Solving SAT Modulo Theories


Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)

Mooly Sagiv
Motivation

• We have seen that efficient SAT solvers exit
  – DPLL is the most successful complete solver

• Can we generalize the results?
  – Is “p ∨ ¬q ∨ (a = f(b – c)) ∨ (g(g(b)) ≠ c) ∨ a-c≤7” satisfiable?

• Improve our understanding of DPLL
Ground First Order Formulas

- Constants
- Functions
- Predicates
- Propositional Formulas ¬, ∨, ∧,
Satisfiability Modulo Theories

• Any SAT solver can be used to decide the satisfiability of ground 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
    • \( f(g(a, b), c) = g(c, a) \)
  – Timed automata: planning: theory of integers/reals,
  – Atoms
    • \( x - y < 2 \)
  – Software verification: combination of theories, atoms
    • \( 5 + \text{car}(a + 2) = \text{cdr}(a[j] + 1) \)

• We refer to this general problems as (ground) Satisfiability Modulo Theories, or SMT
Satisfiability Modulo a Theory $\mathcal{T}$

- **Note:** The $\mathcal{T}$-satisfiability of ground formulas is decidable iff the $\mathcal{T}$-satisfiability of sets of literals is decidable.

- **Fact:** Many theories of interest have (efficient) decision procedures for sets of literals.

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

- **Current solution:** Exploit propositional satisfiability technology.
Example Difference Constraints

• Boolean combinations of `a ≤ b + k'
  – a and b are free constants
  – k ∈ Z
Motivating Example

Skolem-Lowenheim Formulas

- Prenex Normal Form $\exists \forall$
- $\exists x, y \, \forall z, w : P(x, y) \land \neg P(z, w)$
Lifting SAT to SMT

• 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

• DPLL(T) [DPLL T, Sammy]:
  – use the decision procedure to guide the search of a DPLL solver
Goals of the article

• Develop a declarative formal framework to:
  – Reason formally about DPLL-based solvers for SAT and for SMT
  – Model modern features such as non-chronological backtracking lemma learning or restarts
  – Describe different strategies and prove their correctness
  – Compare different systems at a higher level
  – Get new insights for further enhancements of DPPL solvers
Outline

✓ Motivation
• Abstract DPLL
• Abstract DPLL modulo theories
• DPLL(T)
• [Experiments]
The original DPLL procedure

- Tries to **build** incrementally a satisfying truth assignment $M$ for a CNF formula $F$

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

- If a wrong guess for a literal leads to an inconsistency, the procedure **backtracks** and tries the opposite value
**The Original DPLL Procedure – Example**

<table>
<thead>
<tr>
<th>assign</th>
<th>1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2, ¬1 ∨ ¬3 ∨ ¬4, 1</th>
</tr>
</thead>
</table>

Deduce 1

<table>
<thead>
<tr>
<th>1</th>
<th>1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2, ¬1 ∨ ¬3 ∨ ¬4, 1</th>
</tr>
</thead>
</table>

Deduce ¬2

<table>
<thead>
<tr>
<th>1, 2</th>
<th>1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2, ¬1 ∨ ¬3 ∨ ¬4, 1</th>
</tr>
</thead>
</table>

Guess 3

<table>
<thead>
<tr>
<th>1, 2, 3</th>
<th>1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2, ¬1 ∨ ¬3 ∨ ¬4, 1</th>
</tr>
</thead>
</table>

Deduce 4

<table>
<thead>
<tr>
<th>1, 2, 3, 4</th>
<th>1 ∨ 2, 2 ∨ ¬3 ∨ 4, ¬1 ∨ ¬2, ¬1 ∨ ¬3 ∨ ¬4, 1</th>
</tr>
</thead>
</table>

Conflict
# The Original DPLL Procedure – Example

<table>
<thead>
<tr>
<th>assign</th>
<th>( 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor 3 \lor \neg 4, 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deduce 1</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>( 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor 3 \lor \neg 4, 1 )</td>
</tr>
<tr>
<td><strong>Deduce \neg 2</strong></td>
<td></td>
</tr>
<tr>
<td>1, 2</td>
<td>( 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor 3 \lor \neg 4, 1 )</td>
</tr>
<tr>
<td><strong>Guess 3</strong></td>
<td></td>
</tr>
<tr>
<td>1, 2, 3</td>
<td>( 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor 3 \lor \neg 4, 1 )</td>
</tr>
<tr>
<td><strong>Deduce 4</strong></td>
<td></td>
</tr>
<tr>
<td>1, 2, 3, 4</td>
<td>( 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor 3 \lor \neg 4, 1 )</td>
</tr>
<tr>
<td><strong>Undo 3</strong></td>
<td></td>
</tr>
</tbody>
</table>
## The Original DPLL Procedure – Example

<table>
<thead>
<tr>
<th>assign</th>
<th>$1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</th>
</tr>
</thead>
</table>

**Deduce 1**

<table>
<thead>
<tr>
<th>1</th>
<th>$1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</th>
</tr>
</thead>
</table>

**Deduce $\neg 2$**

<table>
<thead>
<tr>
<th>1, 2</th>
<th>$1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</th>
</tr>
</thead>
</table>

**Guess $\neg 3$**

<table>
<thead>
<tr>
<th>1, 2, 3</th>
<th>$1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</th>
</tr>
</thead>
</table>

**Model Found**
An Abstract Framework for DPLL

• 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.

• A transition system is a binary relation over states, induced by a set of conditional transition rules.
An Abstract Framework for DPLL

• State
  – 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
An Abstract Framework for DPLL

• State
  – fail or M || F
  – where
    • F is a CNF formula, a set of clauses, and
    • M is a sequence of annotated literals denoting a partial truth assignment

• Initial State
  – ∅ || F, where F is to be checked for satisfiability

• Expected final states:
  – fail if F is unsatisfiable
  – M || G
    where
    • M is a model of G
    • G is logically equivalent to F
Transition Rules for the Original DPLL

• Extending the assignment:

\[ M \parallel F, C \lor l \rightarrow M \parallel l, F, C \lor l \]

UnitProp \[ M \parallel F, C \lor l \rightarrow M \parallel F, C \lor l \]

\[ M \models \neg C \]

\[ l \text{ is undefined in } M \]

Decide \[ M \parallel F, C \rightarrow M \parallel l^d, F, C \]

\[ l \text{ or } \neg l \text{ occur in } C \]

\[ l \text{ is undefined in } M \]

Notation: \( l^d \) is a decision literal
Transition Rules for the Original DPLL

- Repairing the assignment:

  **Fail**
  \[ M \parallel F, C \lor l \rightarrow \text{fail} \]
  
  \[ M \parallel \neg C \]
  
  \[ M \text{ does not contain decision literals} \]

  **Backtrack**
  \[ M \mid^d N \parallel F, C_l \rightarrow M \neg l \parallel F, C \]
  
  \[ M \mid^d N \models \neg C \]
  
  \[ l \text{ is the last decision literal} \]
Transition Rules DPLL – Example

\[ \emptyset \lor 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1 \]

UnitProp 1

\[ 1 \lor 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1 \]

UnitProp \neg 2

\[ 1, 2 \lor 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1 \]

Decide 3

\[ 1, 2, 3^d \lor 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1 \]

UnitProp 4

\[ 1, 2, 3^d, 4 \lor 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1 \]

Backtrack 3
Transition Rules DPLL – Example

<table>
<thead>
<tr>
<th>Initial Clauses</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset \lor 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</td>
<td>UnitProp 1</td>
</tr>
<tr>
<td>$1 \lor 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</td>
<td>UnitProp $\neg 2$</td>
</tr>
<tr>
<td>$1, 2 \lor 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</td>
<td>Decide 3</td>
</tr>
<tr>
<td>$1, 2, 3 \lor 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</td>
<td>UnitProp 4</td>
</tr>
<tr>
<td>$1, 2, 3, 4 \lor 1 \lor 2, 2 \lor \neg 3 \lor 4, \neg 1 \lor \neg 2, \neg 1 \lor \neg 3 \lor \neg 4, 1$</td>
<td>Backtrack 3</td>
</tr>
</tbody>
</table>
# Transition Rules for the Original DPLL

<table>
<thead>
<tr>
<th>Rule</th>
<th>Transition</th>
<th>Conditions</th>
</tr>
</thead>
</table>
| UnitProp        | $M \parallel F, C \lor l \rightarrow M \parallel F, C \lor l$             | $M \models \neg C$  
|                 |                                                                            | $l$ is undefined in $M$                                                   |
| Decide          | $M \parallel F, C \rightarrow M \lnot l \parallel F, C$                  | $l$ or $\neg l$ occur in $C$  
|                 |                                                                            | $l$ is undefined in $M$                                                   |
| Fail            | $M \parallel F, C \lor l \rightarrow \text{fail}$                       | $M \models \neg C$  
|                 |                                                                            | $M$ does not contain decision literals                                    |
| Backtrack       | $M \lnot l \parallel N \parallel F, Cl \rightarrow M \lnot l \parallel F, C$ | $M \models \neg C$  
|                 |                                                                            | $M \lnot l \parallel N \models \neg C$  
|                 |                                                                            | $l$ is the last decision literal                                          |
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

• 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

• Maintained in more general rules + theories
From Backtracking to Backjumping

Backtrack

\[ M \models^d N \parallel F, C \rightarrow M \models \neg l \parallel F, C \]

- \( M \models^d N \models \neg C \)
- \( l \) is the last decision literal

Backjump

\[ M \models^d N \parallel F, C \rightarrow M \models^d k \parallel F, C \]

- \( \bar{M} \models^d N \models \neg C \)
- For some clause \( D \lor k \):
  - \( F, C \models D \lor k \):
  - \( M \models \neg D \)
  - \( k \) is undefined in \( M \)
  - \( k \) or \( \neg k \) occurs in \( M \models^d N \parallel F, C \)
Enhancements to Basic DPLL

Learn  \[ M \parallel F \rightarrow M \parallel F, C \]

Forget  \[ M \parallel F, C \rightarrow M \parallel F \]

all the atoms in C occur in F
\[ F \not\models C \]

Usually C is identified during conflict analysis.
Enhancements to Basic DPLL

Learn \[ M \parallel F \rightarrow M \parallel F, C \]

Forget \[ M \parallel F, C \rightarrow M \parallel F \]

Restart \[ M \parallel F, C \rightarrow \emptyset \parallel F \]

The DPLL system = \{UnitProp, Decide, Fail, Backjump, Learn, Forget, Restart\}
The DPLL System – Strategies

- 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.
- 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
  2) If a current clause is falsified by the current assignment, apply Fail or Backjump + Learn
  3) Apply UnitProp
**The DPLL System – Correctness**

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

  is finite

- **Proposition (Soundness)** For every execution
  - $\emptyset \parallel F \rightarrow^* M \parallel G$ with $M \parallel G$ irreducible wrt. Basic DPLL, $M \models F$

- **Proposition (Completeness)** If $F$ is unsatisfiable, for every execution $\emptyset \parallel F \rightarrow^* S$ with $S$ irreducible wrt. Basic DPLL
  - $S = \text{fail}$
(Very) Lazy Approach for SMT – Example

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

Theory of uninterpreted functions
(Very) Lazy Approach for SMT – Example

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

Send \{1, \neg 2 \lor 3, \neg 4\} to the SAT solver

SAT solver returns \{1, \neg 2, \neg 4\}

Theory solver finds that \{1, \neg 2\} is E-unsatisfiable

Send \{1, \neg 2 \lor 3, \neg 4, \neg 1 \lor 2\} to the SAT solver

SAT solver returns \{1, 2, 3, \neg 4\}

Theory solver finds that \{1, 3, \neg 4\} is E-unsatisfiable

Send \{1, \neg 2 \lor 3, \neg 4, \neg 1 \lor 2, \neg 1 \lor \neg 3 \lor 4\} to the SAT solver

Return UNSAT
Modeling the lazy approach

• 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
  – 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-Backjump

\[ M \models^d N \parallel F, C \rightarrow M \models_k \parallel F, C \]

\[ M \models^d N \models \neg C \]

For some clause \( D \lor k \):

\[ F, C \models_T D \lor k : \]

\[ M \models \neg D \]

\( k \) is undefined in \( M \)

\( k \) or \( \neg k \) occurs in \( M \models^d N \parallel F, C \)

\[ F \models_T G \text{ iff every model of } T \text{ that satisfies } F \text{ satisfies } G \]
Modeling the Lazy Approach

T-Backjump: $M \models^d N \parallel F, C \rightarrow M \models F, C$

Learn: $M \models F \rightarrow M \models F, C$

Forget: $M \models F, C \rightarrow M \models F$

For some clause $D \lor k$:

$M \models \neg C$

$m \models D \land k$:

$F, C \models T D \land k$:

$M \models \neg D$

$k$ is undefined in $M$

If $k$ or $\neg k$ occurs in $M \models N \parallel F, C$

all the atoms in $C$ occur in $F$

$F \models T C$

all the atoms in $C$ occur in $F$

$F \models T C$
Modeling the Lazy Approach

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

Very Lazy Theory Learning

\[
M \parallel F \rightarrow \emptyset \parallel F, \lnot l_1 \lor \lnot l_2 \ldots \lor \lnot l_n
\]

\[
\begin{align*}
M & \models F \\
\{l_1, l_2, \ldots, l_n\} & \subseteq M \\
\{l_1 \land l_2 \land \ldots \land l_n\} & \not\models T \bot
\end{align*}
\]

A better approach is to detect partial assignments that already T-unsatisfiable.
Modeling the Lazy Approach

\[ M \| F \rightarrow \emptyset \| F, \neg l_1 \lor \neg l_2 \ldots \lor \neg l_n \]

\begin{align*}
\begin{cases}
\neg l_1 \lor \neg l_2 \ldots \lor \neg l_n \notin F \\
\{l_1, l_2, \ldots, l_n\} \subseteq M \\
l_1 \land l_2 \land \ldots \land l_n \models \bot
\end{cases}
\end{align*}

• The learned clause is false in \( M \), hence either Backjump or Fail applies
• If this is always done, the first condition of the rule is unnecessary
• In some solvers, the rule is applied as soon as possible, i.e., with \( M = N l_n \)
Lazy Approach – Strategies

• 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
DPLL(\ T) – Eager Theory Propagation

- Use the theory information as soon as possible by eagerly applying Theory Propagate

\[ M \parallel F \models M I \parallel F \]

- \( M \models_{T} I \)
- \( I \) or \( \neg I \) occur in \( F \)
- \( I \) is undefined in \( M \)
Eager Theory Propagation - Example

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

1 \quad -2 \quad 3 \quad -4

\[\emptyset \parallel \{1, -2 \lor 3, -4\}\]

UnitPropagate 1

1 \parallel \{1, -2 \lor 3, -4\}

TheoryPropagate 2

1, 2 \parallel \{1, -2 \lor 3, -4\}

UnitPropagate 3

1, 2, 3 \parallel \{1, -2 \lor 3, -4\}

TheoryPropagate 4

1, 2, 3, 4 \parallel \{1, -2 \lor 3, -4\}

Fail
Eager Theory Propagation

- By eagerly applying Theory Propagate every assignment is T-satisfiable, since $M \vDash T \iff M \not\vDash T$.

- As a consequence, Lazy Theory Learning never applies.

- For some logics, e.g., difference logic, this approach is extremely effective.

- For some others, e.g., the theory of equality of uninterpreted functions, 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.
Non-Exhaustive Theory Propagation

• The six rules of the DPLL system plus Theory Propagate and Lazy Theory Learning provide a decision procedure for SMT

• 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
History

- The original DP algorithm was developed for first order logic
Conclusions (C. Tinelli)

• 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

• Extensions to SMT are simple and clean

• We can reason formally about the termination and correctness of DPLL variants for SAT/SMT

• We can compare different systems at a higher level

• We got new insights for further enhancements of DPLL solvers for SMT