \qquad \bar{l}_1 \wedge \cdots \wedge \bar{l}_n \models_{\mathcal{T}} \bot \qquad \bar{l}_1, \ldots, \bar{l}_n \in \mathsf{M}}{\mathsf{C} := \{l_1, \ldots, l_n\}}$$

$\mathcal{T}$-Explain
$$\frac{\mathsf{C} = \{l\} \cup D \qquad \bar{l}_1 \wedge \cdots \wedge \bar{l}_n \models_{\mathcal{T}} \bar{l} \qquad \bar{l}_1, \ldots, \bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := \{l_1, \cdots, l_n\} \cup D}$$

Propagate
$$\frac{\{l_1, \ldots, l_n, l\} \in \Delta \qquad \bar{l}_1, \ldots, \bar{l}_n \in \mathsf{M} \qquad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\ l}$$

Explain
$$\frac{\mathsf{C} = \{l\} \cup D \qquad \{l_1, \ldots, l_n, \bar{l}\} \in \Delta \qquad \bar{l}_1, \ldots, \bar{l}_n, \bar{l} \in \mathsf{M} \qquad \bar{l}_1, \ldots, \bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := \{l_1, \ldots, l_n\} \cup D}$$

Backjump
$$\frac{\mathsf{C} = \{l_1, \ldots, l_n, l\} \qquad \texttt{lev}(\bar{l}_1), \ldots, \texttt{lev}(\bar{l}_n) \leq i < \texttt{lev}(\bar{l})}{\mathsf{M} := \mathsf{M}^{[i]} l \qquad \mathsf{C} := \texttt{no}}$$

Conflict
$$\frac{\mathsf{C} = \texttt{no} \qquad \{l_1, \ldots, l_n\} \in \Delta \qquad \bar{l}_1, \ldots, \bar{l}_n \in \mathsf{M}}{\mathsf{C} := \{l_1, \ldots, l_n\}}$$

Modeling the Very Lazy Theory Approach

$\mathcal{T}$**-Conflict** is enough to model the naive integration of SAT solvers and theory solvers seen in the earlier EUF example

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

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

Modeling the Very Lazy Theory Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by Propagate^{+}
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by Decide
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{1} \vee 2 \vee 4$	by $\mathcal{T}$-Conflict
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee 2 \vee 4$	by Learn
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by Restart
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by Propagate^{+}
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee \bar{3} \vee 4$	by $\mathcal{T}$-Conflict
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4$	no	by Learn
	⋮		
	UNSAT		by Fail

Modeling the Very Lazy Theory Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by Propagate^{+}
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by Decide
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{1} \vee 2 \vee 4$	by $\mathcal{T}$-Conflict
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee 2 \vee 4$	by Learn
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by Restart
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by Propagate^{+}
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee \bar{3} \vee 4$	by $\mathcal{T}$-Conflict
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4$	no	by Learn
	$\vdots$		
	UNSAT		by Fail

Modeling the Very Lazy Theory Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Propagate**$^+$
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Decide**
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{1} \vee 2 \vee 4$	by $\mathcal{T}$-Conflict
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee 2 \vee 4$	by Learn
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by Restart
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by Propagate$^+$
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee \bar{3} \vee 4$	by $\mathcal{T}$-Conflict
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4$	no	by Learn
	$\vdots$		
	UNSAT		by Fail

Modeling the Very Lazy Theory Approach

$$\underbrace{g(a) \doteq c}_{1} \quad\wedge\quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad\wedge\quad \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Propagate**$^{+}$
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Decide**
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{1} \vee 2 \vee 4$	by $\mathcal{T}$-**Conflict**
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee 2 \vee 4$	by **Learn**
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by **Restart**
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by **Propagate**$^{+}$
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee \bar{3} \vee 4$	by $\mathcal{T}$-**Conflict**
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4$	no	by **Learn**
	⋮		
	UNSAT		by **Fail**

Modeling the Very Lazy Theory Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\overline{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\overline{4}}$$

M	Δ	C	rule
	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Propagate$^+$
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Decide
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee 2 \vee 4$	by $\mathcal{T}$-Conflict
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	$\overline{1} \vee 2 \vee 4$	by Learn
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	no	by Restart
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	no	by Propagate$^+$
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	$\overline{1} \vee \overline{3} \vee 4$	by $\mathcal{T}$-Conflict
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4,\ \overline{1} \vee \overline{3} \vee 4$	no	by Learn
	⋮		
	UNSAT		by Fail

Modeling the Very Lazy Theory Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\overline{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\overline{4}}$$

M	Δ	C	rule
	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Propagate^{+}
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Decide
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee 2 \vee 4$	by $\mathcal{T}$-Conflict
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	$\overline{1} \vee 2 \vee 4$	by Learn
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	no	by Restart
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	no	by Propagate^{+}
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	$\overline{1} \vee \overline{3} \vee 4$	by $\mathcal{T}$-Conflict
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4,\ \overline{1} \vee \overline{3} \vee 4$	no	by Learn
	⋮		
	UNSAT		by Fail

Modeling the Very Lazy Theory Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Propagate**$^{+}$
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Decide**
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{1} \vee 2 \vee 4$	by $\mathcal{T}$-**Conflict**
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee 2 \vee 4$	by **Learn**
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by **Restart**
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by **Propagate**$^{+}$
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee \bar{3} \vee 4$	by $\mathcal{T}$-**Conflict**
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4$	no	by **Learn**
	⋮		
	UNSAT		by **Fail**

Modeling the Very Lazy Theory Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Propagate**$^+$
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Decide**
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{1} \vee 2 \vee 4$	by $\mathcal{T}$-**Conflict**
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee 2 \vee 4$	by **Learn**
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by **Restart**
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by **Propagate**$^+$
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee \bar{3} \vee 4$	by $\mathcal{T}$-**Conflict**
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4$	no	by **Learn**
	$\vdots$		
	UNSAT		by **Fail**

Modeling the Very Lazy Theory Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\overline{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\overline{4}}$$

M	Δ	C	rule
	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Propagate^{+}
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Decide
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee 2 \vee 4$	by $\mathcal{T}$-Conflict
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	$\overline{1} \vee 2 \vee 4$	by Learn
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	no	by Restart
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	no	by Propagate^{+}
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4$	$\overline{1} \vee \overline{3} \vee 4$	by $\mathcal{T}$-Conflict
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2 \vee 4,\ \overline{1} \vee \overline{3} \vee 4$	no	by Learn
	$\vdots$		
	UNSAT		by Fail

Modeling the Very Lazy Theory Approach

$$\underbrace{g(a) \doteq c}_{1} \;\wedge\; \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \;\wedge\; \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Propagate**$^+$
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Decide**
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{1} \vee 2 \vee 4$	by $\mathcal{T}$-**Conflict**
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee 2 \vee 4$	by **Learn**
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by **Restart**
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	no	by **Propagate**$^+$
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4$	$\bar{1} \vee \bar{3} \vee 4$	by $\mathcal{T}$-**Conflict**
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4},\ \bar{1} \vee 2 \vee 4,\ \bar{1} \vee \bar{3} \vee 4$	no	by **Learn**
	⋮		
	UNSAT		by **Fail**

A Better Lazy Approach

The very lazy approach can be improved considerably with

- an *on-line* SAT engine that accept new input clauses on the fly
- an *incremental and explicating $\mathcal{T}$*-solver that can

A Better Lazy Approach

The very lazy approach can be improved considerably with

- an *on-line* SAT engine that accept new input clauses on the fly
- an *incremental and explicating* $\mathcal{T}$-solver that can
  1. check the $\mathcal{T}$-satisfiability of M as it is extended and
  2. identify a small $\mathcal{T}$-unsatisfiable subset of M once M becomes $\mathcal{T}$-unsatisfiable

A Better Lazy Approach

The very lazy approach can be improved considerably with

- an *on-line* SAT engine that accept new input clauses on the fly
- an *incremental and explicating* $\mathcal{T}$-solver that can
  1. check the $\mathcal{T}$-satisfiability of M as it is extended and
  2. identify a small $\mathcal{T}$-unsatisfiable subset of M once M becomes $\mathcal{T}$-unsatisfiable

A Better Lazy Approach

The very lazy approach can be improved considerably with

- an *on-line* SAT engine that accept new input clauses on the fly
- an *incremental and explicating* $\mathcal{T}$-solver that can
  1. check the $\mathcal{T}$-satisfiability of M as it is extended and
  2. identify a small $\mathcal{T}$-unsatisfiable subset of M once M becomes $\mathcal{T}$-unsatisfiable

A Better Lazy Approach

$$\underbrace{g(a) \doteq c}_{1} \;\land\; \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \lor \underbrace{g(a) \doteq d}_{3} \;\land\; \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \lor 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \lor 3,\ \bar{4}$	no	by Propagate^{+}
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \lor 3,\ \bar{4}$	no	by Decide
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \lor 3,\ \bar{4}$	$\bar{1} \lor 2$	by $\mathcal{T}$-Conflict
$1\ \bar{4}\ 2$	$1,\ \bar{2} \lor 3,\ \bar{4}$	no	by Backjump
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \lor 3,\ \bar{4}$	no	by Propagate
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \lor 3,\ \bar{4}$	$\bar{1} \lor \bar{3} \lor 4$	by $\mathcal{T}$-Conflict
UNSAT			by Fail

A Better Lazy Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by Propagate$^+$
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by Decide
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{1} \vee 2$	by $\mathcal{T}$-Conflict
$1\ \bar{4}\ 2$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by Backjump
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by Propagate
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{1} \vee \bar{3} \vee 4$	by $\mathcal{T}$-Conflict
UNSAT			by Fail

A Better Lazy Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\overline{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\overline{4}}$$

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

A Better Lazy Approach

$$\underbrace{g(a) \doteq c}_{1} \;\wedge\; \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \;\wedge\; \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Propagate**$^{+}$
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Decide**
$1\ \bar{4} \bullet \bar{2}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{1} \vee 2$	by $\mathcal{T}$-Conflict
$1\ \bar{4}\ 2$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Backjump**
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Propagate**
$1\ \bar{4}\ 2\ 3$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{1} \vee \bar{3} \vee 4$	by $\mathcal{T}$-Conflict
UNSAT			by **Fail**

A Better Lazy Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\overline{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\overline{4}}$$

M	Δ	C	rule
	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Propagate^{+}
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Decide
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee 2$	by $\mathcal{T}$-Conflict
$1\ \overline{4}\ 2$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Backjump
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Propagate
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee \overline{3} \vee 4$	by $\mathcal{T}$-Conflict
UNSAT			by Fail

A Better Lazy Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\overline{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\overline{4}}$$

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

A Better Lazy Approach

$$\underbrace{g(a) \doteq c}_{1} \;\wedge\; \underbrace{f(g(a)) \not\doteq f(c)}_{\overline{2}} \vee \underbrace{g(a) \doteq d}_{3} \;\wedge\; \underbrace{c \not\doteq d}_{\overline{4}}$$

M	Δ	C	rule
	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by PROPAGATE^{+}
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by DECIDE
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee 2$	by $\mathcal{T}$-CONFLICT
$1\ \overline{4}\ 2$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by BACKJUMP
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by PROPAGATE
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee \overline{3} \vee 4$	by $\mathcal{T}$-CONFLICT
UNSAT			by FAIL

A Better Lazy Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\overline{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\overline{4}}$$

M	Δ	C	rule
	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Propagate$^+$
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Decide
$1\ \overline{4} \bullet \overline{2}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee 2$	by $\mathcal{T}$-Conflict
$1\ \overline{4}\ 2$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Backjump
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by Propagate
$1\ \overline{4}\ 2\ 3$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{1} \vee \overline{3} \vee 4$	by $\mathcal{T}$-Conflict
UNSAT			by Fail

A Better Lazy Approach

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\overline{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\overline{4}}$$

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

Lazy Approach – Strategies

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

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

Note: Depending on the cost of checking the $\mathcal{T}$-satisfiability of M, Step (2) can be applied with lower frequency or priority

Lazy Approach – Strategies

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

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

Note: Depending on the cost of checking the $\mathcal{T}$-satisfiability of M, Step (2) can be applied with lower frequency or priority

Theory Propagation

With $\mathcal{T}$-**Conflict** as the only theory rule, the theory solver is used just to validate the choices of the SAT engine

With $\mathcal{T}$-**Propagate** and $\mathcal{T}$-**Explain**, it can also be used to guide the engine's search

$$\mathcal{T}\text{-}\textbf{Propagate}\ \frac{l \in \mathtt{Lits}(\Delta) \qquad \mathsf{M} \models_{\mathcal{T}} l \qquad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\ l}$$

$$\mathcal{T}\text{-}\textbf{Explain}\ \frac{\mathsf{C} = \{l\} \cup D \qquad \bar{l}_1 \wedge \cdots \wedge \bar{l}_n \models_{\mathcal{T}} \bar{l} \qquad \bar{l}_1, \ldots, \bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := \{l_1, \cdots, l_n\} \cup D}$$

Theory Propagation

With $\mathcal{T}$-**Conflict** as the only theory rule, the theory solver is used just to validate the choices of the SAT engine

With $\mathcal{T}$-**Propagate** and $\mathcal{T}$-**Explain**, it can also be used to guide the engine's search

$$\mathcal{T}\text{-}\textbf{Propagate}\ \frac{l \in \texttt{Lits}(\Delta) \qquad \mathsf{M} \models_{\mathcal{T}} l \qquad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\ l}$$

$$\mathcal{T}\text{-}\textbf{Explain}\ \frac{\mathsf{C} = \{l\} \cup D \qquad \bar{l}_1 \wedge \cdots \wedge \bar{l}_n \models_{\mathcal{T}} \bar{l} \qquad \bar{l}_1, \ldots, \bar{l}_n \prec_{\mathsf{M}} \bar{l}}{\mathsf{C} := \{l_1, \cdots, l_n\} \cup D}$$

Theory Propagation Example

$$\underbrace{g(a) \doteq c}_{1} \quad \land \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\overline{2}} \lor \underbrace{g(a) \doteq d}_{3} \quad \land \quad \underbrace{c \not\doteq d}_{\overline{4}}$$

M	Δ	C	rule
	$1,\ \overline{2} \lor 3,\ \overline{4}$	no	
$1\ \overline{4}$	$1,\ \overline{2} \lor 3,\ \overline{4}$	no	by Propagate$^+$
$1\ \overline{4}\ 2$	$1,\ \overline{2} \lor 3,\ \overline{4}$	no	by $\mathcal{T}$-Propagate (as $1 \models_{\mathcal{T}} 2$)
$1\ \overline{4}\ 2\ \overline{3}$	$1,\ \overline{2} \lor 3,\ \overline{4}$	no	by $\mathcal{T}$-Propagate (as $1,\ \overline{4} \models_{\mathcal{T}} \overline{3}$)
$1\ \overline{4}\ 2\ \overline{3}$	$1,\ \overline{2} \lor 3,\ \overline{4}$	$\overline{2} \lor 3$	by Conflict
UNSAT			by Fail

Note: $\mathcal{T}$-propagation eliminates search altogether in this case!
No applications of **Decide** are needed

Theory Propagation Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by Propagate^{+}
$1\ \bar{4}\ 2$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by $\mathcal{T}$-Propagate (as $1 \models_{\mathcal{T}} 2$)
$1\ \bar{4}\ 2\ \bar{3}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by $\mathcal{T}$-Propagate (as $1,\ \bar{4} \models_{\mathcal{T}} \bar{3}$)
$1\ \bar{4}\ 2\ \bar{3}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{2} \vee 3$	by Conflict
UNSAT			by Fail

Note: $\mathcal{T}$-propagation eliminates search altogether in this case!
No applications of **Decide** are needed

Theory Propagation Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule	
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no		
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Propagate**$^+$	
$1\ \bar{4}\ 2$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by $\mathcal{T}$-**Propagate**	(as $1 \models_{\mathcal{T}} 2$)
$1\ \bar{4}\ 2\ \bar{3}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by $\mathcal{T}$-**Propagate**	(as $1,\ \bar{4} \models_{\mathcal{T}} \bar{3}$)
$1\ \bar{4}\ 2\ \bar{3}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{2} \vee 3$	by **Conflict**	
UNSAT			by **Fail**	

Note: $\mathcal{T}$-propagation eliminates search altogether in this case!
No applications of **Decide** are needed

Theory Propagation Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by Propagate^{+}
$1\ \bar{4}\ 2$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by $\mathcal{T}$-Propagate (as $1 \models_{\mathcal{T}} 2$)
$1\ \bar{4}\ 2\ \bar{3}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by $\mathcal{T}$-Propagate (as $1, \bar{4} \models_{\mathcal{T}} \bar{3}$)
$1\ \bar{4}\ 2\ \bar{3}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{2} \vee 3$	by Conflict
UNSAT			by Fail

Note: $\mathcal{T}$-propagation eliminates search altogether in this case!
No applications of **Decide** are needed

Theory Propagation Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\overline{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\overline{4}}$$

M	Δ	C	rule	
	$1,\ \overline{2} \vee 3,\ \overline{4}$	no		
$1\ \overline{4}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by **Propagate**$^{+}$	
$1\ \overline{4}\ 2$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by $\mathcal{T}$-**Propagate**	(as $1 \models_{\mathcal{T}} 2$)
$1\ \overline{4}\ 2\ \overline{3}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	no	by $\mathcal{T}$-**Propagate**	(as $1,\ \overline{4} \models_{\mathcal{T}} \overline{3}$)
$1\ \overline{4}\ 2\ \overline{3}$	$1,\ \overline{2} \vee 3,\ \overline{4}$	$\overline{2} \vee 3$	by **Conflict**	
UNSAT			by **Fail**	

Note: $\mathcal{T}$-propagation eliminates search altogether in this case!
No applications of **Decide** are needed

Theory Propagation Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Propagate**$^+$
$1\ \bar{4}\ 2$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by $\mathcal{T}$-**Propagate** (as $1 \models_{\mathcal{T}} 2$)
$1\ \bar{4}\ 2\ \bar{3}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by $\mathcal{T}$-**Propagate** (as $1,\ \bar{4} \models_{\mathcal{T}} \bar{3}$)
$1\ \bar{4}\ 2\ \bar{3}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{2} \vee 3$	by **Conflict**
UNSAT			by **Fail**

Note: $\mathcal{T}$-propagation eliminates search altogether in this case!
No applications of **Decide** are needed

Theory Propagation Example

$$\underbrace{g(a) \doteq c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \not\doteq f(c)}_{\bar{2}} \vee \underbrace{g(a) \doteq d}_{3} \quad \wedge \quad \underbrace{c \not\doteq d}_{\bar{4}}$$

M	Δ	C	rule
	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	
$1\ \bar{4}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by **Propagate**$^+$
$1\ \bar{4}\ 2$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by $\mathcal{T}$-**Propagate** (as $1 \models_{\mathcal{T}} 2$)
$1\ \bar{4}\ 2\ \bar{3}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	no	by $\mathcal{T}$-**Propagate** (as $1,\ \bar{4} \models_{\mathcal{T}} \bar{3}$)
$1\ \bar{4}\ 2\ \bar{3}$	$1,\ \bar{2} \vee 3,\ \bar{4}$	$\bar{2} \vee 3$	by **Conflict**
UNSAT			by **Fail**

Note: $\mathcal{T}$-propagation eliminates search altogether in this case!
No applications of **Decide** are needed

Theory Propagation Features

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

- For theory propagation to be effective in practice, it needs specialized theory solvers

- For some theories, e.g., difference logic, detecting $\mathcal{T}$-entailed literals is cheap and so exhaustive theory propagation is extremely effective

- For others, e.g., the theory of equality, detecting $\mathcal{T}$-entailed equalities is cheap but detecting $\mathcal{T}$-entailed disequalities is quite expensive

- If $\mathcal{T}$-**Propagate** is not applied exhaustively, $\mathcal{T}$-**Conflict** is needed to repair $\mathcal{T}$-unsatisfiable assignments

Theory Propagation Features

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

- For theory propagation to be effective in practice, it needs specialized theory solvers

- For some theories, e.g., difference logic, detecting $\mathcal{T}$-entailed literals is cheap and so exhaustive theory propagation is extremely effective

- For others, e.g., the theory of equality, detecting $\mathcal{T}$-entailed equalities is cheap but detecting $\mathcal{T}$-entailed disequalities is quite expensive

- If $\mathcal{T}$-**Propagate** is not applied exhaustively, $\mathcal{T}$-**Conflict** is needed to repair $\mathcal{T}$-unsatisfiable assignments

Theory Propagation Features

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

- For theory propagation to be effective in practice, it needs specialized theory solvers

- For some theories, e.g., difference logic, detecting $\mathcal{T}$-entailed literals is cheap and so exhaustive theory propagation is extremely effective

- For others, e.g., the theory of equality, detecting $\mathcal{T}$-entailed equalities is cheap but detecting $\mathcal{T}$-entailed disequalities is quite expensive

- If $\mathcal{T}$-**Propagate** is not applied exhaustively, $\mathcal{T}$-**Conflict** is needed to repair $\mathcal{T}$-unsatisfiable assignments

Theory Propagation Features

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

Theory Propagation Features

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

- For theory propagation to be effective in practice, it needs specialized theory solvers

- For some theories, e.g., difference logic, detecting $\mathcal{T}$-entailed literals is cheap and so exhaustive theory propagation is extremely effective

- For others, e.g., the theory of equality, detecting $\mathcal{T}$-entailed equalities is cheap but detecting $\mathcal{T}$-entailed disequalities is quite expensive

- If $\mathcal{T}$**-Propagate** is not applied exhaustively, $\mathcal{T}$**-Conflict** is needed to repair $\mathcal{T}$-unsatisfiable assignments

Theory Propagation Exercise

$$\underbrace{a \doteq b}_{1} \quad \wedge \quad \underbrace{a \doteq c}_{2} \vee \underbrace{c \doteq b}_{3} \quad \wedge \quad \underbrace{a \not\doteq b}_{\bar{1}} \vee \underbrace{f(a) \not\doteq f(c)}_{\bar{4}} \quad \wedge \quad \underbrace{c \not\doteq b}_{\bar{3}} \vee \underbrace{g(a) \doteq g(c)}_{5}$$

$$\Delta_0 \ := \ 1, \ 2 \vee 3, \ \bar{1} \vee \bar{4}, \ \bar{3} \vee 5$$

Theory Propagation Exercise

Scenario 1: propagating only $\mathcal{T}$-entailed equalities (no disequalities)

$$\underbrace{a \doteq b}_{1} \quad \wedge \quad \underbrace{a \doteq c}_{2} \vee \underbrace{c \doteq b}_{3} \quad \wedge \quad \underbrace{a \not\doteq b}_{\bar{1}} \vee \underbrace{f(a) \not\doteq f(c)}_{\bar{4}} \quad \wedge \quad \underbrace{c \not\doteq b}_{\bar{3}} \vee \underbrace{g(a) \doteq g(c)}_{5}$$

$$\Delta_0 \; := \; 1, \; 2 \vee 3, \; \bar{1} \vee \bar{4}, \; \bar{3} \vee 5$$

Theory Propagation Exercise

Scenario 1: propagating only $\mathcal{T}$-entailed equalities (no disequalities)

$$\underbrace{a \doteq b}_{1} \quad \wedge \quad \underbrace{a \doteq c}_{2} \vee \underbrace{c \doteq b}_{3} \quad \wedge \quad \underbrace{a \not\doteq b}_{\bar{1}} \vee \underbrace{f(a) \not\doteq f(c)}_{\bar{4}} \quad \wedge \quad \underbrace{c \not\doteq b}_{\bar{3}} \vee \underbrace{g(a) \doteq g(c)}_{5}$$

$$\Delta_0 := 1,\ 2 \vee 3,\ \bar{1} \vee \bar{4},\ \bar{3} \vee 5$$

M	Δ	C	rule
	Δ_0	no	
$1\ \bar{4}$	Δ_0	no	by **Propagate**$^+$
$1\ \bar{4}\ \bullet\ 2$	Δ_0	no	by **Decide**
$1\ \bar{4}\ \bullet\ 2$	Δ_0	$\bar{2} \vee 4$	by **$\mathcal{T}$-Conflict** (as $2, \bar{4} \models_{\mathcal{T}} \bot$)
$1\ \bar{4}\ \bar{2}$	Δ_0	no	by **Backjump**
$1\ \bar{4}\ \bar{2}\ 3$	Δ_0	no	by **Propagate**
$1\ \bar{4}\ \bar{2}\ 3$	Δ_0	$\bar{2} \vee 4$	by **$\mathcal{T}$-Conflict** (as $1, \bar{3}, \bar{4} \models_{\mathcal{T}} \bot$)
UNSAT			by **Fail**

Theory Propagation Exercise

Scenario 2: propagating $\mathcal{T}$-entailed equalities and disequalities

$$\underbrace{a \doteq b}_{1} \quad \wedge \quad \underbrace{a \doteq c}_{2} \vee \underbrace{c \doteq b}_{3} \quad \wedge \quad \underbrace{a \not\doteq b}_{\bar{1}} \vee \underbrace{f(a) \not\doteq f(c)}_{\bar{4}} \quad \wedge \quad \underbrace{c \not\doteq b}_{\bar{3}} \vee \underbrace{g(a) \doteq g(c)}_{5}$$

$$\Delta_0 \;:=\; 1,\; 2 \vee 3,\; \bar{1} \vee \bar{4},\; \bar{3} \vee 5$$

Theory Propagation Exercise

Scenario 2: propagating $\mathcal{T}$-entailed equalities and disequalities

$$\underbrace{a \doteq b}_{1} \quad \wedge \quad \underbrace{a \doteq c}_{2} \vee \underbrace{c \doteq b}_{3} \quad \wedge \quad \underbrace{a \not\doteq b}_{\bar{1}} \vee \underbrace{f(a) \not\doteq f(c)}_{\bar{4}} \quad \wedge \quad \underbrace{c \not\doteq b}_{\bar{3}} \vee \underbrace{g(a) \doteq g(c)}_{5}$$

$$\Delta_0 \;:=\; 1,\;\; 2 \vee 3,\;\; \bar{1} \vee \bar{4},\;\; \bar{3} \vee 5$$

M	Δ	C	rule
	Δ_0	no	
$1\ \bar{4}$	Δ_0	no	by **Propagate**$^+$
$1\ \bar{4}\ \bar{2}$	Δ_0	no	by $\mathcal{T}$-**Propagate** (as $1, \bar{4} \models_{\mathcal{T}} \bar{2}$)
$1\ \bar{4}\ \bar{2}\ 3$	Δ_0	no	by **Propagate**
$1\ \bar{4}\ \bar{2}\ 3$	Δ_0	$\bar{1} \vee \bar{3} \vee 4$	by $\mathcal{T}$-**Conflict** (as $1, 3, \bar{4} \models_{\mathcal{T}} \bot$)
UNSAT			by **Fail**

Modeling Modern Lazy SMT Solvers

At the core, current lazy SMT solvers are implementations of the proof system with rules:

(1) **Propagate**, **Decide**, **Conflict**, **Explain**, **Backjump**, **Fail**

(2) $\mathcal{T}$-**Conflict**, $\mathcal{T}$-**Propagate**, $\mathcal{T}$-**Explain**

(3) **Learn**, **Forget**, **Restart**

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

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

Modeling Modern Lazy SMT Solvers

At the core, current lazy SMT solvers are implementations of the proof system with rules:

(1) **Propagate**, **Decide**, **Conflict**, **Explain**, **Backjump**, **Fail**

(2) $\mathcal{T}$**-Conflict**, $\mathcal{T}$**-Propagate**, $\mathcal{T}$**-Explain**

(3) **Learn**, **Forget**, **Restart**

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

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

Correctness

Updated terminology:

- *Irreducible state:* state to which no Basic CDCL Modulo Theories rules apply
- *Execution:* a (single-branch) derivation tree starting with $\mathsf{M} = \emptyset$ and $\mathsf{C} = \mathtt{no}$
- *Exhausted execution:* execution ending in an irreducible state

Theorem 2 (Strong Termination)

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

Correctness

Updated terminology:

- *Irreducible state:* state to which no Basic CDCL Modulo Theories rules apply
- *Execution:* a (single-branch) derivation tree starting with $M = \emptyset$ and $C = \texttt{no}$
- *Exhausted execution:* execution ending in an irreducible state

Theorem 2 (Strong Termination)

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

Correctness

Updated terminology:

- *Irreducible state:* state to which no Basic CDCL Modulo Theories rules apply
- *Execution:* a (single-branch) derivation tree starting with $\mathsf{M} = \emptyset$ and $\mathsf{C} = \mathtt{no}$
- *Exhausted execution:* execution ending in an irreducible state

Theorem 2 (Strong Termination)

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

Lemma 3

Every exhausted execution ends with either $\mathsf{C} = no$ *or* UNSAT.

Correctness

Updated terminology:

Irreducible state: state to which no Basic CDCL Modulo Theories rules apply

Execution: a (single-branch) derivation tree starting with $M = \emptyset$ and $C = \texttt{no}$

Exhausted execution: execution ending in an irreducible state

Theorem 2 (Strong Termination)

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

Theorem 3 (Refutation Soundness)

For every exhausted execution starting with $\Delta = \Delta_0$ and ending with UNSAT*, the clause set Δ_0 is $\mathcal{T}$-unsatisfiable.*

Correctness

Updated terminology:

Irreducible state: state to which no Basic CDCL Modulo Theories rules apply

Execution: a (single-branch) derivation tree starting with $\mathsf{M} = \emptyset$ and $\mathsf{C} = \mathtt{no}$

Exhausted execution: execution ending in an irreducible state

Theorem 2 (Strong Termination)

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

Theorem 3 (Refutation Soundness)

For every exhausted execution starting with $\Delta = \Delta_0$ and ending with UNSAT, the clause set Δ_0 is $\mathcal{T}$-unsatisfiable.

Theorem 4 (Refutation Completeness)

For every exhausted execution starting with $\Delta = \Delta_0$ and ending with $\mathsf{C} = no$, the clause set Δ_0 is $\mathcal{T}$-satisfiable; specifically, M is $\mathcal{T}$-satisfiable and $\mathsf{M} \models_{\mathrm{p}} \Delta_0$.

CDCL($\mathcal{T}$) Architecture

The approach formalized so far can be implemented with a simple architecture originally named DPLL(T) but currently known as *CDCL($\mathcal{T}$)*

$$\text{CDCL}(\mathcal{T}) \;=\; \text{CDCL}(X)\text{ engine} \;+\; \mathcal{T}\text{-solver}$$

CDCL($\mathcal{T}$) Architecture

The approach formalized so far can be implemented with a simple architecture originally named DPLL($\mathcal{T}$) but currently known as *CDCL($\mathcal{T}$)*

CDCL($\mathcal{T}$) = CDCL(X) engine + $\mathcal{T}$-solver

CDCL(X):

- Very similar to a SAT solver, enumerates Boolean models
- Not allowed: pure literal rule (and other SAT specific optimizations)
- Required: incremental addition of clauses
- Desirable: partial model detection

CDCL($\mathcal{T}$) Architecture

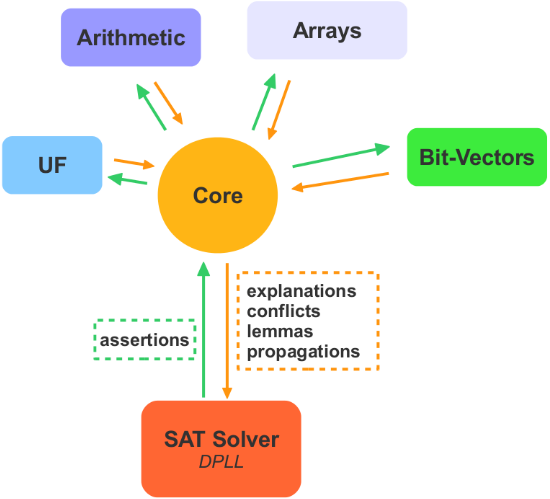
The approach formalized so far can be implemented with a simple architecture originally named DPLL($\mathcal{T}$) but currently known as *CDCL($\mathcal{T}$)*

$$\text{CDCL}(\mathcal{T}) \;=\; \text{CDCL}(X) \text{ engine} \;+\; \mathcal{T}\text{-solver}$$

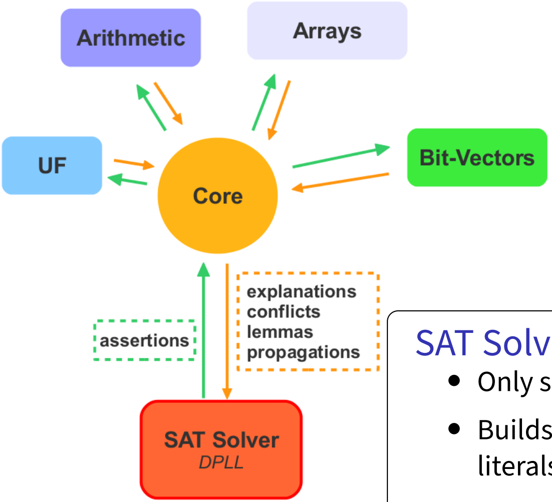
$\mathcal{T}$-solver:

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

Typical SMT solver architecture



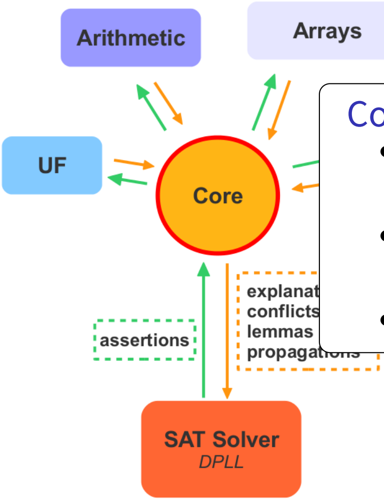
Typical SMT solver architecture



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*

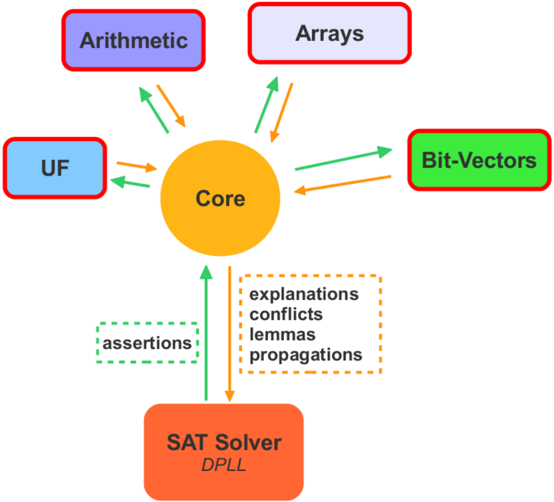
Typical SMT solver architecture



Core

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

Typical SMT solver architecture



Theory Solvers

- Check $\mathcal{T}$-satisfiability of sets of theory literals
- Incremental
- Backtrackable
- Conflict Generation
- Theory Propagation