

#### Formalizing DPLL-based Solvers for Propositional Satisfiability and for Satisfiability Modulo Theories

#### **Cesare Tinelli**

#### (joint work with Robert Nieuwenhuis and Albert Oliveras)

tinelli@cs.uiowa.edu

The University of Iowa



- Observe to the set of a propositional formula is a well-studied and important problem.
- 6 Theoretical interest: first established NP-Complete problem, phase transition, ...
- Operation of the second structure of the second str
  - Development of algorithms and enhancements.
  - Implementation of extremely efficient tools.
  - Solvers based on the DPLL procedure have been the most successful so far.

6 Any SAT solver can be used to decide the satisfiability of ground (i.e., variable-free) first-order formulas.

- 6 Any SAT solver can be used to decide the satisfiability of ground (i.e., variable-free) first-order formulas.
- Often, however, one is interested in the satisfiability of certain ground formulas in a given first-order theory:

- 6 Any SAT solver can be used to decide the satisfiability of ground (i.e., variable-free) first-order formulas.
- Often, however, one is interested in the satisfiability of certain ground formulas in a given first-order theory:
  - △ Pipelined microprocessors: theory of equality, atoms like f(g(a, b), c) = g(c, a).

- 6 Any SAT solver can be used to decide the satisfiability of ground (i.e., variable-free) first-order formulas.
- Often, however, one is interested in the satisfiability of certain ground formulas in a given first-order theory:
  - △ Pipelined microprocessors: theory of equality, atoms like f(g(a, b), c) = g(c, a).
  - ▲ Timed automata, planning: theory of integers/reals, atoms like x y < 2.

- 6 Any SAT solver can be used to decide the satisfiability of ground (i.e., variable-free) first-order formulas.
- Often, however, one is interested in the satisfiability of certain ground formulas in a given first-order theory:
  - △ Pipelined microprocessors: theory of equality, atoms like f(g(a, b), c) = g(c, a).
  - ▲ Timed automata, planning: theory of integers/reals, atoms like x y < 2.
  - Software verification: combination of theories, atoms like 5 + car(a+2) = cdr(a[j]+1).

- 6 Any SAT solver can be used to decide the satisfiability of ground (i.e., variable-free) first-order formulas.
- Often, however, one is interested in the satisfiability of certain ground formulas in a given first-order theory:
  - △ Pipelined microprocessors: theory of equality, atoms like f(g(a, b), c) = g(c, a).
  - ▲ Timed automata, planning: theory of integers/reals, atoms like x y < 2.
  - Software verification: combination of theories, atoms like 5 + car(a+2) = cdr(a[j]+1).
- 6 We refer to this general problems as (ground) Satisfiability Modulo Theories, or SMT.



- 6 Note: The *T*-satisfiability of ground formulas is decidable iff the *T*-satisfiability of sets of literals is decidable.
- Fact: Many theories of interest have (efficient) decision procedures for sets of literals.
- Or Problem: In practice, dealing with Boolean combinations of literals is as hard as in the propositional case.
- 6 Current solution: Exploit propositional satisfiability technology.

## Lifting SAT to SMT

- 6 Eager approach [UCLID]:
  - translate into an equisatisfiable propositional formula,
  - feed it to any SAT solver.
- **Lazy approach** [CVC, ICS, MathSAT, Verifun, Zap]:
  - abstract the input formula into a propositional one,
  - feed it to a DPLL-based SAT solver,
  - use a theory decision procedure to refine the formula.
- **OPLL(T)** [DPLLT, Sammy]:
  - use the decision procedure to guide the search of a DPLL solver.



Develop a declarative formal framework to:

- 6 Reason formally about DPLL-based solvers for SAT and for SMT.
- Model modern features such as non-chronological bactracking, lemma learning or restarts.
- Obscribe different strategies and prove their correctness.
- 6 Compare different systems at a higher level.
- 6 Get new insights for further enhancements of DPPL solvers.



- 6 Motivation: SAT and SMT
- 6 The DPLL procedure
- 6 An Abstract Framework
- SAT case
  - The Original DPLL Procedure
  - The Basic and the Enhanced DPLL System
- SMT case
  - Very Lazy Theory Learning
  - Lazy Theory Learning
  - Theory Propagation



- 6 Tries to build incrementally a satisfying truth assignment M for a CNF formula F.
- 6 M is grown by
  - $\land$  deducing the truth value of a literal from M and F, or
  - guessing a truth value.
- If a wrong guess for a literal leads to an inconsistency, the procedure backtracks and tries the opposite value.



# OperationAssign.Formula $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$



| Operation | Assign. | Formula   |
|-----------|---------|---|
|           |         | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 1  |         | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |



| Operation |      |   |
|-----------|------|---|
|           |      | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 1  | 1    | $1 \lor 2, 2 \lor \overline{3} \lor 4, \overline{1} \lor \overline{2}, \overline{1} \lor \overline{3} \lor \overline{4}, 1$         |
| deduce 2  | 1, 2 | $\begin{array}{c ccccccccccccccccccccccccccccccccccc$   |



| Operation |       | Formula   |
|-----------|-------|---|
|           |       | $1 \lor 2, 2 \lor \overline{3} \lor 4, \overline{1} \lor \overline{2}, \overline{1} \lor \overline{3} \lor \overline{4}, 1$         |
| deduce 1  | 1     | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 2  | 1, 2  | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| guess 3   | 1,2,3 | $ \begin{array}{cccccccccccccccccccccccccccccccccccc$   |



| Operation | -       | Formula   |
|-----------|---------|---|
|           |         | $ \begin{array}{cccccccccccccccccccccccccccccccccccc$   |
| deduce 1  | 1       | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 2  | 1, 2    | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| guess 3   | 1,2,3   | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 4  | 1,2,3,4 | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |



| Operation | Assign. | Formula   |
|-----------|---------|---|
|           |         | $1 \lor 2, 2 \lor \overline{3} \lor 4, \overline{1} \lor \overline{2}, \overline{1} \lor \overline{3} \lor \overline{4}, 1$         |
| deduce 1  | 1       | $1 \lor 2, 2 \lor \overline{3} \lor 4, \overline{1} \lor \overline{2}, \overline{1} \lor \overline{3} \lor \overline{4}, 1$         |
| deduce 2  | 1, 2    | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| guess 3   | 1, 2, 3 | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 4  | 1,2,3,4 | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |

**Inconsistency!** 

| Operation | Assign. | Formula   |
|-----------|---------|---|
|           |         | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 1  | 1       | $1 \lor 2, 2 \lor \overline{3} \lor 4, \overline{1} \lor \overline{2}, \overline{1} \lor \overline{3} \lor \overline{4}, 1$         |
| deduce 2  | 1, 2    | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| guess 3   | 1,2,3   | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 4  | 1,2,3,4 | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| undo 3    | 1,2     | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |

| Operation | Assign. | Formula   |
|-----------|---------|---|
|           |         | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 1  | 1       | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 2  | 1, 2    | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| guess 3   | 1,2,3   | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 4  | 1,2,3,4 | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| undo 3    | 1, 2    | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| guess 3   | 1,2,3   | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |

| Operation | Assign. | Formula   |
|-----------|---------|---|
|           |         | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 1  | 1       | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 2  | 1, 2    | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| guess 3   | 1,2,3   | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| deduce 4  | 1,2,3,4 | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |
| undo 3    | 1, 2    | $1 \lor 2, 2 \lor \overline{3} \lor 4, \overline{1} \lor \overline{2}, \overline{1} \lor \overline{3} \lor \overline{4}, 1$         |
| guess 3   | 1,2,3   | $1 \lor 2, \ 2 \lor \overline{3} \lor 4, \ \overline{1} \lor \overline{2}, \ \overline{1} \lor \overline{3} \lor \overline{4}, \ 1$ |

Model Found!



- 6 Motivation: SAT and SMT
- 6 The DPLL procedure
- 6 An Abstract Framework
- SAT case
  - The Original DPLL Procedure
  - The Basic and the Enhanced DPLL System
- 6 SMT case
  - Very Lazy Theory Learning
  - Lazy Theory Learning
  - Theory Propagation



- 6 The DPLL procedure can be described declaratively by simple sequent-style calculi.
- Such calculi however cannot model meta-logical features such as backtracking, learning and restarts.
- We model DPLL and its enhancements as transition systems instead.
- 6 A transition system is a binary relation over states, induced by a set of conditional transition rules.



Our states:

#### fail or $M \parallel F$

where F is a CNF formula, a set of clauses, and
M is a sequence of annotated literals
denoting a partial truth assignment.



Our states:

fail or 
$$M \parallel F$$

Initial state:

6  $\emptyset \parallel F$ , where F is to be checked for satisfiability.

**Expected final states:** 

- 6 fail, if F is unsatisfiable
  - $M \parallel G$ , where M is a model of G and G is logically equivalent to F.



Extending the assignment:

UnitProp

$$M \parallel F, C \lor l \rightarrow M l \parallel F, C \lor l \quad \text{if } \begin{cases} M \models \neg C, \\ l \text{ is undefined in } M \end{cases}$$



Extending the assignment:

#### UnitProp

$$M \parallel F, C \lor l \rightarrow M \ l \parallel F, C \lor l \quad \text{if } \begin{cases} M \models \neg C, \\ l \text{ is undefined in } M \end{cases}$$

#### Decide

$$M \parallel F \longrightarrow M l^{\mathsf{d}} \parallel F \quad \mathsf{if} \begin{cases} l \text{ or } \overline{l} \text{ occurs in } F, \\ l \text{ is undefined in } M \end{cases}$$

**Notation:**  $l^d$  annotates l as a decision literal.



Repairing the assignment:

#### Fail

$$M \parallel F, C \rightarrow fail \quad \text{if } \begin{cases} M \models \neg C, \\ M \text{ contains no decision literals} \end{cases}$$



Repairing the assignment:

#### Fail

$$M \parallel F, C \rightarrow fail \quad \text{if } \begin{cases} M \models \neg C, \\ M \text{ contains no decision literals} \end{cases}$$

#### Backtrack

$$M l^{\mathsf{d}} N \parallel F, C \rightarrow M \overline{l} \parallel F, C \quad \text{if } \begin{cases} M l^{\mathsf{d}} N \models \neg C, \\ l \text{ last decision literal} \end{cases}$$



- 6 Motivation: SAT and SMT
- 6 The DPLL procedure
- 6 An Abstract Framework
- 6 SAT case
  - △ The Original DPLL Procedure
  - △ The Basic and the Enhanced DPLL System
- 6 SMT case
  - Very Lazy Theory Learning
  - Lazy Theory Learning
  - Theory Propagation



Backtrack

## $M l^{\mathsf{d}} N \parallel F, C \rightarrow M \overline{l} \parallel F, C \quad \text{if } \begin{cases} M l^{\mathsf{d}} N \models \neg C, \\ l \text{ last decision literal} \end{cases}$

#### From Backtracking to Backjumping

**Backtrack** 

$$M l^{\mathsf{d}} N \parallel F, C \rightarrow M \overline{l} \parallel F, C \quad \text{if } \begin{cases} M l^{\mathsf{d}} N \models \neg C, \\ l \text{ last decision literal} \end{cases}$$

Backjump

 $\begin{cases} 1. \ M \ l^{\mathsf{d}} \ N \models \neg C, \\ 2. \ \text{for some clause } D \lor k : \end{cases}$  $Ml^{\mathsf{d}} N \parallel F, C \rightarrow Mk \parallel F, C \quad \text{if} \begin{cases} F, C \vdash L, \ldots, \\ M \models \neg D, \\ k \text{ is undefined in } M, \end{cases}$ k or  $\overline{k}$  occurs in  $M l^{\mathsf{d}} N \parallel F, C$ 



Backtrack

$$M l^{\mathsf{d}} N \parallel F, C \rightarrow M \overline{l} \parallel F, C \text{ if } \begin{cases} M l^{\mathsf{d}} N \models \neg C, \\ l \text{ last decision literal} \end{cases}$$

1.12 B

$$M l^{\mathsf{d}} N \parallel F, C \rightarrow M k \parallel F, C \quad \text{if} \begin{cases} 1. \ M l^{\mathsf{d}} N \models \neg C, \\ 2. \text{ for some clause } D \lor k; \\ F, C \vdash D \lor k, \\ M \models \neg D, \\ k \text{ is undefined in } M, \\ k \text{ or } \overline{k} \text{ occurs in} \\ M l^{\mathsf{d}} N \parallel F, C \end{cases}$$

**Note:** Condition (1) is actually not necessary.



At the core, current DPLL-based SAT solvers are implementations of the transition system:

#### **Basic DPLL**

- 6 UnitProp
- 6 Decide
- 🍯 Fail
- 6 Backjump

#### The Basic DPLL System – Correctness

Some terminology

Irreducible state: state to which no transition rule applies.

**Execution:** sequence of transitions allowed by the rules and starting with states of the form  $\emptyset \parallel F$ .

Exhausted execution: execution ending in an irreducible state

#### The Basic DPLL System – Correctness

Some terminology

Irreducible state: state to which no transition rule applies.

**Execution:** sequence of transitions allowed by the rules and starting with states of the form  $\emptyset \parallel F$ .

Exhausted execution: execution ending in an irreducible state

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

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

#### The Basic DPLL System – Correctness

Some terminology

Irreducible state: state to which no transition rule applies.

**Execution:** sequence of transitions allowed by the rules and starting with states of the form  $\emptyset \parallel F$ .

Exhausted execution: execution ending in an irreducible state

**Proposition** (Soundness) For every exhausted execution starting with  $\emptyset \parallel F$  and ending in  $M \parallel F$ ,  $M \models F$ .

**Proposition** (Completeness) If *F* is unsatisfiable, every exhausted execution starting with  $\emptyset \parallel F$  ends with *fail*.





### $M \parallel F \rightarrow M \parallel F, C \text{ if } \begin{cases} \text{all atoms of } C \text{ occur in } F, \\ F \vdash C \end{cases}$



## $M \parallel F \rightarrow M \parallel F, C \text{ if } \begin{cases} \text{all atoms of } C \text{ occur in } F, \\ F \vdash C \end{cases}$

#### Forget $M \parallel F, C \rightarrow M \parallel F \text{ if } F \vdash C$



$$M \parallel F \rightarrow M \parallel F, C \quad \text{if } \begin{cases} \text{all atoms of } C \text{ occur in } F, \\ F \vdash C \end{cases}$$

#### Forget $M \parallel F, C \rightarrow M \parallel F \text{ if } F \vdash C$

Usually *C* is a clause identified during conflict analysis.



$$M \parallel F \rightarrow M \parallel F, C \quad \text{if } \begin{cases} \text{all atoms of } C \text{ occur in } F, \\ F \vdash C \end{cases}$$

#### Forget

 $M \parallel F, C \quad \to \quad M \parallel F \quad \text{if } F \vdash C$ 

#### Restart

 $M \parallel F \rightarrow \emptyset \parallel F$  if ... you want to



$$M \parallel F \rightarrow M \parallel F, C \quad \text{if } \begin{cases} \text{all atoms of } C \text{ occur in } F, \\ F \vdash C \end{cases}$$

#### Forget

$$M \parallel F, C \quad \to \quad M \parallel F \quad \text{if } F \vdash C$$

#### Restart

 $M \parallel F \rightarrow \emptyset \parallel F$  if ... you want to

#### The DPLL system =

{UnitProp, Decide, Fail, Backjump, Learn, Forget, Restart}



6 Applying one Basic DPLL rule between each two Learn and applying Restart less and less often ensures termination.

- 6 Applying one Basic DPLL rule between each two Learn and applying Restart less and less often ensures termination.
- In practice, Learn is usually (but not only) applied right after Backjump.

- 6 Applying one Basic DPLL rule between each two Learn and applying Restart less and less often ensures termination.
- In practice, Learn is usually (but not only) applied right after Backjump.
- 6 A common strategy is to apply the rules with these priorities:

- 6 Applying one Basic DPLL rule between each two Learn and applying Restart less and less often ensures termination.
- In practice, Learn is usually (but not only) applied right after Backjump.
- 6 A common strategy is to apply the rules with these priorities:
  - 1. If n > 0 conflicts have been found so far, increase n and apply Restart.

- 6 Applying one Basic DPLL rule between each two Learn and applying Restart less and less often ensures termination.
- 6 In practice, Learn is usually (but not only) applied right after Backjump.
- 6 A common strategy is to apply the rules with these priorities:
  - 1. If n > 0 conflicts have been found so far, increase n and apply Restart.
  - If a current clause is falsified by the current assignment, apply Fail Or Backjump + Learn.

- 6 Applying one Basic DPLL rule between each two Learn and applying Restart less and less often ensures termination.
- 6 In practice, Learn is usually (but not only) applied right after Backjump.
- 6 A common strategy is to apply the rules with these priorities:
  - 1. If n > 0 conflicts have been found so far, increase n and apply Restart.
  - If a current clause is falsified by the current assignment, apply Fail Or Backjump + Learn.
  - 3. Apply UnitProp

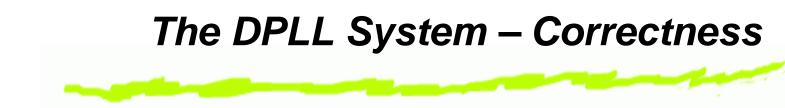


Proposition (Termination) Every execution in which(a) Learn/Forget are applied only finitely many times and(b) Restart is applied with increased periodicityis finite.



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

**Proposition** (Soundness) For every execution  $\emptyset \parallel F \Longrightarrow \cdots \Longrightarrow M \parallel F$  with  $M \parallel F$  irreducible wrt. Basic DPLL,  $M \models F$ .



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

**Proposition** (Soundness) For every execution  $\emptyset \parallel F \Longrightarrow \cdots \Longrightarrow M \parallel F$  with  $M \parallel F$  irreducible wrt. Basic DPLL,  $M \models F$ .

**Proposition** (Completeness) If *F* is unsatisfiable, for every execution  $\emptyset \parallel F \Longrightarrow \cdots \Longrightarrow S$  with *S* irreducible wrt. Basic DPLL, S = fail.

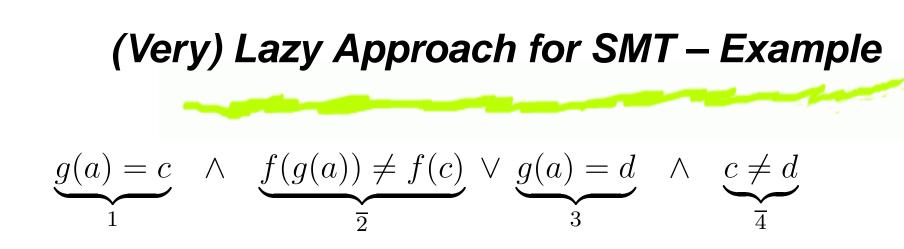


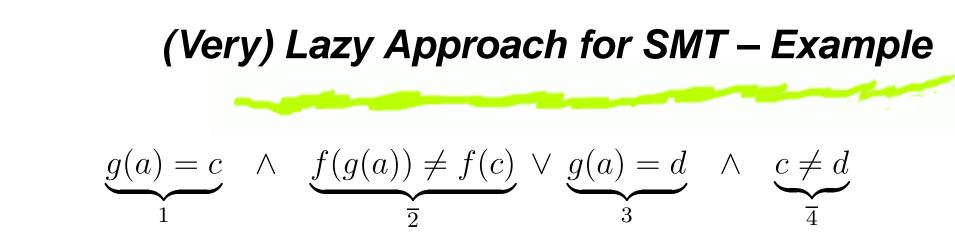
- 6 Motivation: SAT and SMT
- 6 The DPLL procedure
- 6 An Abstract Framework
- SAT case
  - △ The Original DPLL Procedure
  - The Basic and the Enhanced DPLL System
- 6 SMT case
  - Very Lazy Theory Learning
  - Lazy Theory Learning
  - Theory Propagation

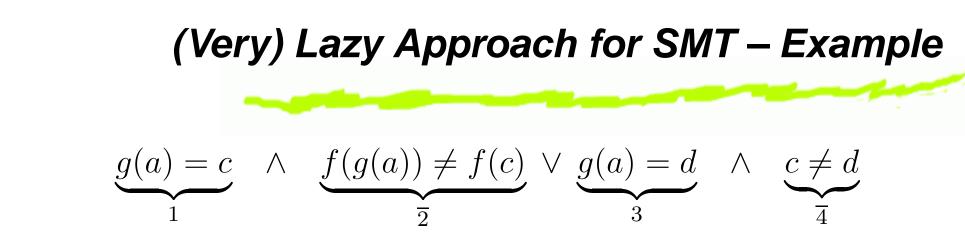


#### $g(a) = c \quad \land \quad f(g(a)) \neq f(c) \lor g(a) = d \quad \land \quad c \neq d$

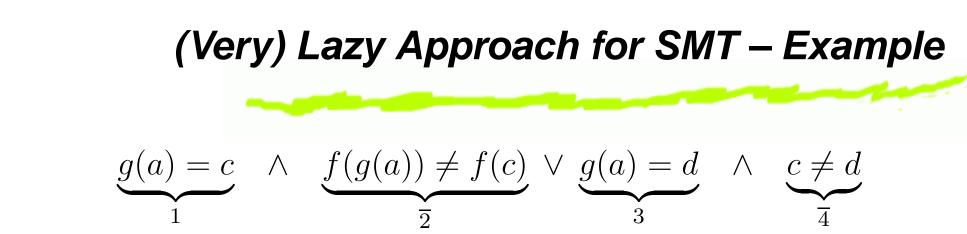
#### **Theory: Equality**



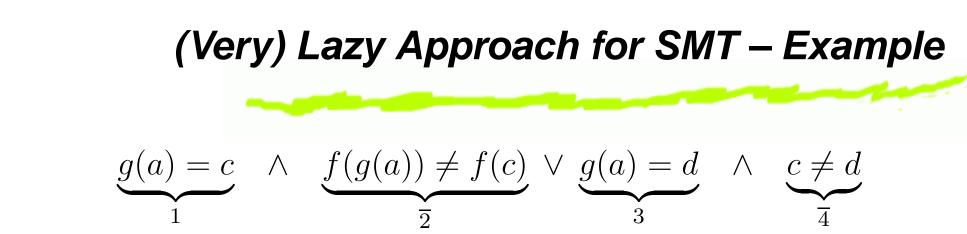




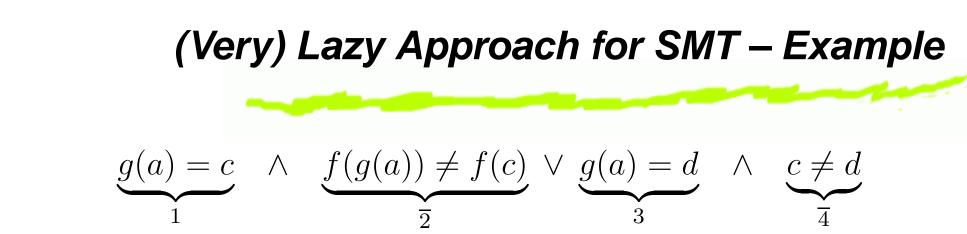
- Send  $\{1, \overline{2} \lor 3, \overline{4}\}$  to SAT solver.
- SAT solver returns model  $\{1, \overline{2}, \overline{4}\}$ . Theory solver finds  $\{1, \overline{2}\}$  *E*-unsatisfiable.



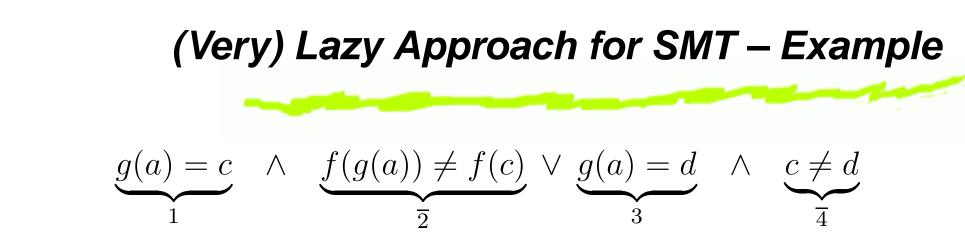
- SAT solver returns model  $\{1, \overline{2}, \overline{4}\}$ . Theory solver finds  $\{1, \overline{2}\}$  *E*-unsatisfiable.
- Send  $\{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2\}$  to SAT solver.



- SAT solver returns model  $\{1, \overline{2}, \overline{4}\}$ . Theory solver finds  $\{1, \overline{2}\}$  *E*-unsatisfiable.
- Send  $\{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2\}$  to SAT solver.
- SAT solver returns model  $\{1, 2, 3, \overline{4}\}$ . Theory solver finds  $\{1, 3, \overline{4}\}$  *E*-unsatisfiable.



- SAT solver returns model  $\{1, \overline{2}, \overline{4}\}$ . Theory solver finds  $\{1, \overline{2}\}$  *E*-unsatisfiable.
- Send  $\{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2\}$  to SAT solver.
- 6 SAT solver returns model  $\{1, 2, 3, \overline{4}\}$ . Theory solver finds  $\{1, 3, \overline{4}\}$  *E*-unsatisfiable.
- Send  $\{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2, \overline{1} \lor \overline{3} \lor 4\}$  to SAT solver.



- SAT solver returns model  $\{1, \overline{2}, \overline{4}\}$ . Theory solver finds  $\{1, \overline{2}\}$  *E*-unsatisfiable.
- Send  $\{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2\}$  to SAT solver.
- 6 SAT solver returns model  $\{1, 2, 3, \overline{4}\}$ . Theory solver finds  $\{1, 3, \overline{4}\}$  *E*-unsatisfiable.
- Send  $\{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2, \overline{1} \lor \overline{3} \lor 4\}$  to SAT solver.
- 6 SAT solver finds  $\{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2, \overline{1} \lor \overline{3} \lor 4\}$  unsatisfiable.



Let T be the background theory.

The previous process can be modeled in Abstract DPLL using the following rules:

- UnitProp, Decide, Fail, Restart
   (as in the propositional case) and
- 6 T-Backjump, T-Learn, T-Forget, Very Lazy Theory Learning

**Note:** The first component of a state  $M \parallel F$  is still a truth assignment, but now for ground, first-order literals.

#### Modeling the Lazy Approach

 $T\text{-}\mathsf{Backjump}$ 

$$M l^{\mathsf{d}} N \parallel F, C \longrightarrow M k \parallel F, C \quad \mathbf{if} \prec$$

 $\begin{cases} 1. \ M \ l^{\mathsf{d}} \ N \models \neg C, \\ 2. \ \text{for some clause } D \lor k: \\ F, C \vdash_T D \lor k, \\ M \models \neg D, \\ k \ \text{is undefined in } M, \\ k \ \text{or } \overline{k} \ \text{occurs in} \\ M \ l^{\mathsf{d}} \ N \parallel F, CF \ \text{or } M \ l^{\mathsf{d}} \ I \end{cases}$ 

#### **Only change:** $\vdash_T$ instead of $\vdash$

 $F \vdash_T G$  iff every model of T that satisfies F satisfies G.

#### Modeling the Lazy Approach

 $\begin{pmatrix} 1 & M \end{pmatrix}^{\mathsf{d}} N \models \neg C$ 

 $T\text{-}\mathsf{Backjump}$ 

$$M l^{\mathsf{d}} N \parallel F, C \rightarrow M k \parallel F, C \text{ if } \begin{cases} 1. M l^{\mathsf{d}} N \parallel F, C \\ 2. \text{ for some clause } D \lor k; \\ F, C \vdash_T D \lor k, \\ M \models \neg D, \\ k \text{ is undefined in } M, \\ k \text{ or } \overline{k} \text{ occurs in} \\ M l^{\mathsf{d}} N \parallel F, CF \text{ or } M l^{\mathsf{d}} N \end{cases}$$

#### T-Learn

 $M \parallel F \rightarrow M \parallel F, C \text{ if } \begin{cases} \text{all atoms of } C \text{ occur in } F, \\ F \vdash_T C \end{cases}$ 

T-Forget

 $M \parallel F, C \rightarrow M \parallel F \text{ if } F \vdash_T C$ 



The interaction between theory solver and SAT solver in the previous example can be modeled with the rule

Very Lazy Theory Learning

$$M \parallel F \rightarrow \emptyset \parallel F, \overline{l_1} \vee \ldots \vee \overline{l_n} \quad \text{if } \begin{cases} M \models F \\ \{l_1, \ldots, l_n\} \subseteq M \\ l_1 \wedge \cdots \wedge l_n \vdash_T \bot \end{cases}$$



The interaction between theory solver and SAT solver in the previous example can be modeled with the rule

Very Lazy Theory Learning

$$M \parallel F \rightarrow \emptyset \parallel F, \overline{l_1} \vee \ldots \vee \overline{l_n} \quad \text{if} \quad \begin{cases} M \models F \\ \{l_1, \ldots, l_n\} \subseteq M \\ l_1 \wedge \cdots \wedge l_n \vdash_T \bot \end{cases}$$

A better approach is to detect partial assignments that are already T-unsatisfiable.

#### Modeling the Lazy Approach

Lazy Theory Learning

$$M \parallel F \rightarrow M \parallel F, \overline{l_1} \vee \ldots \vee \overline{l_n} \quad \text{if } \begin{cases} \{l_1, \ldots, l_n\} \subseteq M \\ l_1 \wedge \cdots \wedge l_n \vdash_T \bot \\ \overline{l_1} \vee \cdots \vee \overline{l_n} \notin F \end{cases}$$

#### Modeling the Lazy Approach

Lazy Theory Learning

$$M \parallel F \rightarrow M \parallel F, \overline{l_1} \vee \ldots \vee \overline{l_n} \quad \text{if } \begin{cases} \{l_1, \ldots, l_n\} \subseteq M \\ l_1 \wedge \cdots \wedge l_n \vdash_T \bot \\ \overline{l_1} \vee \cdots \vee \overline{l_n} \notin F \end{cases}$$

- 6 The learned clause is false in M, hence either Backjump or Fail applies.
- If this is always done, the third condition of the rule is unnecessary
- In some solvers, the rule is applied as soon as possible, i.e., with  $M = N l_n$ .



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

- 1. If a current clause is falsified by the current assignment, apply Fail/Backjump + Learn.
- 2. If the assignment is *T*-unsatisfiable, apply Lazy Theory Learning + (Fail/Backjump).
- 3. Apply UnitProp.
- 4. Apply Decide.



- 6 Motivation: SAT and SMT
- 6 The DPLL procedure
- 6 An Abstract Framework
- SAT case
  - △ The Original DPLL Procedure
  - The Basic and the Enhanced DPLL System
- 6 SMT case
  - Very Lazy Theory Learning
  - Lazy Theory Learning
  - Theory Propagation

DPLL(T) – Eager Theory Propagation

Use the theory information as soon as possible by eagerly applying

Theory Propagate  $M \parallel F \rightarrow M \ l \parallel F \quad \text{if} \begin{cases} M \vdash_T l \\ l \text{ or } \overline{l} \text{ occurs in } F \\ l \text{ is undefined in } M \end{cases}$ 

# $\underbrace{ \begin{array}{c} \textbf{Eager Theory Propagation - Example} \\ \underbrace{g(a) = c}_{1} & \land & \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \lor \underbrace{g(a) = d}_{3} & \land & \underbrace{c \neq d}_{\overline{4}} \end{array} }$

 $\emptyset \parallel 1, \ \overline{2} \lor 3, \ \overline{4}$ 

# $\begin{array}{c} \textbf{Eager Theory Propagation - Example}\\\\ \underbrace{g(a) = c}_{1} \land \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \lor \underbrace{g(a) = d}_{3} \land \underbrace{c \neq d}_{\overline{4}} \\\\ \emptyset \parallel 1, \overline{2} \lor 3, \overline{4} \implies (\text{UnitProp})\\ 1 \parallel 1, \overline{2} \lor 3, \overline{4} \end{array}$

# **Eager Theory Propagation - Example** $\underbrace{g(a) = c}_{1} \land \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \lor \underbrace{g(a) = d}_{3} \land \underbrace{c \neq d}_{\overline{4}}$ $1 2 \parallel 1, \overline{2} \lor 3, \overline{4}$

# **Eager Theory Propagation - Example** $\underbrace{g(a) = c}_{1} \land \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \lor \underbrace{g(a) = d}_{3} \land \underbrace{c \neq d}_{\overline{4}}$

 $1\ 2\ 3\ \|\ 1,\ \overline{2}\lor 3,\ \overline{4}$ 

#### Eager Theory Propagation - Example

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

 $\emptyset \parallel 1, \overline{2} \lor 3, \overline{4}$  $\parallel 1, \overline{2} \lor 3, \overline{4}$ 1  $1 \ 2 \parallel 1, \ \overline{2} \lor 3, \ \overline{4} \implies$  $1\ 2\ 3\ \|\ 1,\ \overline{2}\lor 3,\ \overline{4}$  $1 2 3 4 \parallel 1, \overline{2} \lor 3, \overline{4}$ 

 $\Longrightarrow$  $\implies$  $\Longrightarrow$ 

(UnitProp)

- (Theory Propagate)
- (UnitProp)
- (Theory Propagate)

#### **Eager Theory Propagation - Example**

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

 $\emptyset \parallel 1, \ \overline{2} \lor 3, \ \overline{4} \implies$  $1 \parallel 1, \overline{2} \lor 3, \overline{4} \implies$  $1 \ 2 \parallel 1, \ \overline{2} \lor 3, \ \overline{4} \implies (UnitProp)$  $1\ 2\ 3\ \|\ 1,\ \overline{2}\lor 3,\ \overline{4} \implies$  $1 \ 2 \ 3 \ 4 \ \| \ 1, \ \overline{2} \lor 3, \ \overline{4}$  $\Longrightarrow$ fail

(UnitProp) (Theory Propagate)

- (Theory Propagate)

(Fail)



- 6 By eagerly applying Theory Propagate every assignment is *T*-satisfiable, since M l is *T*-unsatisfiable iff  $M \vdash_T \overline{l}$ .
- 6 As a consequence, Lazy Theory Learning never applies.
- 6 For some logics, e.g., difference logic, his approach is extremely effective.
- 6 For some others, e.g., the theory of equality, it is too expensive to detect all *T*-consequences.
- If Theory Propagate is not applied eagerly, Lazy Theory Learning is needed to repair T-unsatisfiable assignments.



- 6 The six rules of the DPLL system plus Theory Propagate and Lazy Theory Learning provide a decision procedure for SMT.
- **5** Termination can be guaranteed this way:
  - 1. Apply at least one Basic DPLL rule between any two consecutive Learn applications.
  - 2. Apply Fail/Backjump immediately after Lazy Theory Learning.
- Soundness and completeness are proved similarly to the propositional case.



- 6 The DPLL procedure can be modelled abstractly by a transition system.
- Modern features such as backjumping, learning and restarts can be captured with our transition systems.
- 6 Extensions to SMT are simple and clean.
- 6 We can reason formally about the termination and correctness of DPLL variants for SAT/SMT.
- 6 We can compare different systems at a higher level.
- 6 We got new insights for further enhancements of DPLL solvers for SMT. (Stay tuned.)



#### Thank you

