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

Propositional Satisfiability: SAT

- Deciding the satisfiability of a propositional formula is a well-studied and important problem.
- Theoretical interest: first established NP-Complete problem, phase transition, . . .
- Practical interest: applications to scheduling, planning, logic synthesis, verification, . . .
  - Development of algorithms and enhancements.
  - Implementation of extremely efficient tools.
  - Solvers based on the DPLL procedure have been the most successful so far.

Satisfiability Modulo Theories

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

Satisfiability Modulo Theories

- 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:

Satisfiability Modulo Theories

- 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 $f(g(a,b),c) = g(c,a)$.

Satisfiability Modulo Theories

- 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 $x - y < 2$.

Satisfiability Modulo Theories

- 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 $5 + car(a + 2) = cdr(a[j] + 1)$.

Satisfiability Modulo Theories

- 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 $5 + car(a + 2) = cdr(a[j] + 1)$.
- We refer to this general problems as (ground) Satisfiability Modulo Theories, or SMT.

Satisfiability Modulo a Theory T

- 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.
- Problem: In practice, dealing with Boolean combinations of literals is as hard as in the propositional case.
- Current solution: Exploit propositional satisfiability technology.

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) [DPLLT, Sammy]:
  - use the decision procedure to guide the search of a DPLL solver.

Goals of This Work

Develop a declarative formal framework to:

- Reason formally about DPLL-based solvers for SAT and for SMT.
- Model modern features such as non-chronological bactracking, 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.

Talk Overview

- Motivation: SAT and SMT
- The DPLL procedure
- 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

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

Operation	Assign.	Formula
		$1 \vee 2,\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee \overline{2},\ \overline{1} \vee \overline{3} \vee \overline{4},\ 1$

The Original DPLL Procedure – Example

Operation	Assign.	Formula
		$1 \vee 2,\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee \overline{2},\ \overline{1} \vee \overline{3} \vee \overline{4},\ 1$
deduce 1	1	$1 \vee 2,\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee \overline{2},\ \overline{1} \vee \overline{3} \vee \overline{4},\ 1$

The Original DPLL Procedure – Example

Operation	Assign.	Formula
		$1 \vee 2,\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee \overline{2},\ \overline{1} \vee \overline{3} \vee \overline{4},\ 1$
deduce 1	1	$1 \vee 2,\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee \overline{2},\ \overline{1} \vee \overline{3} \vee \overline{4},\ 1$
deduce 2	$1,\ 2$	$1 \vee 2,\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee \overline{2},\ \overline{1} \vee \overline{3} \vee \overline{4},\ 1$

The Original DPLL Procedure – Example

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

The Original DPLL Procedure – Example

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

The Original DPLL Procedure – Example

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

Inconsistency!

The Original DPLL Procedure – Example

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

The Original DPLL Procedure – Example

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

The Original DPLL Procedure – Example

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

Model Found!

Talk Overview

- Motivation: SAT and SMT
- The DPLL procedure
- 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

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

Our states:

$$fail \quad \text{or} \quad M \parallel F$$

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

Our states:

$$fail \quad \text{or} \quad M \parallel F$$

Initial state:

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

Expected final states:

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

Transition Rules for the Original DPLL

Extending the assignment:

UnitProp

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

Transition Rules for the Original DPLL

Extending the assignment:

UnitProp

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

Decide

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

Notation: l^{d} annotates l as a decision literal.

Transition Rules for the Original DPLL

Repairing the assignment:

Fail

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

Transition Rules for the Original DPLL

Repairing the assignment:

Fail

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

Backtrack

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

Talk Overview

- Motivation: SAT and SMT
- The DPLL procedure
- 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

From Backtracking to Backjumping

Backtrack

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

From Backtracking to Backjumping

Backtrack

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

Backjump

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

From Backtracking to Backjumping

Backtrack

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

Backjump

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

Note: Condition (1) is actually not necessary.

Basic DPLL System

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

Basic DPLL

- UnitProp
- Decide
- Fail
- 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 $fail$.

Enhancements to Basic DPLL

Enhancements to Basic DPLL

Learn

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

Enhancements to Basic DPLL

Learn

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

Forget

$$M \parallel F, C \;\; \longrightarrow \;\; M \parallel F \quad \textbf{if } F \vdash C$$

Enhancements to Basic DPLL

Learn

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

Forget

$$M \parallel F, C \quad \longrightarrow \quad M \parallel F \quad \textbf{if } F \vdash C$$

Usually C is a clause identified during conflict analysis.

Enhancements to Basic DPLL

Learn

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

Forget

$$M \parallel F, C \quad \longrightarrow \quad M \parallel F \quad \textbf{if } F \vdash C$$

Restart

$$M \parallel F \quad \longrightarrow \quad \emptyset \parallel F \quad \textbf{if} \ldots \textit{you want to}$$

Enhancements to Basic DPLL

Learn

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

Forget

$$M \parallel F, C \;\; \longrightarrow \;\; M \parallel F \;\; \textbf{if } F \vdash C$$

Restart

$$M \parallel F \;\; \longrightarrow \;\; \emptyset \parallel F \;\; \textbf{if} \ldots \textit{you want to}$$

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.

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.

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:

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.

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.

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

(a) Learn/Forget are applied only finitely many times and

(b) Restart is applied with increased periodicity

is finite.

The DPLL System – Correctness

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, $M \models F$.

The DPLL System – Correctness

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, $S = \mathit{fail}$.

Talk Overview

- Motivation: SAT and SMT
- The DPLL procedure
- 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

(Very) Lazy Approach for SMT – Example

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

Theory: Equality

(Very) Lazy Approach for SMT – Example

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

(Very) Lazy Approach for SMT – Example

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

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.

(Very) Lazy Approach for SMT – Example

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

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

(Very) Lazy Approach for SMT – Example

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

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

(Very) Lazy Approach for SMT – Example

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

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

(Very) Lazy Approach for SMT – Example

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

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

(Very) Lazy Approach for SMT – Example

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

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

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

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

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

T-Backjump

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

T-Learn

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

T-Forget

$$M \parallel F, C \quad \longrightarrow \quad M \parallel F \quad \textbf{if } F \vdash_T C$$

Modeling the Lazy Approach

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

Very Lazy Theory Learning

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

Modeling the Lazy Approach

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

Very Lazy Theory Learning

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

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

Modeling the Lazy Approach

Lazy Theory Learning

$$M \parallel F \quad \longrightarrow \quad M \parallel F, \overline{l_1} \vee \ldots \vee \overline{l_n} \quad \textbf{if} \begin{cases} \{l_1, \ldots, l_n\} \subseteq M \\ l_1 \wedge \cdots \wedge l_n \vdash_T \bot \\ \bar{l}_1 \vee \cdots \vee \bar{l}_n \notin F \end{cases}$$

Modeling the Lazy Approach

Lazy Theory Learning

$$M \parallel F \quad \longrightarrow \quad M \parallel F, \overline{l_1} \vee \ldots \vee \overline{l_n} \quad \textbf{if} \begin{cases} \{l_1, \ldots, l_n\} \subseteq M \\ l_1 \wedge \cdots \wedge l_n \vdash_T \bot \\ \overline{l}_1 \vee \cdots \vee \overline{l}_n \notin F \end{cases}$$

- 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 $M = N\, l_n$.

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.

Talk Overview

- Motivation: SAT and SMT
- The DPLL procedure
- 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

DPLL(T) – Eager Theory Propagation

Use the theory information as soon as possible by eagerly applying

Theory Propagate

$$M \parallel F \quad \rightarrow \quad M\, l \parallel F \quad \textbf{if} \begin{cases} M \vdash_T l \\ l \text{ or } \bar{l} \text{ occurs in } F \\ l \text{ is undefined in } M \end{cases}$$

Eager Theory Propagation - Example

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

$$\emptyset \;\|\; \quad 1,\; \overline{2} \vee 3,\; \overline{4}$$

Eager Theory Propagation - Example

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

$$\begin{array}{lcl} \emptyset \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow \quad (\text{UnitProp}) \\ 1 \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} & \end{array}$$

Eager Theory Propagation - Example

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

$$\begin{array}{rcl} \emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \quad \Longrightarrow \quad \text{(UnitProp)} \\ 1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \quad \Longrightarrow \quad \text{(Theory Propagate)} \\ 1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \end{array}$$

Eager Theory Propagation - Example

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

$$\begin{array}{rcl} \emptyset \;\|\; 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{UnitProp}) \\ 1 \;\|\; 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{Theory Propagate}) \\ 1\ 2 \;\|\; 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{UnitProp}) \\ 1\ 2\ 3 \;\|\; 1,\ \overline{2} \vee 3,\ \overline{4} & & \end{array}$$

Eager Theory Propagation - Example

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

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow \quad (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow \quad (\text{Theory Propagate}) \\
1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow \quad (\text{UnitProp}) \\
1\ 2\ 3 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow \quad (\text{Theory Propagate}) \\
1\ 2\ 3\ 4 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} &
\end{array}$$

Eager Theory Propagation - Example

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

$$\begin{array}{rcl}
\emptyset \;\|\; 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{UnitProp}) \\
1 \;\|\; 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{Theory Propagate}) \\
1\ 2 \;\|\; 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{UnitProp}) \\
1\ 2\ 3 \;\|\; 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{Theory Propagate}) \\
1\ 2\ 3\ 4 \;\|\; 1,\ \overline{2} \vee 3,\ \overline{4} & \Longrightarrow & (\text{Fail}) \\
\mathit{fail} & &
\end{array}$$

Eager Theory Propagation

- By eagerly applying Theory Propagate every assignment is T-satisfiable, since $M\,l$ is T-unsatisfiable iff $M \vdash_T \bar{l}$.
- As a consequence, Lazy Theory Learning never applies.
- For some logics, e.g., difference logic, his approach is extremely effective.
- 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.

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.

Conclusions

- 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. (Stay tuned.)

Thank you