Satisfiability Modulo Theories

Roberto Sebastiani

DISI, University of Trento, Italy.
roberto.sebastiani@disi.unitn.it
http://www.disi.unitn.it/~rseba

j. w. w.
Gilles Audemard, Clark Barrett, Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, Tommi Junttila, Ziyad Hanna, Zurab Khasidashvili, Arthur Kornilowicz, Alexander Nadel, Silvio Ranise, Peter vanRossum, Sanjit Seshia, Bas Schaafsma, Stephan Schultz, Cristian Stenico, Cesare Tinelli, Silvia Tomasi,...

22nd International Joint Conference on Artificial Intelligence – IJCAI 11
Barcelona, Spain, July 16th 2011.
Outline

1. Motivations and goals
2. Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities
3. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories
4. Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization
5. Conclusions & current research directions
Motivations and goals

1. Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities

2. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories

3. Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization

4. Conclusions & current research directions
Satisfiability Modulo Theories (SMT(\mathcal{T}))

The problem of deciding the satisfiability of (typically quantifier-free) formulas in some decidable first-order theory \( \mathcal{T} \)

- \( \mathcal{T} \) can also be a combination of theories \( \bigcup_i \mathcal{T}_i \).
Some theories of interest (e.g., for formal verification)

- **Equality and Uninterpreted Functions (EUФ)**:
  \(((x = y) \land (y = f(z))) \rightarrow (g(x) = g(f(z)))\)

- **Difference logic (DL)**: \(((x = y) \land (y - z \leq 4)) \rightarrow (x - z \leq 6)\)

- **UTVPI (UTVПI)**: \(((x = y) \land (y - z \leq 4)) \rightarrow (x + z \leq 6)\)

- **Linear arithmetic over the rationals (LA(Q))**: 
  \((T_\delta \rightarrow (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0)) \land (\neg T_\delta \rightarrow (s_1 = s_0))\)

- **Linear arithmetic over the integers (LA(Z))**: 
  \((x := x_l + 2^{16}x_h) \land (x \geq 0) \land (x \leq 2^{16} - 1)\)

- **Arrays (AR)**: \((i = j) \lor \text{read(write}(a, i, e), j) = \text{read}(a, j)\)

- **Bit vectors (BV)**: 
  \(x_{[16]}[15 : 0] = (y_{[16]}[15 : 8] :: z_{[16]}[7 : 0]) \ll w_{[8]}[3 : 0]\)

- **Non-Linear arithmetic over the reals (NL A(Q))**: 
  \(((c = a \cdot b) \land (a_1 = a - 1) \land (b_1 = b + 1)) \rightarrow (c = a_1 \cdot b_1 + 1)\)
Satisfiability Modulo Theories (SMT(\mathcal{T})): Example

\[ \varphi \overset{\text{def}}{=} (d \geq 0) \land (d < 1) \land ((f(d) = f(0)) \rightarrow (\text{read}(\text{write}(V, i, x), i + d) = x + 1)) \]

- involves arithmetical, arrays, and uninterpreted function/predicate symbols, plus Boolean operators

- Is it consistent?
- No:

\[ \varphi \rightarrow \mathcal{L}A(\mathbb{Z}) (d = 0) \]
\[ \varphi \rightarrow \mathcal{EUF} (f(d) = f(0)) \]
\[ \varphi \rightarrow \text{Bool} (\text{read}(\text{write}(V, i, x), i + d) = x + 1) \]
\[ \varphi \rightarrow \mathcal{L}A(\mathbb{Z}) (\text{read}(\text{write}(V, i, x), i) = x + 1) \]
\[ \varphi \rightarrow \mathcal{L}A(\mathbb{Z}) \neg(\text{read}(\text{write}(V, i, x), i) = x) \]
\[ \varphi \rightarrow \mathcal{A}R \bot \]
Satisfiability Modulo Theories (SMT(\mathcal{T})): Example

Example: SMT(\mathcal{LA}(\mathbb{Z}) \cup \mathcal{EUF} \cup \mathcal{AR})

\[ \varphi \overset{\text{def}}{=} (d \geq 0) \land (d < 1) \land \]
\[ ((f(d) = f(0)) \rightarrow (\text{read} (\text{write}(V, i, x), i + d) = x + 1)) \]

- involves arithmetical, arrays, and uninterpreted function/predicate symbols, plus Boolean operators
- Is it consistent?
- No:

\[ \varphi \equiv \left( \begin{array}{c}
\mathcal{LA} (\mathbb{Z}) \\
\mathcal{EUF} \\
\mathcal{Bool} \\
\mathcal{LA} (\mathbb{Z}) \\
\mathcal{LA} (\mathbb{Z}) \\
\mathcal{AR}
\end{array} \right) \]

\[ (d = 0) \\
(f(d) = f(0)) \\
(\text{read} (\text{write}(V, i, x), i + d) = x + 1) \\
(\text{read} (\text{write}(V, i, x), i) = x + 1) \\
(\text{read} (\text{write}(V, i, x), i) = x) \\
\bot \]
Satisfiability Modulo Theories (SMT(\mathcal{T})): Example

\begin{align*}
\varphi \overset{\text{def}}{=} (d \geq 0) \land (d < 1) \land \\
((f(d) = f(0)) \rightarrow (\text{read}(\text{write}(V, i, x), i + d) = x + 1))
\end{align*}

- involves arithmetical, arrays, and uninterpreted function/predicate symbols, plus Boolean operators

- Is it consistent?
- No:

\[
\begin{array}{ll}
\varphi & (d = 0) \\
\mathcal{LA}(\mathbb{Z}) & (f(d) = f(0)) \\
\mathcal{EUF} & (\text{read}(\text{write}(V, i, x), i + d) = x + 1) \\
\text{Bool} & (\text{read}(\text{write}(V, i, x), i) = x + 1) \\
\mathcal{LA}(\mathbb{Z}) & (\text{read}(\text{write}(V, i, x), i) = x) \\
\mathcal{LA}(\mathbb{Z}) & \bot
\end{array}
\]
Motivations and goals

Some Motivating Applications

Interest in SMT triggered by some real-word applications

- Verification of Hybrid & Timed Systems
- Verification of RTL Circuit Designs & of Microcode
- SW Verification
- Planning with Resources
- Temporal reasoning
- Scheduling
- Compiler optimization
- ...

Roberto Sebastiani ()

Satisfiability Modulo Theories

July 16th 2011
Verification of Timed Systems

- **Bounded/inductive model checking of Timed Systems** [6, 35, 56], ...

- **Timed Automata** encoded into $\mathcal{T}$-formulas:
  - **discrete information** (locations, transitions, events) with Boolean vars.
  - **timed information** (clocks, elapsed time) with differences $(t_{3} - x_{3} \leq 2)$, equalities $(x_{4} = x_{3})$ and linear constraints $(t_{8} - x_{8} = t_{2} - x_{2})$ on $\mathbb{Q}$

$\implies$ SMT on $\mathcal{DL}(\mathbb{Q})$ or $\mathcal{LA}(\mathbb{Q})$ required
Motivations and goals

Verification of Hybrid Systems ...

- **Bounded model checking of Hybrid Systems** \[5\], ...
- **Hybrid Automata** encoded into $\mathcal{L}$-formulas:
  - discrete information (locs, trans., events) with Boolean vars.
  - timed information (clocks, elapsed time) with differences $(t_3 - x_3 \leq 2)$, equalities $(x_4 = x_3)$ and linear constraints $(t_8 - x_8 = t_2 - x_2)$ on $\mathbb{Q}$
  - Evolution of Physical Variables (e.g., speed, pressure) with linear $(\omega_4 = 2\omega_3)$ and non-linear constraints $(P_1 V_1 = 4T_1)$ on $\mathbb{Q}$
- **Undecidable** under simple hypotheses!

$\implies$ SMT on $\mathcal{DL}(\mathbb{Q})$, $\mathcal{LA}(\mathbb{Q})$ or $\mathcal{NLA}(\mathbb{Q})$ required
Motivations and goals

Verification of HW circuit designs & microcode

- SAT/SMT-based **Model Checking & Equiv. Checking** of RTL designs, *symbolic simulation* of $\mu$-code [22, 19, 41]
- **Control paths** handled by Boolean reasoning
- **Data paths** information abstracted into theory-specific terms
  - *words* (bit-vectors, integers, $EUF$ vars, ...): $a[31:0]$, $a$
  - *word operations*: ($BV$, $EUF$, $AR$, $LA(\mathbb{Z})$, $NLA(\mathbb{Z})$ operators)
    
    \[ x_{16}[15:0] = (y_{16}[15:8] :: z_{16}[7:0]) << w_{8}[3:0], \]
    
    \[ (a = a_L + 2^{16} a_H), (m_1 = store(m_0, l_0, v_0)), \ldots \]

- Trades heavy Boolean reasoning ($\approx 2^{64}$ factors) with $T$-solving
  
  $\Rightarrow$ SMT on $BV$, $EUF$, $AR$, modulo-$LA(\mathbb{Z})$ [$NLA(\mathbb{Z})$ ] required
Verification of SW systems

\[
\ldots \land \\
(i_0 = 0) \land \\
(acc_0 = init()) \land \\
((i_0 < dim) \rightarrow (acc_1 = acc_0 + read(V, i_0)) \land \\
(i_1 = i_0 + 1))) \land \\
(\neg(i_0 < dim) \rightarrow (acc_1 = acc_0) \land (i_1 = i_0))) \land \\
((i_1 < dim) \rightarrow (acc_2 = acc_1 + read(V, i_1)) \land \\
(i_2 = i_1 + 1))) \land \\
(\neg(i_1 < dim) \rightarrow (acc_2 = acc_1) \land (i_2 = i_1))) \land \\
\ldots 
\]

- Verification of SW code
  - \textit{BMC, K-induction, Check of proof obligations, interpolation-based model checking, symbolic simulation, concolic testing, ...}

\[\Rightarrow\text{ SMT on } BV, EUF, AR, (\text{modulo-}) LA(Z) [\text{\textit{N}} LA(Z) ] \text{ required}\]
Planning with Resources [75]

- SAT-bases planning augmented with numerical constraints
- Straightforward to encode into SMT($\mathcal{L}A(\mathbb{Q})$)

Example (sketch) [75]

\[
\begin{align*}
(\text{Deliver}) & \quad \land \quad \text{goal} \\
(\text{MaxLoad}) & \quad \land \quad \text{load constraint} \\
(\text{MaxFuel}) & \quad \land \quad \text{fuel constraint} \\
(\text{Move} \rightarrow \text{MinFuel}) & \quad \land \quad \text{move requires fuel} \\
(\text{Move} \rightarrow \text{Deliver}) & \quad \land \quad \text{move implies delivery} \\
(\text{GoodTrip} \rightarrow \text{Deliver}) & \quad \land \quad \text{a good trip requires} \\
(\text{GoodTrip} \rightarrow \text{AllLoaded}) & \quad \land \quad \text{a full delivery} \\
(\text{MaxLoad} \rightarrow (\text{load} \leq 30)) & \quad \land \quad \text{load limit} \\
(\text{MaxFuel} \rightarrow (\text{fuel} \leq 15)) & \quad \land \quad \text{fuel limit} \\
(\text{MinFuel} \rightarrow (\text{fuel} \geq 7 + 0.5 \text{load})) & \quad \land \quad \text{fuel constraint} \\
(\text{AllLoaded} \rightarrow (\text{load} = 45)) & \quad \land \quad \text{full delivery}
\end{align*}
\]
(Disjunctive) Temporal Reasoning [73, 2]

- Temporal reasoning problems encoded as disjunctions of difference constraints

\[((x_1 - x_2 \leq 6) \lor (x_3 - x_4 \leq -2)) \land \\
((x_2 - x_3 \leq -2) \lor (x_4 - x_5 \leq 5)) \land \\
((x_2 - x_1 \leq 4) \lor (x_3 - x_7 \leq -6)) \land \\
\ldots\]

- Straightforward to encode into SMT(\(\mathcal{DL}\))
Motivations and goals

**SMT and SMT solvers**

**Common fact about SMT problems from various applications**

SMT requires capabilities for **heavy Boolean reasoning** combined with capabilities for **reasoning in expressive decidable F.O. theories**

- SAT alone not expressive enough
- Standard automated theorem proving inadequate (e.g., arithmetic)
- May involve also numerical computation (e.g., simplex)

**Modern SMT solvers**

- Combine **SAT solvers with decision procedures** (*theory solvers*)
  - Contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)
Motivations and goals

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for heavy Boolean reasoning combined with capabilities for reasoning in expressive decidable F.O. theories

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine SAT solvers with decision procedures (theory solvers)
  - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)
Motivations and goals

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for heavy Boolean reasoning combined with capabilities for reasoning in expressive decidable F.O. theories

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine SAT solvers with decision procedures (theory solvers)
  - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)
Motivations and goals

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for heavy Boolean reasoning combined with capabilities for reasoning in expressive decidable F.O. theories

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine SAT solvers with decision procedures (theory solvers)
  - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)
Motivations and goals

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for heavy Boolean reasoning combined with capabilities for reasoning in expressive decidable F.O. theories

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine SAT solvers with decision procedures (theory solvers)
  - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)
**Motivations and goals**

**Goal of this tutorial**

Provide an overview of standard “lazy” SMT:
- foundations
- SMT-solving techniques
- beyond solving: advanced SMT functionalities
- ongoing research

Does not cover related approaches like:
- Eager SAT encodings
- Rewrite-based approaches

We refer to [70, 10] for an overview and references.
Motivations and goals

Notational remark (1): most/all examples in $\mathcal{L}A(\mathbb{Q})$

For better readability, in most/all the examples of this presentation we will use the theory of linear arithmetic on rational numbers ($\mathcal{L}A(\mathbb{Q})$) because of its intuitive semantics. E.g.:

$$(-A_1 \lor (3x_1 - 2x_2 - 3 \leq 5)) \land (A_2 \lor (-2x_1 + 4x_3 + 2 = 3))$$

Nevertheless, analogous examples can be built with all other theories of interest.
Notational remark (2): “constants” vs. “variables”

- Consider, e.g., the formula:
  \((-A_1 \lor (3x_1 - 2x_2 - 3 \leq 5)) \land (A_2 \lor (-2x_1 + 4x_3 + 2 = 3))\)

- How do we call \(A_1, A_2\)?:
  (a) Boolean/propositional variables?
  (b) uninterpreted 0-ary predicates?

- How do we call \(x_1, x_2, x_3\)?:
  (a) domain variables?
  (b) uninterpreted Skolem constants/0-ary uninterpreted functions?

- Hint:
  (a) typically used in SAT, CSP and OR communities
  (b) typically used in logic & ATP communities

Hereafter we call \(A_1, A_2\) “Boolean/propositional variables” and \(x_1, x_2, x_3\) “domain variables” (logic purists, please forgive me!)
Motivations and goals

Notational remark (2): “constants” vs. “variables”

Consider, e.g., the formula:
\((\neg A_1 \lor (3x_1 - 2x_2 - 3 \leq 5)) \land (A_2 \lor (-2x_1 + 4x_3 + 2 = 3))\)

How do we call \(A_1, A_2\):?
(a) Boolean/propositional \textit{variables}?
(b) uninterpreted \textit{0-ary predicates}?

How do we call \(x_1, x_2, x_3\):?
(a) domain \textit{variables}?
(b) uninterpreted Skolem \textit{constants/0-ary uninterpreted functions}?

Hint:
(a) typically used in SAT, CSP and OR communities
(b) typically used in logic & ATP communities

Hereafter we call \(A_1, A_2\) “Boolean/propositional \textit{variables}” and \(x_1, x_2, x_3\) “domain \textit{variables}” (logic purists, please forgive me!)
Notational remark (2): “constants” vs. “variables”

Consider, e.g., the formula:
\[
(\neg A_1 \lor (3x_1 - 2x_2 - 3 \leq 5)) \land (A_2 \lor (-2x_1 + 4x_3 + 2 = 3))
\]

How do we call \(A_1, A_2\):?
(a) Boolean/propositional variables?
(b) uninterpreted 0-ary predicates?

How do we call \(x_1, x_2, x_3\):?
(a) domain variables?
(b) uninterpreted Skolem constants/0-ary uninterpreted functions?

Hint:
(a) typically used in SAT, CSP and OR communities
(b) typically used in logic & ATP communities

Hereafter we call \(A_1, A_2\) “Boolean/propositional variables” and \(x_1, x_2, x_3\) "domain variables” (logic purists, please forgive me!)
Outline

1 Motivations and goals

2 Background: Modern SAT solving
   • Basics
   • Modern CDCL SAT solving
   • Other SAT functionalities

3 Efficient SMT solving
   • Combining SAT with Theory Solvers
   • Theory Solvers for theories of interest
   • SMT for combinations of theories

4 Beyond Solving: advanced SMT functionalities
   • Proofs and unsatisfiable cores
   • Interpolants
   • All-SMT & Predicate Abstraction
   • SMT with cost optimization

5 Conclusions & current research directions
Outline

1 Motivations and goals

2 Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities

3 Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories

4 Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization

5 Conclusions & current research directions
Basic notation & definitions

- **Boolean formula**
  - $\top, \bot$ are formulas
  - A *propositional atom* $A_1, A_2, A_3, \ldots$ is a formula;
  - if $\varphi_1$ and $\varphi_2$ are formulas, then $\neg \varphi_1, \varphi_1 \land \varphi_2, \varphi_1 \lor \varphi_2, \varphi_1 \rightarrow \varphi_2, \varphi_1 \leftrightarrow \varphi_2$ are formulas.

- **Literal**: a propositional atom $A_i$ (positive literal) or its negation $\neg A_i$ (negative literal)

- Notation: if $l := \neg A_i$, then $\neg l := A_i$

- **Atoms($\varphi$)**: the set $\{A_1, \ldots, A_N\}$ of atoms occurring in $\varphi$.

- a Boolean formula can be represented as a tree or as a DAG
Basic notation & definitions (cont)

- **Total truth assignment** $\mu$ for $\varphi$:
  
  $\mu : \text{Atoms}(\varphi) \mapsto \{\top, \bot\}$.

- **Partial Truth assignment** $\mu$ for $\varphi$:
  
  $\mu : \mathcal{A} \mapsto \{\top, \bot\}$, $\mathcal{A} \subset \text{Atoms}(\varphi)$.

- Set and formula representation of an assignment:
  
  - $\mu$ can be represented as a set of literals:
    
    EX: $\{\mu(A_1) := \top, \mu(A_2) := \bot\}$ $\implies$ $\{A_1, \neg A_2\}$

  - $\mu$ can be represented as a formula:
    
    EX: $\{\mu(A_1) := \top, \mu(A_2) := \bot\}$ $\implies$ $A_1 \land \neg A_2$
Basic notation & definitions (cont)

- \( \mu \models \varphi \) (\( \mu \) satisfies \( \varphi \)):
  - \( \mu \models A_i \iff \mu(A_i) = \top \)
  - \( \mu \models \neg \varphi \iff \text{not} \ \mu \models \varphi \)
  - \( \mu \models \varphi_1 \land \varphi_2 \iff \mu \models \varphi_1 \text{ and } \mu \models \varphi_2 \)
  - \( \mu \models \varphi_1 \lor \varphi_2 \iff \mu \models \varphi_1 \text{ or } \mu \models \varphi_2 \)
  - \( \mu \models \varphi_1 \rightarrow \varphi_2 \iff \text{if } \mu \models \varphi_1, \text{ then } \mu \models \varphi_2 \)
  - \( \mu \models \varphi_1 \leftrightarrow \varphi_2 \iff \mu \models \varphi_1 \text{ iff } \mu \models \varphi_2 \)

- \( \varphi \) is satisfiable iff \( \mu \models \varphi \) for some \( \mu \)

- \( \varphi_1 \models \varphi_2 \) (\( \varphi_1 \) entails \( \varphi_2 \)):
  \( \varphi_1 \models \varphi_2 \) iff for every \( \mu \) \( \mu \models \varphi_1 \implies \mu \models \varphi_2 \)

- \( \models \varphi \) (\( \varphi \) is valid):
  \( \models \varphi \) iff for every \( \mu \) \( \mu \models \varphi \)

Property

- \( \varphi \) is valid \( \iff \neg \varphi \) is not satisfiable
Conjunctive Normal Form (CNF)

- \( \varphi \) is in **Conjunctive normal form** iff it is a conjunction of disjunctions of literals:

\[
L \bigwedge K_i \bigvee l_{ji} \quad \quad i=1 \quad j_i=1
\]

- the disjunctions of literals \( \bigvee_{j_i=1}^{K_i} l_{ji} \) are called **clauses**

- Easier to handle: list of lists of literals.
  \( \implies \) no reasoning on the recursive structure of the formula
Labeling CNF conversion $\text{CNF}_{\text{label}}(\varphi)$ [66, 33]

- Every $\varphi$ can be reduced into CNF by, e.g., applying recursively bottom-up the rules:
  - $\varphi \implies \varphi[(l_i \lor l_j)|B] \land \text{CNF}(B \leftrightarrow (l_i \lor l_j))$
  - $\varphi \implies \varphi[(l_i \land l_j)|B] \land \text{CNF}(B \leftrightarrow (l_i \land l_j))$
  - $\varphi \implies \varphi[(l_i \leftrightarrow l_j)|B] \land \text{CNF}(B \leftrightarrow (l_i \leftrightarrow l_j))$

$l_i, l_j$ being literals and $B$ being a “new” variable.

- Worst-case linear.

- $\text{Atoms}(\text{CNF}_{\text{label}}(\varphi)) \supseteq \text{Atoms}(\varphi)$.

- $\text{CNF}_{\text{label}}(\varphi)$ is equi-satisfiable w.r.t. $\varphi$.

- More used in practice.
Labeling CNF conversion $CNF_{label}(\varphi)$ (cont.)

| CNF($B \leftrightarrow (l_i \lor l_j)$) | $\iff$ | $(\neg B \lor l_i \lor l_j) \land$ |
|                                           |     | $(B \lor \neg l_i) \land$ |
|                                           |     | $(B \lor \neg l_j)$ |

| CNF($B \leftrightarrow (l_i \land l_j)$) | $\iff$ | $(\neg B \lor l_i) \land$ |
|                                           |     | $(\neg B \lor l_j) \land$ |
|                                           |     | $(B \lor \neg l_i \land \neg l_j)$ |

| CNF($B \leftrightarrow (l_i \leftrightarrow l_j)$) | $\iff$ | $(\neg B \lor \neg l_i \lor l_j) \land$ |
|                                           |     | $(\neg B \lor l_i \lor \neg l_j) \land$ |
|                                           |     | $(B \lor l_i \lor l_j) \land$ |
|                                           |     | $(B \lor \neg l_i \lor \neg l_j)$ |
Labeling CNF conversion $CNF_{label}$ – example

$CNF(B_1 \leftrightarrow (\neg A_3 \lor A_1)) \land \land \left\{ \begin{array}{l} (\neg B_1 \lor \neg A_3 \lor A_1) \land \\
(B_1 \lor A_3) \land \\
(B_1 \lor \neg A_3) \land \\
\end{array} \right\}$

$CNF(B_8 \leftrightarrow (A_1 \lor \neg A_4)) \land \land \left\{ \begin{array}{l} \vdots \land \\
\vdots \land \\
\vdots \land \\
\end{array} \right\}$

$CNF(B_9 \leftrightarrow (B_1 \leftrightarrow B_2)) \land \land \left\{ \begin{array}{l} \vdots \land \\
\vdots \land \\
\vdots \land \\
\end{array} \right\}$

$CNF(B_{12} \leftrightarrow (B_7 \land B_8)) \land \land \left\{ \begin{array}{l} \vdots \land \\
\vdots \land \\
\vdots \land \\
\end{array} \right\}$

$CNF(B_{13} \leftrightarrow (B_9 \lor B_{10})) \land \land \left\{ \begin{array}{l} \vdots \land \\
\vdots \land \\
\vdots \land \\
\end{array} \right\}$

$CNF(B_{14} \leftrightarrow (B_{11} \lor B_{12})) \land \land \left\{ \begin{array}{l} \vdots \land \\
\vdots \land \\
\vdots \land \\
\end{array} \right\}$

$CNF(B_{15} \leftrightarrow (B_{13} \land B_{14})) \land \land \left\{ \begin{array}{l} \vdots \land \\
\vdots \land \\
\vdots \land \\
\end{array} \right\}$

$B_{15}$
Resolution Rule

- Resolution of a pair of clauses with exactly one incompatible variable (resolvent/pivot):

\[
\begin{align*}
&\left( l_1 \lor \ldots \lor l_k \lor l \right) \lor \left( l_{k+1} \lor \ldots \lor l_m \lor C' \right) \\
&\left( l_1 \lor \ldots \lor l_k \lor \neg l \right) \lor \left( l_{k+1} \lor \ldots \lor l_m \lor C'' \right)
\end{align*}
\]

- Example:

\[
\begin{align*}
&\left( A \lor B \lor C \lor D \lor E \right) \lor \left( A \lor B \lor \neg C \lor F \right) \\
&\left( A \lor B \lor D \lor E \lor F \right)
\end{align*}
\]

- Note: many standard inference rules subcases of resolution:

\[
\begin{align*}
&A \rightarrow B \quad B \rightarrow C \\
&\rightarrow A \rightarrow C \\
&A \rightarrow B \quad (M. Ponens) \\
&\rightarrow B \rightarrow A \quad (M. Tollens)
\end{align*}
\]
Resolution Rules [32, 31]: Unit Propagation

- **Unit resolution:**
  \[
  \Gamma' \land (l) \land (\neg l \lor \lor i \ l_i) \\
  \frac{\Gamma' \land (l) \land (\lor i \ l_i)}{\Gamma' \land (l) \land (\lor i \ l_i)}
  \]

- **Unit subsumption:**
  \[
  \Gamma' \land (l) \land (l \lor \lor i \ l_i) \\
  \frac{\Gamma' \land (l) \land (l \lor \lor i \ l_i)}{\Gamma' \land (l)}
  \]

- **Unit propagation**: Unit resolution + Unit subsumption

“Deterministic” rule, applied before other “non-deterministic” rules!
Outline

1. Motivations and goals
2. Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities
3. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories
4. Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization
5. Conclusions & current research directions
Conflict-Driven Clause-Learning (CDCL) SAT solvers [72, 59, 40]

- Evolution of Davis-Putnam-Longeman-Loveland (DPLL) [32, 31]
- non-recursive: stack-based representation of data structures
- Perform conflict-directed backtracking (backjumping) and learning
- efficient data structures for doing and undoing assignments (e.g., two-watched-literal scheme)
- perform search restarts
- ...

Dramatically efficient: solve industrial-derived problems with $\approx 10^7$ Boolean variables and $\approx 10^7 - 10^8$ clauses!
Stack-based representation of a truth assignment $\mu$

- Assign one truth-value at a time (add one literal to a stack representing $\mu$).
- Stack partitioned into decision levels:
  - One decision literal.
  - Its implied literals.
  - Each implied literal tagged with the clause causing its unit-propagation (antecedent clause).
- Equivalent to an implication graph:
  - A node without incoming edges represent a decision literal.
  - The graph contains $l_1 \rightarrow^c l, \ldots, l_n \rightarrow^c l$ iff $c \overset{\text{def}}{=} \bigvee_{j=1}^n \neg l_i \lor l$ is the antecedent clause of $l$.

Representation of the dependencies between literals in $\mu$.
Implication graph - example

\[ c_1 : \neg A_1 \lor A_2 \lor \sqrt{\neg A_9} \]
\[ c_2 : \neg A_1 \lor A_3 \lor A_9 \lor \sqrt{\neg A_{10}} \]
\[ c_3 : \neg A_2 \lor \neg A_3 \lor A_4 \lor \sqrt{\neg A_{11}} \]
\[ c_4 : \neg A_4 \lor A_5 \lor A_{10} \lor \sqrt{\neg A_{13}} \]
\[ c_5 : \neg A_4 \lor A_6 \lor A_{11} \lor \sqrt{\neg A_{13}} \]
\[ c_6 : \neg A_5 \lor \neg A_6 \lor \sqrt{\neg A_{13}} \]
\[ c_7 : A_1 \lor A_7 \lor \neg A_{12} \lor \sqrt{\neg A_{13}} \]
\[ c_8 : A_1 \lor A_8 \lor \sqrt{\neg A_{13}} \]
\[ c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13} \lor \sqrt{\neg A_{13}} \]

...
Schema of a CDCL SAT solver

Function CDCL-SAT (formula: \( \varphi \), assignment & \( \mu \)) {
    status := preprocess(\( \varphi \), \( \mu \));

    while (1) {
        decide_next_branch(\( \varphi \), \( \mu \));

        while (1) {
            status := bcp(\( \varphi \), \( \mu \), \( \eta \));

            if (status == Sat)
                return Sat;

            if (status == Conflict) {
                blevel := analyze_conflict(\( \varphi \), \( \mu \), \( \eta \));

                if (blevel == 0)
                    return Unsat;

                else backtrack(blevel, \( \varphi \), \( \mu \));

            } else break;

        }
    }
}
Schema of a CDCL SAT solver (2)

- **preprocess**($\varphi, \mu$) simplifies $\varphi$ into an easier equisatisfiable formula (and updates $\mu$ if it is the case)
- **decide_next_branch**($\varphi, \mu$) chooses a new decision literal from $\varphi$ according to some heuristic, and adds it to $\mu$
- **bcp**($\varphi, \mu, \eta$) performs all deterministic assignments (unit), and updates $\varphi$ and $\mu$ accordingly. If this causes a conflict, $\eta$ is the subset of $\mu$ causing the conflict (conflict set).
- **analyze_conflict**($\varphi, \mu, \eta$) returns the “wrong-decision” level suggested by $\eta$ (“0” means that a conflict exists even without branching)
- **backtrack**(blevel, $\varphi, \mu$) undoes the branches up to blevel, and updates $\varphi$ and $\mu$ accordingly
Example

\[
\begin{align*}
c_1 : & \neg A_1 \lor A_2 \\
c_2 : & \neg A_1 \lor A_3 \lor A_9 \\
c_3 : & \neg A_2 \lor \neg A_3 \lor A_4 \\
c_4 : & \neg A_4 \lor A_5 \lor A_{10} \\
c_5 : & \neg A_4 \lor A_6 \lor A_{11} \\
c_6 : & \neg A_5 \lor \neg A_6 \\
c_7 : & A_1 \lor A_7 \lor \neg A_{12} \\
c_8 : & A_1 \lor A_8 \\
c_9 : & \neg A_7 \lor \neg A_8 \lor \neg A_{13} \\
\end{align*}
\]
Example

\[ c_1 : \neg A_1 \lor A_2 \]
\[ c_2 : \neg A_1 \lor A_3 \lor A_9 \]
\[ c_3 : \neg A_2 \lor \neg A_3 \lor A_4 \]
\[ c_4 : \neg A_4 \lor A_5 \lor A_{10} \]
\[ c_5 : \neg A_4 \lor A_6 \lor A_{11} \]
\[ c_6 : \neg A_5 \lor \neg A_6 \]
\[ c_7 : A_1 \lor A_7 \lor \neg A_{12} \]
\[ c_8 : A_1 \lor A_8 \]
\[ c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13} \]
\[ \ldots \]

\{ \ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots \} \\
(Initial assignment. Note: \( c_1, \ldots, c_9 \) inconsistent.)
Example

\[ c_1 : \neg A_1 \lor A_2 \]
\[ c_2 : \neg A_1 \lor A_3 \lor A_9 \]
\[ c_3 : \neg A_2 \lor \neg A_3 \lor A_4 \]
\[ c_4 : \neg A_4 \lor A_5 \lor A_{10} \]
\[ c_5 : \neg A_4 \lor A_6 \lor A_{11} \]
\[ c_6 : \neg A_5 \lor \neg A_6 \]
\[ c_7 : A_1 \lor A_7 \lor \neg A_{12} \lor \checkmark \]
\[ c_8 : A_1 \lor A_8 \lor \checkmark \]
\[ c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13} \]

\ldots

\{ \ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, A_1 \} \]
\ldots (decide \ A_1 \)

Roberto Sebastiani ()
Satisfiability Modulo Theories
July 16th 2011 36 / 132
Example

\[ c_1 : \neg A_1 \lor A_2 \] \( \checkmark \)
\[ c_2 : \neg A_1 \lor A_3 \lor A_9 \] \( \checkmark \)
\[ c_3 : \neg A_2 \lor \neg A_3 \lor A_4 \]
\[ c_4 : \neg A_4 \lor A_5 \lor A_{10} \]
\[ c_5 : \neg A_4 \lor A_6 \lor A_{11} \]
\[ c_6 : \neg A_5 \lor \neg A_6 \]
\[ c_7 : A_1 \lor A_7 \lor \neg A_{12} \] \( \checkmark \)
\[ c_8 : A_1 \lor A_8 \] \( \checkmark \)
\[ c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13} \]
\[ \ldots \]

\{ ... , \neg A_9 , \neg A_{10} , \neg A_{11} , A_{12} , A_{13} , ... , A_1 , A_2 , A_3 \} \\
(unit A_2 , A_3)
Example

$c_1 : \neg A_1 \lor A_2 \quad \checkmark$
$c_2 : \neg A_1 \lor A_3 \lor A_9 \quad \checkmark$
$c_3 : \neg A_2 \lor \neg A_3 \lor A_4 \quad \checkmark$
$c_4 : \neg A_4 \lor A_5 \lor A_{10}$
$c_5 : \neg A_4 \lor A_6 \lor A_{11}$
$c_6 : \neg A_5 \lor \neg A_6$
$c_7 : A_1 \lor A_7 \lor \neg A_{12} \quad \checkmark$
$c_8 : A_1 \lor A_8 \quad \checkmark$
$c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13}$
...

$\{ ..., \neg A_9, \neg A_{10}, \neg A_{11}, A_1, A_2, A_3, A_4 \}$
(unit $A_4$)
Example

\[ c_1 : \neg A_1 \lor A_2 \quad \checkmark \]
\[ c_2 : \neg A_1 \lor A_3 \lor A_9 \quad \checkmark \]
\[ c_3 : \neg A_2 \lor \neg A_3 \lor A_4 \quad \checkmark \]
\[ c_4 : \neg A_4 \lor A_5 \lor A_{10} \quad \checkmark \]
\[ c_5 : \neg A_4 \lor A_6 \lor A_{11} \quad \checkmark \]
\[ c_6 : \neg A_5 \lor \neg A_6 \quad \times \]
\[ c_7 : A_1 \lor A_7 \lor \neg A_{12} \quad \checkmark \]
\[ c_8 : A_1 \lor A_8 \quad \checkmark \]
\[ c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13} \]
\[ \ldots \]

\{ ..., \neg A_9, \neg A_{10}, \neg A_1 \neg A_{41}, A_{12}, A_{13}, ..., A_1, A_2, A_3, A_4, A_5, A_6 \}

(unit \( A_5, A_6 \)) \Rightarrow \text{conflict}
State-of-the-art backjumping and learning [77]

- Idea: when a branch $\mu$ fails,
  
  (i) **conflict analysis**: find the **conflict set** $\eta \subseteq \mu$ by generating the **conflict clause** $C \overset{\text{def}}{=} \neg \eta$ via resolution from the falsified clause
  
  (ii) **learning**: add the conflict clause $C$ to the clause set
  
  (iii) **backjumping**: backtrack to the highest branching point s.t. the stack contains all-but-one literals in $\eta$, and then unit-propagate the unassigned literal on $C$

  $\implies$ may climb up to many decision levels in the stack

- if $\eta (\neg C)$ entirely assigned at level 0, then return UNSAT
Conflict analysis: build a conflict clause by resolution

1. $C :=$ falsified clause (*conflicting clause*)
2. repeat
   (i) resolve the current clause $C$ with the antecedent clause of the last unit-propagated literal $l$ in $C$
   until $C$ verifies some given termination criteria
Conflict analysis: build a conflict clause by resolution

1. $C :=$ falsified clause (conflicting clause)
2. repeat
   (i) resolve the current clause $C$ with the antecedent clause of the last unit-propagated literal $l$ in $C$
   until $C$ verifies some given termination criterium:
   - decision

...until $C$ contains only decision literals

Conflicting cl.: $\neg A_6 \lor \neg A_5 \lor A_11$

$A_2 \lor \neg A_1 \lor A_9 \lor A_{10} \lor A_{11}$

$\neg A_1 \lor A_3 \lor A_9$

$\neg A_2 \lor \neg A_3 \lor A_{10} \lor A_{11}$

$\neg A_1 \lor A_9 \lor A_{10} \lor A_{11}$

$(A_2)$

$(A_3)$

$(A_4)$

$(A_5)$

$(A_6)$
Conflict analysis: build a conflict clause by resolution

1. $C :=$ falsified clause (conflicting clause)
2. repeat
   (i) resolve the current clause $C$ with the antecedent clause of the last unit-propagated literal $l$ in $C$
   until $C$ verifies some given termination criteria

criterium: 1st UIP

... until $C$ contains only one literal assigned at current decision level (1st UIP)

\[
\neg A_4 \lor A_5 \lor A_{10} \quad \neg A_4 \lor A_6 \lor A_{11} \quad \neg A_5 \lor \neg A_6
\]

Conflicting cl.

\[
\neg A_4 \lor A_{10} \lor A_{11} \quad (A_5)
\]

Roberto Sebastiani

Satisfiability Modulo Theories

July 16th 2011
Conflict analysis: build a conflict clause by resolution

1. $C :=$ falsified clause (conflicting clause)
2. repeat
   (i) resolve the current clause $C$ with the antecedent clause of the last unit-propagated literal $l$ in $C$
   until $C$ verifies some given termination criteria

Note:
Equivalent to finding a partition in the implication graph of $\mu$ with all decision literals on one side and the conflict on the other.

Note
$\varphi \models C$, so that $C$ can be safely added to $C$. 
Example

\[ c_1 : \neg A_1 \vee A_2 \quad \checkmark \]
\[ c_2 : \neg A_1 \vee A_3 \vee A_9 \quad \checkmark \]
\[ c_3 : \neg A_2 \vee \neg A_3 \vee A_4 \quad \checkmark \]
\[ c_4 : \neg A_4 \vee A_5 \vee A_{10} \quad \checkmark \]
\[ c_5 : \neg A_4 \vee A_6 \vee A_{11} \quad \checkmark \]
\[ c_6 : \neg A_5 \vee \neg A_6 \quad \times \]
\[ c_7 : A_1 \vee A_7 \vee \neg A_{12} \quad \checkmark \]
\[ c_8 : A_1 \vee A_8 \quad \checkmark \]
\[ c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13} \]
...

\[ \implies \text{Conflict set: } \{ \neg A_{10}, \neg A_{11}, A_4 \} \text{, learn } c_{10} := A_{10} \vee A_{11} \vee \neg A_4 \]
Example

\[ \begin{align*}
  c_1 & : \neg A_1 \lor A_2 \\
  c_2 & : \neg A_1 \lor A_3 \lor A_9 \\
  c_3 & : \neg A_2 \lor \neg A_3 \lor A_4 \\
  c_4 & : \neg A_4 \lor A_5 \lor A_{10} \\
  c_5 & : \neg A_4 \lor A_6 \lor A_{11} \\
  c_6 & : \neg A_5 \lor \neg A_6 \\
  c_7 & : A_1 \lor A_7 \lor \neg A_{12} \\
  c_8 & : A_1 \lor A_8 \\
  c_9 & : \neg A_7 \lor \neg A_8 \lor \neg A_{13} \\
  c_{10} & : A_{10} \lor A_{11} \lor \neg A_4 \\
  \ldots
\end{align*} \]

\[ \Rightarrow \text{backtrack up to } A_{11} \Rightarrow \{ \ldots, \neg A_9, \neg A_{10}, \neg A_{11} \} \]
Example

\[ c_1 : \neg A_1 \lor A_2 \]
\[ c_2 : \neg A_1 \lor A_3 \lor A_9 \]
\[ c_3 : \neg A_2 \lor \neg A_3 \lor A_4 \]
\[ c_4 : \neg A_4 \lor A_5 \lor A_{10} \checkmark \]
\[ c_5 : \neg A_4 \lor A_6 \lor A_{11} \checkmark \]
\[ c_6 : \neg A_5 \lor \neg A_6 \]
\[ c_7 : A_1 \lor A_7 \lor \neg A_{12} \]
\[ c_8 : A_1 \lor A_8 \]
\[ c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13} \]
\[ c_{10} : A_{10} \lor A_{11} \lor \neg A_4 \checkmark \]

\[ \Rightarrow \text{unit propagate } \neg A_4 \Rightarrow \{ \ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_4 \} \ldots \]
Learning – example

\[c_1 : \neg A_1 \lor A_2\]
\[c_2 : \neg A_1 \lor A_3 \lor A_9\]
\[c_3 : \neg A_2 \lor \neg A_3 \lor A_4\]
\[c_4 : \neg A_4 \lor A_5 \lor A_{10}\]
\[c_5 : \neg A_4 \lor A_6 \lor A_{11}\]
\[c_6 : \neg A_5 \lor \neg A_6\]
\[c_7 : A_1 \lor A_7 \lor \neg A_{12}\]
\[c_8 : A_1 \lor A_8\]
\[c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13}\]
\[c_{10} : A_9 \lor A_{10} \lor A_{11} \lor \neg A_1\]
\[c_{11} : A_9 \lor A_{10} \lor A_{11} \lor \neg A_{12} \lor \neg A_{13}\]

\[\Rightarrow \text{Unit: } \{\neg A_1, \neg A_{13}\}\]
State-of-the-art backjumping and learning: intuitions

- **Backjumping**: allows for climbing up to many decision levels in the stack
  - intuition: “go back to the oldest decision where you’d have done something different if only you had known $C$”
  - $\implies$ may avoid lots of redundant search

- **Learning**: in future branches, when all-but-one literals in $\eta$ are assigned, the remaining literal is assigned to false by unit-propagation:
  - intuition: “when you’re about to repeat the mistake, do the opposite of the last step”
  - $\implies$ avoids finding the same conflict again
Drawbacks of Learning & Clause discharging

Problem with Learning

Learning can generate exponentially-many clauses
- may cause a blowup in space
- may drastically slow down BCP

A solution: clause discharging

Techniques to drop learned clauses when necessary
- according to their size
- according to their activity.

A clause is currently active if it occurs in the current implication graph (i.e., it is the antecedent clause of a literal in the current assignment).
Drawbacks of Learning & Clause discharging

Problem with Learning

Learning can generate exponentially-many clauses
- may cause a blowup in space
- may drastically slow down BCP

A solution: clause discharging

Techniques to drop learned clauses when necessary
- according to their size
- according to their activity.

A clause is currently active if it occurs in the current implication graph (i.e., it is the antecedent clause of a literal in the current assignment).
Drawbacks of Learning & Clause discharging

- **Is clause-discharging safe?**
- Yes, if done properly.

Property (see, e.g., [65])

In order to guarantee correctness, completeness & termination of a CDCL solver, it suffices to keep each clause until it is active.

⇒ CDCL solvers require polynomial space
Drawbacks of Learning & Clause discharging

- Is clause-discharging safe?
- Yes, if done properly.

**Property (see, e.g., [65])**

In order to guarantee correctness, completeness & termination of a CDCL solver, it suffices to keep each clause until it is active.

⇒ **CDCL solvers require polynomial space**
Outline

1. Motivations and goals
2. Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities
3. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories
4. Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization
5. Conclusions & current research directions
Building Proofs of Unsatisfiability in CDCL SAT solvers

- recall: each conflict clause $C_i$ learned is computed from the conflicting clause $C_{i-k}$ by backward resolving with the antecedent clause of one literal

- each resolution (sub)proof can be easily tracked:
  
  $k \ i-k \rightarrow i-k-1$
  
  $\ldots$
  
  $2 \ i-2 \rightarrow i-1$
  
  $1 \ i-1 \rightarrow i$
... in particular, if $\varphi$ is unsatisfiable, the last step produces “false” as conflict clause:

\[
\begin{array}{c}
C_1 \quad C_{i-1} \\
C_2 \quad C_{i-2} \\
\vdots \\
C_k \quad C_{i-k}
\end{array}
\]

conflicting clause

\[
C_i - k
\]

note: $C_1 = l$, $C_{i-1} = \neg l$ for some literal $l$

$C_1, \ldots, C_k$, and $C_{i-k}$ are either original or learned clauses...
Building Proofs of Unsatisfiability in CDCL SAT solvers

Starting from the previous proof of unsatisfiability, repeat recursively:

- for every **learned** leaf clause \( C_i \), substitute \( C_i \) with the resolution proof generating it

until all leaf clauses are original clauses

\[
C_{11} \quad \ldots \quad C_{1i} \quad \ldots \quad C_{1j_i} \\
\vdots \\
C_1 \\
\downarrow
\]

\[
C_2 \quad \ldots \quad C_{i-1} \\
\vdots \\
C_i \\
\downarrow \\
\perp = \Rightarrow \text{we obtain a resolution proof of unsatisfiability for (a subset of) the clauses in } \varphi
\]
SAT under assumptions: $\text{SAT}(\varphi, \{l_1, \ldots, l_n\})$

- Many SAT solvers allow for solving a CNF formula $\varphi$ under a set of assumption literals $A \overset{\text{def}}{=} \{l_1, \ldots, l_n\} : \text{SAT}(\varphi, \{l_1, \ldots, l_n\})$
  - $\text{SAT}(\varphi, \{l_1, \ldots, l_n\})$: same result as $\text{SAT}(\varphi \land \bigwedge_{i=1}^{n} l_i)$

- Idea:
  - $l_1, \ldots, l_n$ “decided” at decision level 0 before starting the search
  - if backjump to level 0 on $C \overset{\text{def}}{=} \neg \eta$ s.t. $\eta \subseteq A$, then return UNSAT
  - if the “decision” strategy for conflict analysis is used, then $\eta$ is the subset of assumptions causing the inconsistency

- incremental calls $\text{SAT}(\varphi, A_1), \ldots, \text{SAT}(\varphi, A_n)$ without restarting
  - stack-based interface for $\{l_1, \ldots, l_n\}$
  - reuse of search (e.g. learned clauses) from call to call
SAT under assumptions: $\text{SAT}(\varphi, \{l_1, \ldots, l_n\})$

- Many SAT solvers allow for solving a CNF formula $\varphi$ under a set of assumption literals $A \stackrel{\text{def}}{=} \{l_1, \ldots, l_n\} : \text{SAT}(\varphi, \{l_1, \ldots, l_n\})$
  - $\text{SAT}(\varphi, \{l_1, \ldots, l_n\})$: same result as $\text{SAT}(\varphi \land \bigwedge_{i=1}^{n} l_i)$

- Idea:
  - $l_1, \ldots, l_n$ “decided” at decision level 0 before starting the search
  - if backjump to level 0 on $C \stackrel{\text{def}}{=} \neg \eta$ s.t. $\eta \subseteq A$, then return UNSAT
  - if the “decision” strategy for conflict analysis is used, then $\eta$ is the subset of assumptions causing the inconsistency

- incremental calls $\text{SAT}(\varphi, A_1), \ldots, \text{SAT}(\varphi, A_n)$ without restarting
  - stack-based interface for $\{l_1, \ldots, l_n\}$
  - reuse of search (e.g. learned clauses) from call to call
SAT under assumptions: \( \textit{SAT}(\varphi, \{l_1, \ldots, l_n\}) \)

- Many SAT solvers allow for solving a CNF formula \( \varphi \) under a set of assumption literals \( A \overset{\text{def}}{=} \{l_1, \ldots, l_n\} : \textit{SAT}(\varphi, \{l_1, \ldots, l_n\}) \)

  - \( \textit{SAT}(\varphi, \{l_1, \ldots, l_n\}) \): same result as \( \textit{SAT}(\varphi \land \bigwedge_{i=1}^{n} l_i) \)

- Idea:
  - \( l_1, \ldots, l_n \) “decided” at decision level 0 before starting the search
  - if backjump to level 0 on \( C \overset{\text{def}}{=} \neg \eta \) s.t. \( \eta \subseteq A \), then return UNSAT
  - if the “decision” strategy for conflict analysis is used, then \( \eta \) is the subset of assumptions causing the inconsistency

- incremental calls \( \textit{SAT}(\varphi, A_1), \ldots, \textit{SAT}(\varphi, A_n) \) without restarting
  - stack-based interface for \( \{l_1, \ldots, l_n\} \)
  - reuse of search (e.g. learned clauses) from call to call
Selection of sub-formulas

Let $\varphi$ be $\bigwedge_{i=1}^{n} C_i$.

**Idea [40, 54]**

- let $S_1\ldots S_n$ be fresh Boolean atoms (*selection variables*).
- let $\eta \overset{\text{def}}{=} \{S_{i_1}, \ldots, S_{i_K}\} \subseteq \{S_1, \ldots, S_n\}$
- $\text{SAT}( \bigwedge_{i=1}^{n} (\neg S_i \lor C_i), \eta )$: same as $\text{SAT}( \bigwedge_{i=i_1}^{i_K} (C_i) )$

$\quad \Rightarrow$ allows for “selecting” (*activating*) only a subset of the clauses in $\varphi$ at each call.
Outline

1. Motivations and goals
2. Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities
3. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories
4. Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization
5. Conclusions & current research directions
Outline

1. Motivations and goals
2. Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities
3. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories
4. Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization
5. Conclusions & current research directions
Modern “lazy” SMT(\(\mathcal{T}\)) solvers

A prominent “lazy” approach [44, 2, 75, 3, 8, 35]

- a **CDCL SAT solver** is used to enumerate truth assignments \(\mu_i\) for (the Boolean abstraction of) the input formula \(\varphi\)
- a theory-specific solver \(\mathcal{T}\)-solver checks the \(\mathcal{T}\)-consistency of the set of \(\mathcal{T}\)-literals corresponding to each assignment

- Many techniques to maximize the benefits of integration [70, 10]
- Many lazy SMT tools available
  - (Barcelogic, CVC3, MathSAT, OpenSMT, SATeen, Yices, Z3, ...)
- Note: lazy SMT(\(\mathcal{T}\)) approach often also referred to as “DPLL(\(\mathcal{T}\))” approach
Basic schema: example

\( \varphi = \)

\( c_1 : \neg (2v_2 - v_3 > 2) \lor A_1 \)
\( c_2 : \neg A_2 \lor (v_1 - v_5 \leq 1) \)
\( c_3 : (3v_1 - 2v_2 \leq 3) \lor A_2 \)
\( c_4 : \neg (2v_3 + v_4 \geq 5) \lor \neg (3v_1 - v_3 \leq 6) \lor \neg A_1 \)
\( c_5 : A_1 \lor (3v_1 - 2v_2 \leq 3) \)
\( c_6 : (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \)
\( c_7 : A_1 \lor (v_3 = 3v_5 + 4) \lor A_2 \)

\( \varphi^p = \)

\( c_1 : \neg B_1 \lor A_1 \)
\( c_2 : \neg A_2 \lor B_2 \)
\( c_3 : B_3 \lor A_2 \)
\( c_4 : \neg B_4 \lor \neg B_5 \lor \neg A_1 \)
\( c_5 : B_1 \lor B_3 \)
\( c_6 : B_6 \lor B_7 \lor \neg A_1 \)
\( c_7 : A_1 \lor B_8 \lor A_2 \)

true, false

\( \mu^p = \{ \neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2 \} \)
\( \mu = \{ \neg (3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg (2v_2 - v_3 \geq 2), \neg (3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1) \} \)

\( \implies \) inconsistent in \( \mathcal{L}A(Q) \) \( \implies \) backtrack
Efficient SMT solving  Combining SAT with Theory Solvers

Basic schema: example

\[ \varphi = \]
\[ c_1 : \neg(2v_2 - v_3 > 2) \lor A_1 \]
\[ c_2 : \neg A_2 \lor (v_1 - v_5 \leq 1) \]
\[ c_3 : (3v_1 - 2v_2 \leq 3) \lor A_2 \]
\[ c_4 : \neg(2v_3 + v_4 \geq 5) \lor \neg(3v_1 - v_3 \leq 6) \lor \neg A_1 \]
\[ c_5 : A_1 \lor (3v_1 - 2v_2 \leq 3) \]
\[ c_6 : (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \]
\[ c_7 : A_1 \lor (v_3 = 3v_5 + 4) \lor A_2 \]

\[ \varphi^p = \]
\[ \neg B_1 \lor A_1 \]
\[ \neg A_2 \lor B_2 \]
\[ B_3 \lor A_2 \]
\[ \neg B_4 \lor \neg B_5 \lor \neg A_1 \]
\[ A_1 \lor B_3 \]
\[ B_6 \lor B_7 \lor \neg A_1 \]
\[ A_1 \lor B_8 \lor A_2 \]

true, false

\[ \mu^p = \{ \neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2 \} \]
\[ \mu = \{ \neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \]
\[ \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1) \} \]

\[ \Rightarrow \text{inconsistent in } \mathcal{L}A(Q) \Rightarrow \text{backtrack} \]
Basic schema: example

\[ \varphi = \neg (2v_2 - v_3 > 2) \lor A_1 \]

\[ c_1 : \neg A_2 \lor (v_1 - v_5 \leq 1) \]

\[ c_2 : (3v_1 - 2v_2 \leq 3) \lor A_2 \]

\[ c_3 : \neg (2v_3 + v_4 \geq 5) \lor \neg (3v_1 - v_3 \leq 6) \lor \neg A_1 \]

\[ c_4 : A_1 \lor (3v_1 - 2v_2 \leq 3) \]

\[ c_5 : (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \]

\[ c_6 : A_1 \lor (v_3 = 3v_5 + 4) \lor A_2 \]

\[ \text{true, false} \]

\[ \varphi^p = \neg B_1 \lor A_1 \]

\[ c_1 : \neg A_2 \lor B_2 \]

\[ c_2 : B_3 \lor A_2 \]

\[ c_3 : \neg B_4 \lor \neg B_5 \lor \neg A_1 \]

\[ c_4 : A_1 \lor B_3 \]

\[ c_5 : B_6 \lor B_7 \lor \neg A_1 \]

\[ c_6 : A_1 \lor B_8 \lor A_2 \]

\[ \mu^p = \{ \neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2 \} \]

\[ \mu = \{ \neg (3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \]

\[ \neg (2v_2 - v_3 > 2), \neg (3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1) \} \]

\[ \Rightarrow \text{inconsistent in } \mathcal{L}A(Q) \Rightarrow \text{backtrack} \]
Basic schema: example

\( \varphi = \)

\begin{align*}
c_1 & : \neg(2v_2 - v_3 > 2) \lor A_1 \\
c_2 & : \neg A_2 \lor (v_1 - v_5 \leq 1) \\
c_3 & : (3v_1 - 2v_2 \leq 3) \lor A_2 \\
c_4 & : \neg(2v_3 + v_4 \geq 5) \lor \neg(3v_1 - v_3 \leq 6) \lor \neg A_1 \\
c_5 & : A_1 \lor (3v_1 - 2v_2 \leq 3) \\
c_6 & : (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \\
c_7 & : A_1 \lor (v_3 = 3v_5 + 4) \lor A_2
\end{align*}

\( \varphi^p = \)

\begin{align*}
\mu^p & = \{ \neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2 \} \\
\mu & = \{ \neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \\
& \quad \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1) \}
\end{align*}

\( \rightarrow \) inconsistent in \( \mathcal{L} \mathcal{A}(\mathbb{Q}) \) \( \rightarrow \) backtrack
\( \mathcal{T} \)-Backjumping & \( \mathcal{T} \)-learning [49, 75, 3, 8, 35]

- Similar to Boolean backjumping & learning
- Important property of \( \mathcal{T} \)-solver:
  - Extraction of \( \mathcal{T} \)-conflict sets: if \( \mu \) is \( \mathcal{T} \)-unsatisfiable, then \( \mathcal{T} \)-solver (\( \mu \)) returns the subset \( \eta \) of \( \mu \) causing the \( \mathcal{T} \)-inconsistency of \( \mu \) (\( \mathcal{T} \)-conflict set)
- If so, the \( \mathcal{T} \)-conflict clause \( C := \neg \eta \) is used to drive the backjumping & learning mechanism of the SAT solver
  \[ \neg l_1 \lor \neg l_2 \lor \neg l_3 \lor \neg l_4 \lor l_5 \]
- The less redundant is \( \eta \), the more search is saved
Efficient SMT solving
Combining SAT with Theory Solvers

\( \mathcal{T} \)-Backjumping & \( \mathcal{T} \)-learning: example

\( \varphi = \)
\[
\begin{align*}
\varphi_1 & : \neg(2v_2 - v_3 > 2) \lor A_1 \\
\varphi_2 & : \neg A_2 \lor (v_1 - v_5 \leq 1) \\
\varphi_3 & : (3v_1 - 2v_2 \leq 3) \lor A_2 \\
\varphi_4 & : \neg(2v_3 + v_4 \geq 5) \lor \neg(3v_1 - v_3 \leq 6) \lor \neg A_1 \\
\varphi_5 & : A_1 \lor (3v_1 - 2v_2 \leq 3) \\
\varphi_6 & : (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \\
\varphi_7 & : A_1 \lor (v_3 = 3v_5 + 4) \lor A_2 \\
\varphi_8 & : (3v_1 - v_3 \leq 6) \lor \neg(v_3 = 3v_5 + 4) \lor \ldots
\end{align*}
\]

\( \varphi^p = \)
\[
\begin{align*}
\varphi^p_1 & : \neg B_1 \lor A_1 \\
\varphi^p_2 & : \neg A_2 \lor B_2 \\
\varphi^p_3 & : B_3 \lor A_2 \\
\varphi^p_4 & : \neg B_4 \lor \neg B_5 \lor \neg A_1 \\
\varphi^p_5 & : A_1 \lor B_3 \\
\varphi^p_6 & : B_6 \lor B_7 \lor \neg A_1 \\
\varphi^p_7 & : A_1 \lor B_8 \lor A_2 \\
\varphi^p_8 & : B_5 \lor \neg B_8 \lor \neg B_2
\end{align*}
\]

true, false

\( \mu^p = \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\} \)
\( \mu = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1)\} \)
\( \eta = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_1 - v_5 \leq 1)\} \)
\( \eta^p = \{\neg B_5, B_8, B_2\} \)
$\varphi =$

$c_1 : \neg(2v_2 - v_3 > 2) \lor A_1$
$c_2 : \neg A_2 \lor (v_1 - v_5 \leq 1)$
$c_3 : (3v_1 - 2v_2 \leq 3) \lor A_2$
$c_4 : \neg(2v_3 + v_4 \geq 5) \lor \neg(3v_1 - v_3 \leq 6) \lor \neg A_1$
$c_5 : A_1 \lor (3v_1 - 2v_2 \leq 3)$
$c_6 : (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor \neg A_1$
$c_7 : A_1 \lor (v_3 = 3v_5 + 4) \lor A_2$
$c_8 : (3v_1 - v_3 \leq 6) \lor \neg(v_3 = 3v_5 + 4) \lor \ldots$

$\varphi^p =$

$\neg B_1 \lor A_1$
$\neg A_2 \lor B_2$
$B_3 \lor A_2$
$\neg B_4 \lor \neg B_5 \lor \neg A_1$
$A_1 \lor B_3$
$B_6 \lor B_7 \lor \neg A_1$
$A_1 \lor B_8 \lor A_2$
$B_5 \lor \neg B_8 \lor \neg B_2$

$\mu^p = \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\}$
$\mu = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2),$
$\neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1)\}$
$\eta = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_1 - v_5 \leq 1)\}$
$\eta^p = \{\neg B_5, B_8, B_2\}$
\( \varphi = \)

\[
\begin{align*}
    c_1 : & - (2v_2 - v_3 > 2) \lor A_1 \\
    c_2 : & -A_2 \lor (v_1 - v_5 \leq 1) \\
    c_3 : & (3v_1 - 2v_2 \leq 3) \lor A_2 \\
    c_4 : & -(2v_3 + v_4 \geq 5) \lor -(3v_1 - v_3 \leq 6) \lor -A_1 \\
    c_5 : & A_1 \lor (3v_1 - 2v_2 \leq 3) \\
    c_6 : & (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor -A_1 \\
    c_7 : & A_1 \lor (v_3 = 3v_5 + 4) \lor A_2 \\
    c_8 : & (3v_1 - v_3 \leq 6) \lor -(v_3 = 3v_5 + 4) \lor ... \\
\end{align*}
\]

\( \varphi^p = \)

\[
\begin{align*}
    \mu^p = & \{ -B_5, B_8, B_6, -B_1, -B_3, A_1, A_2, B_2 \} \\
    \mu = & \{ -(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), -(2v_2 - v_3 > 2), \\
              -(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1) \} \\
    \eta = & \{ -(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_1 - v_5 \leq 1) \} \\
    \eta^p = & \{ -B_5, B_8, B_2 \}
\end{align*}
\]

true, false
$\mathcal{T}$-Backjumping & $\mathcal{T}$-learning: example (2)

\[ \varphi = \]

\[ c_1 : \neg(2v_2 - v_3 > 2) \lor A_1 \]
\[ c_2 : \neg A_2 \lor (v_1 - v_5 \leq 1) \]
\[ c_3 : (3v_1 - 2v_2 \leq 3) \lor A_2 \]
\[ c_4 : \neg(2v_3 + v_4 \geq 5) \lor \neg(3v_1 - v_3 \leq 6) \lor \neg A_1 \]
\[ c_5 : A_1 \lor (3v_1 - 2v_2 \leq 3) \]
\[ c_6 : (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \]
\[ c_7 : A_1 \lor (v_3 = 3v_5 + 4) \lor A_2 \]
\[ c'_8 : (3v_1 - v_3 \leq 6) \lor \neg(v_3 = 3v_5 + 4) \lor ... \]
\[ c_8 : (3v_1 - v_3 \leq 6) \lor \neg(v_3 = 3v_5 + 4) \lor ... \]

true, false

\[ \varphi^p = \]

\[ \neg B_1 \lor A_1 \]
\[ \neg A_2 \lor B_2 \]
\[ B_3 \lor A_2 \]
\[ \neg B_4 \lor \neg B_5 \lor \neg A_1 \]
\[ A_1 \lor B_3 \]
\[ B_6 \lor B_7 \lor \neg A_1 \]
\[ A_1 \lor B_8 \lor A_2 \]
\[ B_5 \lor \neg B_8 \lor B_1 \]
\[ B_5 \lor \neg B_8 \lor \neg B_2 \]

\[ c'_8 : \text{mixed Boolean+theory conflict clause} \]

\[ c_2 : \text{theory conflicting clause} \]
\( \mathcal{T} \)-Backjumping & \( \mathcal{T} \)-learning: example (2)

\[ \varphi = \]

\begin{align*}
    c_1 : & \quad \neg(2v_2 - v_3 > 2) \lor A_1 \\
    c_2 : & \quad \neg A_2 \lor (v_1 - v_5 \leq 1) \\
    c_3 : & \quad (3v_1 - 2v_2 \leq 3) \lor A_2 \\
    c_4 : & \quad \neg(2v_3 + v_4 \geq 5) \lor \neg(3v_1 - v_3 \leq 6) \lor \neg A_1 \\
    c_5 : & \quad A_1 \lor (3v_1 - 2v_2 \leq 3) \\
    c_6 : & \quad (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \\
    c_7 : & \quad A_1 \lor (v_3 = 3v_5 + 4) \lor A_2 \\
    c'_8 : & \quad (3v_1 - v_3 \leq 6) \lor \neg(v_3 = 3v_5 + 4) \lor \ldots \\
    c_8 : & \quad (3v_1 - v_3 \leq 6) \lor \neg(v_3 = 3v_5 + 4) \lor \ldots \quad \text{true, false}
\end{align*}

\[ \varphi^p = \]

\begin{align*}
    \neg B_1 \lor A_1 \\
    \neg A_2 \lor B_2 \\
    B_3 \lor A_2 \\
    \neg B_4 \lor \neg B_5 \lor \neg A_1 \\
    A_1 \lor B_3 \\
    B_6 \lor B_7 \lor \neg A_1 \\
    A_1 \lor B_8 \lor A_2 \\
    B_5 \lor \neg B_8 \lor B_1 \\
    B_5 \lor \neg B_8 \lor \neg B_2 \\
    c'_8 : & \quad B_5 \lor \neg B_8 \lor B_1
\end{align*}

\( c_8 : \) theory conflicting clause

\[ \overbrace{B_5 \lor \neg B_8 \lor \neg B_2}^{c_8} \]

\[ \overbrace{\neg A_2 \lor B_2}^{c_2} \]

\[ \overbrace{\neg A_2 \lor B_2}^{(B_2)} \]

\[ \overbrace{B_5 \lor \neg B_8 \lor A_2}^{c_3} \]

\[ \overbrace{B_3 \lor A_2}^{(\neg A_2)} \]

\[ \overbrace{B_5 \lor B_1 \lor \neg B_3}^{c_\mathcal{T}} \]

\[ (B_3) \]

\( c'_8 : \) mixed Boolean+theory conflict clause

\[ \overbrace{B_5 \lor \neg B_8 \lor B_1}^{c'_8} \]
\( \varphi = \)
\begin{align*}
c_1 : \quad & \neg(2v_2 - v_3 > 2) \lor A_1 \\
c_2 : \quad & \neg A_2 \lor (v_1 - v_5 \leq 1) \\
c_3 : \quad & (3v_1 - 2v_2 \leq 3) \lor A_2 \\
c_4 : \quad & \neg(2v_3 + v_4 \geq 5) \lor \neg(3v_1 - v_3 \leq 6) \lor \neg A_1 \\
c_5 : \quad & A_1 \lor (3v_1 - 2v_2 \leq 3) \\
c_6 : \quad & (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \\
c_7 : \quad & A_1 \lor (v_3 = 3v_5 + 4) \lor A_2 \\
c_8' : \quad & (3v_1 - v_3 \leq 6) \lor \neg (v_3 = 3v_5 + 4) \lor ... \\
c_8 : \quad & (3v_1 - v_3 \leq 6) \lor \neg (v_3 = 3v_5 + 4) \lor ...
\end{align*}

true, false

\( \varphi^p = \)
\begin{align*}
\neg B_1 \lor A_1 \\
\neg A_2 \lor B_2 \\
B_3 \lor A_2 \\
\neg B_4 \lor \neg B_5 \lor \neg A_1 \\
A_1 \lor B_3 \\
B_6 \lor B_7 \lor \neg A_1 \\
A_1 \lor B_8 \lor A_2 \\
B_5 \lor \neg B_8 \lor B_1 \\
B_5 \lor \neg B_8 \lor \neg B_2
\end{align*}

\( c_8 : \) theory conflicting clause

\[
\frac{B_5 \lor \neg B_8 \lor \neg B_2}{B_5 \lor \neg B_8 \lor \neg A_2}
\]

\( c_2 \)

\[
\frac{\neg A_2 \lor B_2}{B_2}
\]

\( B_2 \)

\( c_3 \)

\[
\frac{B_3 \lor A_2}{\neg A_2}
\]

\( c_T \)

\[
\frac{B_5 \lor B_1 \lor \neg B_3}{B_3}
\]

\( B_3 \)

\( c'_8 : \) mixed Boolean+theory conflict clause

\[
\frac{B_5 \lor \neg B_8 \lor B_1}{B_5 \lor \neg B_8 \lor \neg B_2}
\]
Early Pruning [44, 2, 75]

- Introduce a $\mathcal{T}$-satisfiability test on \textit{intermediate assignments}: if $\mathcal{T}$-solver returns UNSAT, the procedure backtracks.
  - benefit: prunes drastically the Boolean search
  - Drawback: possibly \textbf{many useless calls to $\mathcal{T}$-solver}
Early Pruning [44, 2, 75] II

Different strategies for interleaving Boolean search steps and \( T \)-solver calls

- **Eager E.P.** [75, 11, 74, 43]: invoke \( T \)-solver every time a new \( T \)-atom is added to the assignment (unit propagations included).

- **Selective E.P.**: Do not call \( T \)-solver if the have been added only literals which hardly cause any \( T \)-conflict with the previous assignment (e.g., Boolean literals, disequalities \( (x - y \neq 3) \), \( T \)-literals introducing new variables \( (x - z = 3) \)).

- **Weakened E.P.**: for intermediate checks only, use weaker but faster versions of \( T \)-solver (e.g., check \( \mu \) on \( \mathbb{R} \) rather than on \( \mathbb{Z} \)):
  \[
  \{(x - y \leq 4), (z - x \leq -6), (z = y), (3x + 2y - 3z = 4)\} 
  \]
Early pruning: example

\[ \varphi = \{ \neg(2v_2 - v_3 > 2) \lor A_1 \} \land \\
\{ \neg A_2 \lor (2v_1 - 4v_5 > 3) \} \land \\
\{ 3v_1 - 2v_2 \leq 3 \} \lor A_2 \} \land \\
\{ \neg(2v_3 + v_4 \geq 5) \lor \neg(3v_1 - v_3 \leq 6) \lor \neg A_1 \} \land \\
\{ A_1 \lor (3v_1 - 2v_2 \leq 3) \} \land \\
\{ (v_1 - v_5 \leq 1) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \} \land \\
\{ A_1 \lor (v_3 = 3v_5 + 4) \lor A_2 \}. \]

\[ \varphi^p = \{ \neg B_1 \lor A_1 \} \land \\
\{ \neg A_2 \lor B_2 \} \land \\
\{ B_3 \lor A_2 \} \land \\
\{ \neg B_4 \lor \neg B_5 \lor \neg A_1 \} \land \\
\{ A_1 \lor B_3 \} \land \\
\{ B_6 \lor B_7 \lor \neg A_1 \} \land \\
\{ A_1 \lor B_8 \lor A_2 \}. \]

- Suppose it is built the intermediate assignment:
  \[ \mu' = \neg B_1 \land \neg A_2 \land B_3 \land \neg B_5. \]
  corresponding to the following set of \( \mathcal{T} \)-literals
  \[ \mu' = \neg(2v_2 - v_3 > 2) \land \neg A_2 \land (3v_1 - 2v_2 \leq 3) \land \neg(3v_1 - v_3 \leq 6). \]

- If \( \mathcal{T} \)-solver is invoked on \( \mu' \), then it returns \text{UNSAT}, and DPLL backtracks without exploring any extension of \( \mu' \).
Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$-solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$-solver*:

  - $\mathcal{T}$-solver $(\mu_1)$ $\Rightarrow$ Sat
  - $\mathcal{T}$-solver $(\mu_1 \cup \mu_2)$ $\Rightarrow$ Sat
  - $\mathcal{T}$-solver $(\mu_1 \cup \mu_2 \cup \mu_3)$ $\Rightarrow$ Sat
  - $\mathcal{T}$-solver $(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$ $\Rightarrow$ Unsat

  - Undo $\mu_4$, $\mu_3$, $\mu_2$
  - Undo $\mu_4$, $\mu_3$
  - Undo $\mu_4$

  - $\mathcal{T}$-solver $(\mu_1 \cup \mu_2')$ $\Rightarrow$ Sat
  - $\mathcal{T}$-solver $(\mu_1 \cup \mu_2' \cup \mu_3')$ $\Rightarrow$ Sat

  - ... 

- Desirable features of $\mathcal{T}$-solvers:
  - **incrementality**: $\mathcal{T}$-solver$(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}$-solver$(\mu_1)$ without restarting from scratch
  - **backtrackability (resettablity)**: $\mathcal{T}$-solver can efficiently undo steps and return to a previous status on the stack

- $\mathcal{T}$-solver requires a **stack-based interface**
Early pruning: remark

Incrementality & Backtrackability of $T$-solvers

- With early pruning, lots of *incremental calls to $T$-solver*:

  \[
  \begin{align*}
  T\text{-solver}(\mu_1) & \Rightarrow \text{Sat} \quad \text{Undo } \mu_4, \mu_3, \mu_2 \\
  T\text{-solver}(\mu_1 \cup \mu_2) & \Rightarrow \text{Sat} \\
  T\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3) & \Rightarrow \text{Sat} \\
  T\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4) & \Rightarrow \text{Unsat}
  \end{align*}
  \]

  \[\Rightarrow \text{Desirable features of } T\text{-solvers:}\]

  - *incrementality:* $T$-solver$(\mu_1 \cup \mu_2)$ reuses computation of $T$-solver$(\mu_1)$ without restarting from scratch
  - *backtrackability (resettability):* $T$-solver can efficiently undo steps and return to a previous status on the stack

  \[\Rightarrow T\text{-solver requires a stack-based interface}\]
Incrementality & Backtrackability of $\mathcal{T}$-solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$-solver*:

  \[
  \begin{align*}
  \mathcal{T}\text{-solver} (\mu_1) & \Rightarrow \text{Sat} & \text{Undo} \mu_4, \mu_3, \mu_2 \\
  \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2) & \Rightarrow \text{Sat} & \mathcal{T}\text{-solver} (\mu_1 \cup \mu'_2) & \Rightarrow \text{Sat} \\
  \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2 \cup \mu_3) & \Rightarrow \text{Sat} & \mathcal{T}\text{-solver} (\mu_1 \cup \mu'_2 \cup \mu'_3) & \Rightarrow \text{Sat} \\
  \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4) & \Rightarrow \text{Unsat} & \ldots 
  \end{align*}
  \]

$\implies$ Desirable features of $\mathcal{T}$-solvers:

- **incrementality**: $\mathcal{T}$-solver$(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}$-solver$(\mu_1)$ without restarting from scratch
- **backtrackability (resettablility)**: $\mathcal{T}$-solver can efficiently undo steps and return to a previous status on the stack

$\implies$ $\mathcal{T}$-solver requires a *stack-based interface*
Early pruning: remark

### Incrementality & Backtrackability of $\mathcal{T}$-solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$-solver*:

  - $\mathcal{T}$-solver ($\mu_1$) $\Rightarrow$ *Sat*\hspace{1cm} Undo $\mu_4$, $\mu_3$, $\mu_2$
  - $\mathcal{T}$-solver ($\mu_1 \cup \mu_2$) $\Rightarrow$ *Sat* $\hspace{1cm} \mathcal{T}$-solver ($\mu_1 \cup \mu'_2$) $\Rightarrow$ *Sat*
  - $\mathcal{T}$-solver ($\mu_1 \cup \mu_2 \cup \mu_3$) $\Rightarrow$ *Sat* $\hspace{1cm} \mathcal{T}$-solver ($\mu_1 \cup \mu'_2 \cup \mu'_3$) $\Rightarrow$ *Sat*
  - $\mathcal{T}$-solver ($\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4$) $\Rightarrow$ *Unsat* $\hspace{1cm} \ldots$

### Desirable features of $\mathcal{T}$-solvers:

- **Incrementality**: $\mathcal{T}$-solver($\mu_1 \cup \mu_2$) reuses computation of $\mathcal{T}$-solver($\mu_1$) without restarting from scratch
- **Backtrackability (resettablity)**: $\mathcal{T}$-solver can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$ $\mathcal{T}$-solver requires a stack-based interface
Early pruning: remark

Incrementality & Backtrackability of $T$-solvers

- With early pruning, lots of *incremental calls to* $T$-solver:

$$
\begin{align*}
T\text{-solver}(\mu_1) & \Rightarrow Sat & \text{Undo } \mu_4, \mu_3, \mu_2 \\
T\text{-solver}(\mu_1 \cup \mu_2) & \Rightarrow Sat & T\text{-solver}(\mu_1 \cup \mu'_2) & \Rightarrow Sat \\
T\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3) & \Rightarrow Sat & T\text{-solver}(\mu_1 \cup \mu'_2 \cup \mu'_3) & \Rightarrow Sat \\
T\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4) & \Rightarrow Unsat & \ldots
\end{align*}
$$

Desirable features of $T$-solvers:

- **incrementality**: $T\text{-solver}(\mu_1 \cup \mu_2)$ reuses computation of $T\text{-solver}(\mu_1)$ without restarting from scratch
- **backtrackability (resettablility)**: $T$-solver can efficiently undo steps and return to a previous status on the stack

$T$-solver requires a stack-based interface
Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$-solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$-solver*:

  \[
  \begin{align*}
  \mathcal{T}\text{-solver} (\mu_1) & \Rightarrow \text{Sat} & \text{Undo } \mu_4, \mu_3, \mu_2 \\
  \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2) & \Rightarrow \text{Sat} & \mathcal{T}\text{-solver} (\mu_1 \cup \mu'_2) & \Rightarrow \text{Sat} \\
  \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2 \cup \mu_3) & \Rightarrow \text{Sat} & \mathcal{T}\text{-solver} (\mu_1 \cup \mu'_2 \cup \mu'_3) & \Rightarrow \text{Sat} \\
  \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4) & \Rightarrow \text{Unsat} & \text{...} \\
  \end{align*}
  \]

- **Desirable features of $\mathcal{T}$-solvers:**
  - *incrementality*: $\mathcal{T}$-solver($\mu_1 \cup \mu_2$) reuses computation of $\mathcal{T}$-solver($\mu_1$) without restarting from scratch.
  - *backtrackability (resettability)*: $\mathcal{T}$-solver can efficiently undo steps and return to a previous status on the stack.

$\mathcal{T}$-solver requires a *stack-based interface*.
Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$-solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$-solver*:

  \[
  \begin{align*}
  \mathcal{T}\text{-solver} (\mu_1) &\Rightarrow \text{Sat} & \text{Undo } \mu_4, \mu_3, \mu_2 \\
  \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2) &\Rightarrow \text{Sat} & \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2') &\Rightarrow \text{Sat} \\
  \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2 \cup \mu_3) &\Rightarrow \text{Sat} & \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2' \cup \mu_3') &\Rightarrow \text{Sat} \\
  \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4) &\Rightarrow \text{Unsat} & \text{...} \\
  \end{align*}
  \]

- *Desirable features of $\mathcal{T}$-solvers*:
  - *incrementality*: $\mathcal{T}$-solver$(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}$-solver$(\mu_1)$ without restarting from scratch
  - *backtrackability* (resettablility): $\mathcal{T}$-solver can efficiently undo steps and return to a previous status on the stack

$\mathcal{T}$-solver requires a stack-based interface
Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$-solvers

- With early pruning, lots of \textit{incremental calls to $\mathcal{T}$-solver}:
  \[
  \begin{align*}
  \mathcal{T}\text{-solver} (\mu_1) & \Rightarrow \text{Sat} & \text{Undo } \mu_4, \mu_3, \mu_2 \\
  \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2) & \Rightarrow \text{Sat} & \mathcal{T}\text{-solver} (\mu_1 \cup \mu'_2) & \Rightarrow \text{Sat} \\
  \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2 \cup \mu_3) & \Rightarrow \text{Sat} & \mathcal{T}\text{-solver} (\mu_1 \cup \mu'_2 \cup \mu'_3) & \Rightarrow \text{Sat} \\
  \mathcal{T}\text{-solver} (\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4) & \Rightarrow \text{Unsat} & & \ldots
  \end{align*}
\]

- \textbf{Desirable features of $\mathcal{T}$-solvers:}
  - \textit{incrementality}: $\mathcal{T}$-solver$(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}$-solver$(\mu_1)$ without restarting from scratch
  - \textit{backtrackability (resettability)}: $\mathcal{T}$-solver can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$ \textit{$\mathcal{T}$-solver} requires a \texttt{stack-based interface}
Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$-solvers

- With early pruning, lots of \textit{incremental calls to $\mathcal{T}$-solver}:

  $\mathcal{T}$-solver ($\mu_1$) $\Rightarrow$ Sat 
  Undo $\mu_4, \mu_3, \mu_2$

  $\mathcal{T}$-solver ($\mu_1 \cup \mu_2$) $\Rightarrow$ Sat 
  $\mathcal{T}$-solver ($\mu_1 \cup \mu'_2$) $\Rightarrow$ Sat

  $\mathcal{T}$-solver ($\mu_1 \cup \mu_2 \cup \mu_3$) $\Rightarrow$ Sat 
  $\mathcal{T}$-solver ($\mu_1 \cup \mu'_2 \cup \mu'_3$) $\Rightarrow$ Sat

  $\mathcal{T}$-solver ($\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4$) $\Rightarrow$ Unsat ... 

$\implies$ Desirable features of $\mathcal{T}$-solvers:

- \textit{incrementality}: $\mathcal{T}$-solver($\mu_1 \cup \mu_2$) reuses computation of $\mathcal{T}$-solver($\mu_1$) without restarting from scratch

- \textit{backtrackability (resettablility)}: $\mathcal{T}$-solver can efficiently undo steps and return to a previous status on the stack

$\implies$ $\mathcal{T}$-solver requires a \textit{stack-based interface}
Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$-solvers

- With early pruning, lots of \textit{incremental calls to $\mathcal{T}$-solver}: 

  \[
  \begin{align*}
  \mathcal{T}\text{-solver}\left(\mu_1\right) & \Rightarrow \text{Sat} & \text{Undo } \mu_4, \mu_3, \mu_2 \\
  \mathcal{T}\text{-solver}\left(\mu_1 \cup \mu_2\right) & \Rightarrow \text{Sat} & \mathcal{T}\text{-solver}\left(\mu_1 \cup \mu'_2\right) & \Rightarrow \text{Sat} \\
  \mathcal{T}\text{-solver}\left(\mu_1 \cup \mu_2 \cup \mu_3\right) & \Rightarrow \text{Sat} & \mathcal{T}\text{-solver}\left(\mu_1 \cup \mu'_2 \cup \mu'_3\right) & \Rightarrow \text{Sat} \\
  \mathcal{T}\text{-solver}\left(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4\right) & \Rightarrow \text{Unsat} & \text{...}
  \end{align*}
  \]

\[\Rightarrow\text{ Desirable features of } \mathcal{T}\text{-solvers:} \]

- \textit{incrementality:} $\mathcal{T}$\text{-solver}(\mu_1 \cup \mu_2) reuses computation of $\mathcal{T}$\text{-solver}(\mu_1) without restarting from scratch
- \textit{backtrackability (resettability):} $\mathcal{T}$\text{-solver} can efficiently undo steps and return to a previous status on the stack

\[\Rightarrow \mathcal{T}\text{-solver} \text{ requires a stack-based interface} \]


\( \mathcal{T} \)-Propagation [2, 3, 43]

- strictly related to early pruning
- important property of \( \mathcal{T} \)-solver:
  - \( \mathcal{T} \)-deduction: when a partial assignment \( \mu \) is \( \mathcal{T} \)-satisfiable, \( \mathcal{T} \)-solver may be able to return also an assignment \( \eta \) to some unassigned atom occurring in \( \varphi \) s.t. \( \mu \models \mathcal{T} \eta \).
  - If so:
    - the literal \( \eta \) is then unit-propagated;
    - optionally, a \( \mathcal{T} \)-deduction clause \( C := \neg \mu' \lor \eta \) can be learned, \( \mu' \) being the subset of \( \mu \) which caused the deduction (\( \mu' \models \mathcal{T} \eta \))

\( \Rightarrow \) may prune drastically the search

Both \( \mathcal{T} \)-deduction clauses and \( \mathcal{T} \)-conflict clauses are called \( \mathcal{T} \)-lemmas since they are valid in \( \mathcal{T} \)
**\( \mathcal{T} \)-Propagation [2, 3, 43]**

- strictly related to early pruning
- important property of \( \mathcal{T} \)-solver:
  - \( \mathcal{T} \)-deduction: when a partial assignment \( \mu \) is \( \mathcal{T} \)-satisfiable, \( \mathcal{T} \)-solver may be able to return also an assignment \( \eta \) to some unassigned atom occurring in \( \varphi \) s.t. \( \mu \models \mathcal{T} \eta \).

- If so:
  - the literal \( \eta \) is then unit-propagated;
  - optionally, a \( \mathcal{T} \)-deduction clause \( C := \neg \mu' \lor \eta \) can be learned, \( \mu' \) being the subset of \( \mu \) which caused the deduction (\( \mu' \models \mathcal{T} \eta \))

\( \implies \) may prune drastically the search

Both \( \mathcal{T} \)-deduction clauses and \( \mathcal{T} \)-conflict clauses are called \( \mathcal{T} \)-lemmas since they are valid in \( \mathcal{T} \).
\( T \)-propagation: example

\[ \varphi = \]

\[ c_1 : \neg(2v_2 - v_3 > 2) \lor A_1 \]
\[ c_2 : \neg A_2 \lor (v_1 - v_5 \leq 1) \]
\[ c_3 : (3v_1 - 2v_2 \leq 3) \lor A_2 \]
\[ c_4 : \neg(2v_3 + v_4 \geq 5) \lor \neg(3v_1 - v_3 \leq 6) \lor \neg A_1 \]
\[ c_5 : A_1 \lor (3v_1 - 2v_2 \leq 3) \]
\[ c_6 : (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \]
\[ c_7 : A_1 \lor (v_3 = 3v_5 + 4) \lor A_2 \]

\[ \varphi^p = \]

\[ \neg B_1 \lor A_1 \]
\[ \neg A_2 \lor B_2 \]
\[ B_3 \lor A_2 \]
\[ \neg B_4 \lor \neg B_5 \lor \neg A_1 \]
\[ A_1 \lor B_3 \]
\[ B_6 \lor B_7 \lor \neg A_1 \]
\[ A_1 \lor B_8 \lor A_2 \]

\( \text{true, false} \)

\[ \longrightarrow \text{propagate } \neg B_3 \text{ [and learn the deduction clause } B_5 \lor B_1 \lor \neg B_3 \text{]} \]
\( \mathcal{T} \) -propagation: example

\[ \varphi = \]

\begin{align*}
c_1 & : \neg (2v_2 - v_3 > 2) \lor A_1 \\
c_2 & : \neg A_2 \lor (v_1 - v_5 \leq 1) \\
c_3 & : (3v_1 - 2v_2 \leq 3) \lor A_2 \\
c_4 & : \neg (2v_3 + v_4 \geq 5) \lor \neg (3v_1 - v_3 \leq 6) \lor \neg A_1 \\
c_5 & : A_1 \lor (3v_1 - 2v_2 \leq 3) \\
c_6 & : (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \\
c_7 & : A_1 \lor (v_3 = 3v_5 + 4) \lor A_2
\end{align*}

\[ \varphi^p = \]

\begin{align*}
\neg B_1 \lor A_1 \\
\neg A_2 \lor B_2 \\
B_3 \lor A_2 \\
\neg B_4 \lor \neg B_5 \lor \neg A_1 \\
A_1 \lor B_3 \\
B_6 \lor B_7 \lor \neg A_1 \\
A_1 \lor B_8 \lor A_2
\end{align*}

true, false

\[ \mu^p = \{ \neg B_5, B_8, B_6, \neg B_1 \} \]

\[ \mu = \{ \neg (3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg (2v_2 - v_3 > 2) \} \]

\[ \models \mathcal{L}_A (Q) \underbrace{\neg (3v_1 - 2v_2 \leq 3)}_{\neg B_3} \]

\[ \rightarrow \text{propagate } \neg B_3 \text{ [and learn the deduction clause } B_5 \lor B_1 \lor \neg B_3 \]
$\varphi = \neg (2v_2 - v_3 > 2) \lor A_1$
\begin{align*}
c_1 & : \quad \neg A_2 \lor (v_1 - v_5 \leq 1) \\
c_2 & : \quad (3v_1 - 2v_2 \leq 3) \lor A_2 \\
c_3 & : \quad (2v_3 + v_4 \geq 5) \lor \neg (3v_1 - v_3 \leq 6) \lor \neg A_1 \\
c_4 & : \quad A_1 \lor (3v_1 - 2v_2 \leq 3) \\
c_5 & : \quad (v_2 - v_4 \leq 6) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \\
c_6 & : \quad A_1 \lor (v_3 = 3v_5 + 4) \lor A_2 \\
c_7 & : \quad A_1 \lor (v_3 = 3v_5 + 4) \lor A_2
\end{align*}

$\varphi^p = \neg B_1 \lor A_1$
\begin{align*}
\mu^p & = \{ \neg B_5, B_8, B_6, \neg B_1 \} \\
\mu & = \{ \neg (3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg (2v_2 - v_3 > 2) \} \\
& \models \mathcal{L}(\varphi) \ \neg (3v_1 - 2v_2 \leq 3) \ \neg B_3
\end{align*}

$\implies$ propagate $\neg B_3$ [and learn the deduction clause $B_5 \lor B_1 \lor \neg B_3$]
Pure-literal filtering [75, 3, 14]

Property

If we have non-Boolean \( \mathcal{T} \)-atoms occurring only positively [negatively] in the original formula \( \varphi \) (learned clauses are not considered), we can drop every negative [positive] occurrence of them from the assignment to be checked by \( \mathcal{T} \)-solver (and from the \( \mathcal{T} \)-deducible ones).

- increases the chances of finding a model
- reduces the effort for the \( \mathcal{T} \)-solver
- eliminates unnecessary “nasty” negated literals (e.g. negative equalities like \( \neg(3v_1 - 9v_2 = 3) \) in \( \mathcal{L}\mathcal{A}(\mathbb{Z}) \) force splitting: \( (3v_1 - 9v_2 > 3) \lor (3v_1 - 9v_2 < 3) \)).
- may weaken the effect of early pruning.
Pure literal filtering: example

\[
\varphi = \{ \neg(2v_2 - v_3 > 2) \lor A_1 \} \land \\
\{ \neg A_2 \lor (2v_1 - 4v_5 > 3) \} \land \\
\{ (3v_1 - 2v_2 \leq 3) \lor A_2 \} \land \\
\{ \neg(2v_3 + v_4 \geq 5) \lor \neg(3v_1 - v_3 \leq 6) \lor \neg A_1 \} \land \\
\{ A_1 \lor (3v_1 - 2v_2 \leq 3) \} \land \\
\{ (v_1 - v_5 \leq 1) \lor (v_5 = 5 - 3v_4) \lor \neg A_1 \} \land \\
\{ A_1 \lor (v_3 = 3v_5 + 4) \lor A_2 \} \land \\
\{ (2v_2 - v_3 > 2) \lor \neg(3v_1 - 2v_2 \leq 3) \lor (3v_1 - v_3 \leq 6) \} \quad \text{learned}
\]

\[
\mu' = \{ \neg(2v_2 - v_3 > 2), \neg A_2, (3v_1 - 2v_2 \leq 3), \neg A_1, (v_3 = 3v_5 + 4), (3v_1 - v_3 \leq 6) \}.
\]

\[\implies \text{Sat: } v_1 = v_2 = v_3 = 0, v_5 = -4/3 \text{ is a solution}
\]

\[\text{N.B. } (3v_1 - v_3 \leq 6) \text{ “filtered out” from } \mu' \text{ because it occurs only negatively in the original formula } \varphi\]
Preprocessing atoms [44, 49, 4]

Source of inefficiency: semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other]

⇒ they may be assigned different [resp. identical] truth values.
⇒ lots of redundant unsatisfiable assignment generated

Solution
Rewrite a priori trivially-equivalent atoms/literals into the same atom/literal.
Preprocessing atoms [44, 49, 4]

Source of inefficiency: semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other]

⇒ they may be assigned different [resp. identical] truth values.

⇒ lots of redundant unsatisfiable assignment generated

Solution

Rewrite a priori trivially-equivalent atoms/literals into the same atom/literal.
Preprocessing atoms [44, 49, 4]

Source of inefficiency: semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other]

⇒ they may be assigned different [resp. identical] truth values.
⇒ lots of redundant unsatisfiable assignment generated

Solution
Rewrite a priori trivially-equivalent atoms/literals into the same atom/literal.
Preprocessing atoms \([44, 49, 4]\)

Source of inefficiency: semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other]

\[\Rightarrow\] they may be assigned different [resp. identical] truth values.

\[\Rightarrow\] lots of redundant unsatisfiable assignment generated

Solution

Rewrite a priori trivially-equivalent atoms/literals into the same atom/literal.
Preprocessing atoms (cont.)

- **Sorting:** \((v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \implies (v_1 + v_2 - v_3 \leq 1))\);

- **Rewriting dual operators:**
  \((v_1 < v_2), (v_1 \geq v_2) \implies (v_1 < v_2), \neg(v_1 < v_2)\)

- **Exploiting associativity:**
  \((v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1) \implies (v_1 + v_2 + v_3 = 1);\)

- **Factoring** \((v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \implies (0.25v_1 + 0.5v_2 \leq 1.0);\)

- **Exploiting properties of \(T\):**
  \((v_1 \leq 3), (v_1 < 4) \implies (v_1 \leq 3) \text{ if } v_1 \in \mathbb{Z};\)

- ...
Preprocessing atoms (cont.)

- **Sorting:** \((v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3)\) \(\implies (v_1 + v_2 - v_3 \leq 1))\);

- **Rewriting dual operators:**
  \((v_1 < v_2), (v_1 \geq v_2) \implies (v_1 < v_2), \neg(v_1 < v_2)\)

- **Exploiting associativity:**
  \((v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1) \implies (v_1 + v_2 + v_3 = 1)\);

- **Factoring** \((v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \implies (0.25v_1 + 0.5v_2 \leq 1.0)\);

- **Exploiting properties of \(T\):**
  \((v_1 \leq 3), (v_1 < 4) \implies (v_1 \leq 3) \text{ if } v_1 \in \mathbb{Z}\);

- ...
Preprocessing atoms (cont.)

- **Sorting:** \((v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \implies (v_1 + v_2 - v_3 \leq 1))\);

- **Rewriting dual operators:**
  \((v_1 < v_2), (v_1 \geq v_2) \implies (v_1 < v_2), \neg (v_1 < v_2)\)

- **Exploiting associativity:**
  \((v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1) \implies (v_1 + v_2 + v_3 = 1)\);

- **Factoring** \((v_1 + 2.0v_2 \leq 4.0), (\neg 2.0v_1 - 4.0v_2 \geq -8.0) \implies (0.25v_1 + 0.5v_2 \leq 1.0)\);

- **Exploiting properties of \(T\):**
  \((v_1 \leq 3), (v_1 < 4) \implies (v_1 \leq 3) \text{ if } v_1 \in \mathbb{Z}\);

- ...
Preprocessing atoms (cont.)

- **Sorting:** \((v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \implies (v_1 + v_2 - v_3 \leq 1))\);

- **Rewriting dual operators:**
  \((v_1 < v_2), (v_1 \geq v_2) \implies (v_1 < v_2), \neg(v_1 < v_2)\)

- **Exploiting associativity:**
  \((v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3 = 1) \implies (v_1 + v_2 + v_3 = 1)\);

- **Factoring** \((v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \implies (0.25v_1 + 0.5v_2 \leq 1.0)\);

- **Exploiting properties of \(T\):**
  \((v_1 \leq 3), (v_1 < 4) \implies (v_1 \leq 3)\) if \(v_1 \in \mathbb{Z}\);

- ...
Preprocessing atoms (cont.)

- **Sorting:** \((v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3)\) \(\implies (v_1 + v_2 - v_3 \leq 1))\);

- **Rewriting dual operators:**
  \((v_1 < v_2), (v_1 \geq v_2) \implies (v_1 < v_2), \neg(v_1 < v_2)\)

- **Exploiting associativity:**
  \((v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1) \implies (v_1 + v_2 + v_3 = 1);\)

- **Factoring** \((v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \implies (0.25v_1 + 0.5v_2 \leq 1.0)\);

- **Exploiting properties of \(T\):**
  \((v_1 \leq 3), (v_1 < 4) \implies (v_1 \leq 3)\text{ if } v_1 \in \mathbb{Z};\)

- **...**
Static Learning [2, 4]

- Often possible to quickly detect a priori short and “obviously inconsistent” pairs or triplets of literals occurring in $\varphi$.
  - mutual exclusion $\{x = 0, x = 1\}$,
  - congruence $\{(x_1 = y_1), (x_2 = y_2), \neg(f(x_1, x_2) = f(y_1, y_2))\}$,
  - transitivity $\{(x - y = 2), (y - z \leq 4), \neg(x - z \leq 7)\}$,
  - substitution $\{(x = y), (2x - 3z \leq 3), \neg(2y - 3z \leq 3)\}$
  - ...

- Preprocessing step: detect these literals and add blocking clauses to the input formula:
  (e.g., $\neg(x = 0) \lor \neg(x = 1)$)

$\implies$ No assignment including one such group of literals is ever generated: as soon as all but one literals are assigned, the remaining one is immediately assigned false by unit-propagation.
Static Learning [2, 4]

Often possible to quickly detect a priori short and “obviously inconsistent” pairs or triplets of literals occurring in $\varphi$.

- mutual exclusion $\{x = 0, x = 1\}$,
- congruence $\{(x_1 = y_1), (x_2 = y_2), \neg(f(x_1, x_2) = f(y_1, y_2))\}$,
- transitivity $\{(x - y = 2), (y - z \leq 4), \neg(x - z \leq 7)\}$,
- substitution $\{(x = y), (2x - 3z \leq 3), \neg(2y - 3z \leq 3)\}$
- ...

Preprocessing step: detect these literals and add blocking clauses to the input formula:

(e.g., $\neg(x = 0) \lor \neg(x = 1)$)

$\Rightarrow$ No assignment including one such group of literals is ever generated: as soon as all but one literals are assigned, the remaining one is immediately assigned false by unit-propagation.
Static Learning [2, 4]

- Often possible to quickly detect a priori short and “obviously inconsistent” pairs or triplets of literals occurring in $\varphi$.
  - mutual exclusion $\{x = 0, x = 1\}$,
  - congruence $\{(x_1 = y_1), (x_2 = y_2), \neg(f(x_1, x_2) = f(y_1, y_2))\}$,
  - transitivity $\{(x - y = 2), (y - z \leq 4), \neg(x - z \leq 7)\}$,
  - substitution $\{(x = y), (2x - 3z \leq 3), \neg(2y - 3z \leq 3)\}$
  - ...

- Preprocessing step: detect these literals and add blocking clauses to the input formula:
  (e.g., $\neg(x = 0) \lor \neg(x = 1)$)

$\Rightarrow$ No assignment including one such group of literals is ever generated: as soon as all but one literals are assigned, the remaining one is immediately assigned false by unit-propagation.
Other optimization techniques

- $T$-deduced-literal filtering
- Ghost-literal filtering
- $T$-solver layering
- $T$-solver clustering
- ...

(see [70, 10] for an overview)
Other SAT-solving techniques for SMT?

Frequently-asked question:
Are CDCL SAT solvers the only suitable Boolean Engines for SMT?

Some previous attempts:
- Ordered Binary Decision Diagrams (OBDDs) [76, 58, 1]
- Stochastic Local Search [48]

CDCL based currently much more efficient.
\(\mathcal{T}\)-solvers for Equality and Uninterpreted Functions (EUF)

- EUF polynomial: \(O(n \cdot \log(n))\)
- use a congruence closure data structures (E-Graphs) [38, 61, 34], based on the Union-Find data-structure for equivalence classes
- Idea (sketch):
  - if \(t = s\), then merge the eq. classes of \(t\) and \(s\)
  - if \(\forall i \in 1...k\), \(t_i\) and \(s_i\) pairwise belong to the same eq. classes, then merge the eq. classes of \(f(t_1, ..., t_k)\) and \(f(s_1, ..., s_k)\)
  - if \(t \neq s\) and \(t\) and \(s\) belong to the same eq. class, then conflict

Example borrowed from [38].
**T-solvers for Equality and Uninterpreted Functions (EUF)**

- **EUF polynomial**: $O(n \cdot \log(n))$
- use a congruence closure data structures (E-Graphs) [38, 61, 34], based on the Union-Find data-structure for equivalence classes
- Idea (sketch):
  - if $(t = s)$, then merge the eq. classes of $t$ and $s$
  - if $\forall i \in 1...k$, $t_i$ and $s_i$ pairwise belong to the same eq. classes, then merge the the eq. classes of $f(t_1, ..., t_k)$ and $f(s_1, ..., s_k)$
  - if $(t \neq s)$ and $t$ and $s$ belong to the same eq. class, then conflict

Example borrowed from [38].

$$f(a, b) = a$$
$$f(f(a, b), b) = c$$
$$g(a) \neq g(c)$$
**T-solvers for Equality and Uninterpreted Functions (EUF)**

- **EUF polynomial:** $O(n \cdot \log(n))$
- use a congruence closure data structures (**E-Graphs**) [38, 61, 34], based on the Union-Find data-structure for equivalence classes
- **Idea (sketch):**
  - if $(t = s)$, then merge the eq. classes of $t$ and $s$
  - if $\forall i \in 1...k$, $t_i$ and $s_i$ pairwise belong to the same eq. classes, then merge the the eq. classes of $f(t_1, ..., t_k)$ and $f(s_1, ..., s_k)$
  - if $(t \neq s)$ and $t$ and $s$ belong to the same eq. class, then conflict

Example borrowed from [38].
\( \mathcal{T} \)-solvers for Equality and Uninterpreted Functions (\( \mathcal{EUF} \))

- \( \mathcal{EUF} \) polynomial: \( O(n \cdot \log(n)) \)
- use a congruence closure data structures (E-Graphs) [38, 61, 34], based on the Union-Find data-structure for equivalence classes
- Idea (sketch):
  - if \( t = s \), then merge the eq. classes of \( t \) and \( s \)
  - if \( \forall i \in 1...k \), \( t_i \) and \( s_i \) pairwise belong to the same eq. classes, then merge the the eq. classes of \( f(t_1, ..., t_k) \) and \( f(s_1, ..., s_k) \)
  - if \( t \neq s \) and \( t \) and \( s \) belong to the same eq. class, then conflict

Example borrowed from [38].
Efficient SMT solving  
Theory Solvers for theories of interest

\( \mathcal{T} \)-solvers for Equality and Uninterpreted Functions (EUF)

- EUF polynomial: \( O(n \cdot \log(n)) \)
- use a congruence closure data structures (E-Graphs) [38, 61, 34], based on the Union-Find data-structure for equivalence classes
- Idea (sketch):
  - if \( t = s \), then merge the eq. classes of \( t \) and \( s \)
  - if \( \forall i \in 1...k, \ t_i \) and \( s_i \) pairwise belong to the same eq. classes, then merge the the eq. classes of \( f(t_1,...,t_k) \) and \( f(s_1,...,s_k) \)
  - if \( t \neq s \) and \( t \) and \( s \) belong to the same eq. class, then conflict

Example borrowed from [38].
Efficient SMT solving  
Theory Solvers for theories of interest

$\mathcal{T}$-solvers for Difference logic ($\mathcal{DL}$)

- $\mathcal{DL}$ polynomial: $O(\#vars \cdot \#constraints)$
- variants of the Bellman-Ford shortest-path algorithm: a negative cycle reveals a conflict [62, 30]
- Ex:

$$\{(x_1 - x_2 \leq -1), (x_1 - x_4 \leq -1), (x_1 - x_3 \leq -2), (x_3 - x_1 \leq -1), (x_3 - x_2 \leq -1), (x_4 - x_2 \leq 3), (x_4 - x_3 \leq 6)\}$$

---

Roberto Sebastiani ()

Satisfiability Modulo Theories  
July 16th 2011 77 / 132
\( \mathcal{T} \)-solvers for Difference logic (\( \mathcal{DL} \))

- \( \mathcal{DL} \) polynomial: \( O(\#\text{vars} \cdot \#\text{constraints}) \)
- variants of the Bellman-Ford shortest-path algorithm: a negative cycle reveals a conflict [62, 30]
- Ex:

\[
\{(x_1 - x_2 \leq -1), (x_1 - x_4 \leq -1), (x_1 - x_3 \leq -2), (x_2 - x_1 \leq 2), (x_3 - x_1 \leq -1), (x_3 - x_2 \leq -1), (x_4 - x_2 \leq 3), (x_4 - x_3 \leq 6)\}
\]

\(\xrightarrow{\text{Sat}}\)

\(\xrightarrow{\text{Unsat}}\)
\( \mathcal{T} \)-solvers for Linear arithmetic over the rationals (\( \mathcal{LA}(\mathbb{Q}) \))

- **EX:** \( \{ (s_1 - s_2 \leq 5.2), (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0), \neg (s_1 = s_0) \} \)
- \( \mathcal{LA}(\mathbb{Q}) \) polynomial variants of the simplex LP algorithm [39]
- [39] allows for detecting conflict sets & performing \( \mathcal{T} \)-propagation
- strict inequalities \( t < 0 \) rewritten as \( t + \epsilon \leq 0 \), \( \epsilon \) treated symbolically

\[
\begin{bmatrix}
B \\
x_1 \\
\vdots \\
x_i \\
\vdots \\
x_N
\end{bmatrix}
= 
\begin{bmatrix}
\ldots A_{1j} \ldots \\
\vdots \\
A_{i1} \ldots A_{ij} \ldots A_{iM}
\end{bmatrix}
\begin{bmatrix}
\mathcal{N} \\
x_{N+1} \\
\vdots \\
x_j \\
\vdots \\
x_{N+M}
\end{bmatrix};
\]

Invariant: \( \beta(x_j) \in [l_j, u_j] \ \forall x_j \in \mathcal{N} \)
Remark: infinite precision arithmetic

In order to avoid incorrect results due to numerical errors and to overflows, all $T$-solvers for $\mathcal{LA}(\mathbb{Q})$, $\mathcal{LA}(\mathbb{Z})$ and their subtheories which are based on numerical algorithms must be implemented on top of infinite-precision-arithmetic software packages.
Efficient SMT solving  
Theory Solvers for theories of interest

\[\mathcal{T}\]-solvers for Linear arithmetic over the integers \((\mathcal{L}A(\mathbb{Z}))\)

- EX: \(\{(x := x_l + 2^{16}x_h), (x \geq 0), (x \leq 2^{16} - 1)\}\)
- \(\mathcal{L}A(\mathbb{Z})\) NP-complete
- combination of many techniques: simplex, branch&bound, cutting planes, ... [39, 46]

Figure courtesy of A. Griggio [46]
\( \mathcal{T} \)-solvers for Arrays (\( \mathcal{AR} \))

- \textbf{EX:} \((\text{write}(A, i, v) = \text{write}(B, i, w)) \land \neg(v = w)\)
- NP-complete
- congruence closure (\( \mathcal{EU} \mathcal{F} \)) plus on-the-fly instantiation of array’s axioms:

\[
\forall a. \forall i. \forall e. \ (\text{read}(\text{write}(a, i, e), i) = e), \\
\forall a. \forall i. \forall j. \forall e. \ ((i \neq j) \rightarrow \text{read}(\text{write}(a, i, e), j) = \text{read}(a, j)),
\]

\[
\forall a. \forall b. \ (\forall i. (\text{read}(a, i) = \text{read}(b, i)) \rightarrow (a = b)).
\]

- \textbf{EX:}

\textit{Input}: \((\text{write}(A, i, v) = \text{write}(B, i, w)) \land \neg(v = w)\)

\textit{inst. (1)}: \((\text{read}(\text{write}(A, i, v), i) = v)\)

\((\text{read}(\text{write}(B, i, w), i) = w)\)

\(\models \mathcal{EU} \mathcal{F} \) \(\models \neg(v = w)\)

\(\models \mathcal{EU} \mathcal{F} \) \(\models \mathcal{EU} \mathcal{F} \)

\(\models \) \(\bot\)
Efficient SMT solving  
Theory Solvers for theories of interest

\( \mathcal{T} \)-solvers for Bit vectors (\( \mathcal{BV} \))

- EX: \( \{ (x_{[16]}[15 : 0] = (y_{[16]}[15 : 8] :: z_{[16]}[7 : 0])) \ll w_{[16]}[3 : 0]), \ldots \} \)
- NP-complete
- involve complex word-level operations: word partition/concat, modulo-\( 2^N \) arithmetic, shifts, bitwise-operations, multiplexers, ...
- combination of rewriting & simplification techniques with either:
  - final encoding into SAT [22, 42, 18, 41]
  - final encoding into \( \mathcal{LA}(\mathbb{Z}) \) [16, 19]

---

**Diagram**

- **Preprocessor**
  - Bool/word1 encoding
  - Control paths extraction
  - Unconstrained variables
  - Frontier propagation
  - ITE expansion
  - Formula Normalizer

- **Solver**
  - Concat. Elimination (match)
  - Variable elimination
  - Concat. elimination (no match)
  - Deduction rules

- **Literal Normalization**
  - Bit-mask Elimination
  - Selection Propagation

Example borrowed from [19]
Outline

1. Motivations and goals
2. Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities
3. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories
4. Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization
5. Conclusions & current research directions
SMT for combined theories: \( SMT(\bigcup_i T_i) \)

**Problem:** Many problems can be expressed as SMT problems only in combination of theories \( \bigcup_i T_i \) — \( SMT(\bigcup_i T_i) \)

\[
\begin{align*}
\mathcal{L}_{A}(\mathbb{Z}) : & \quad (GE_{01} \iff (v_0 \geq v_1)) \land (LE_{01} \iff (v_0 \leq v_1)) \land \\
\mathcal{EUF} : & \quad (v_3 = h(v_0)) \land (v_4 = h(v_1)) \land \\
\mathcal{L}_{A}(\mathbb{Z}) : & \quad (v_2 = v_3 - v_4) \land (RESET_5 \rightarrow (v_5 = 0)) \land \\
\mathcal{EUF} \text{ or } \mathcal{L}_{A}(\mathbb{Z}) : & \quad (\neg RESET_5 \rightarrow (v_5 = v_8)) \land \\
\mathcal{EUF} : & \quad (v_6 = f(v_2)) \land (v_7 = f(v_5)) \land \\
\mathcal{EUF} \text{ or } \mathcal{L}_{A}(\mathbb{Z}) : & \quad (EQ_{67} \iff (v_6 = v_7)) \land \ldots 
\end{align*}
\]
Efficient SMT solving  SMT for combinations of theories

\( \text{SMT}(\bigcup_i T_i) \) via “classic” Nelson-Oppen

Main idea

Combine two or more \( T_i \)-solvers into one \( (\bigcup_i T_i) \)-solver via \textit{Nelson-Oppen/Shostak (N.O.) combination procedure} \([60, 71]\]

- based on the deduction and exchange of equalities between shared variables/terms (\textit{interface equalities, } \( e_{ij} \)s)

- important improvements and evolutions \([68, 7, 38]\]

- drawbacks \([20, 21]\):
  - require (possibly expensive) deduction capabilities from \( T_i \)-solvers
  - [with non-convex theories] case-splits forced by the deduction of disjunctions of \( e_{ij} \)’s
  - generate (typically long) \( (\bigcup_i T_i) \)-lemmas, without interface equalities \( \Rightarrow \) no backjumping & learning from \( e_{ij} \)-reasoning
N.O.: example

\[ \begin{align*}
\mathcal{EUF} : & \quad (v_3 = h(v_0)) \land (v_4 = h(v_1)) \land (v_6 = f(v_2)) \land (v_7 = f(v_5)) \land \\
\mathcal{LA}(Q) : & \quad (v_0 \geq v_1) \land (v_0 \leq v_1) \land (v_2 = v_3 - v_4) \land (\text{RESET}_5 \rightarrow (v_5 = 0)) \land \\
\text{Both} : & \quad (\neg \text{RESET}_5 \rightarrow (v_5 = v_8)) \land \neg (v_6 = v_7).
\end{align*} \]
N.O.: example (cont.)

Let's consider the following example:

\[
\begin{align*}
\mathcal{EUF} & : v_3 = h(v_0) \\
v_4 & = h(v_1) \\
v_6 & = f(v_2) \\
v_7 & = f(v_5) \\
\neg(v_6 = v_7) & \\
v_0 & \geq v_1 \\
v_0 & \leq v_1 \\
v_2 & = v_3 - v_4 \\
v_5 & = 0 \\
\langle e_{ij}\text{-deduction} \rangle & \\
v_3 & = v_4 \\
v_2 & = v_5 \\
\end{align*}
\]

\[
\begin{align*}
\mathcal{LA}(\mathbb{Q}) & : v_2 = v_3 - v_4 \\
v_5 & = 0 \\
\langle e_{ij}\text{-deduction} \rangle & \\
v_3 & = v_4 \\
\end{align*}
\]

\[
\begin{align*}
\mathcal{EUF} & : (v_6 = f(v_2)) \land (v_7 = f(v_5)) \land \neg(v_6 = v_7) \land (v_2 = v_3 - v_4) \land (v_5 = 0) \land (v_3 = h(v_0)) \land (v_4 = h(v_1)) \land (v_0 \geq v_1) \rightarrow \bot.
\end{align*}
\]

\[
\begin{align*}
\mathcal{LA}(\mathbb{Q}) & : (v_0 \geq v_1) \land (v_0 \leq v_1) \rightarrow (v_0 = v_1)
\end{align*}
\]
N.O.: example (non-convex theory)

\[\mu L(A)\]
\[
\begin{align*}
\nu_1 &\geq 0 & \nu_5 &= \nu_4 - 1 \\
\nu_1 &\leq 1 & \nu_3 &= 0 \\
\nu_2 &\geq \nu_6 & \nu_4 &= 1 \\
\nu_2 &\leq \nu_6 + 1
\end{align*}
\]

\[
\langle e_{ij}\text{-deduction} \rangle
\]
\[
\nu_1 = \nu_3 \lor \nu_1 = \nu_4
\]
\[
\nu_5 = \nu_6
\]
\[
\langle e_{ij}\text{-deduction} \rangle
\]
\[
\nu_2 = \nu_3 \lor \nu_2 = \nu_4
\]

\[\mu EUF\]
\[
\begin{align*}
\neg (f(\nu_1) = f(\nu_2)) \\
\neg (f(\nu_2) = f(\nu_4)) \\
f(\nu_3) &= \nu_5 \\
f(\nu_1) &= \nu_6
\end{align*}
\]

\[
\langle e_{ij}\text{-deduction} \rangle
\]
\[
\nu_1 = \nu_3 \quad \nu_1 = \nu_4
\]
\[
\langle e_{ij}\text{-deduction} \rangle
\]
\[
\nu_5 = \nu_6
\]
\[
\langle e_{ij}\text{-deduction} \rangle
\]
\[
\nu_2 = \nu_3 \quad \nu_2 = \nu_4
\]
\[
\bot \\
\bot
\]

SAT!
Efficient SMT solving

SMT for combinations of theories

SMT\( (\bigcup_i T_i) \) via Delayed Theory Combination (DTC)

Main idea

Delegate to the CDCL SAT solver part/most of the (possibly very expensive) reasoning effort on interface equalities previously due to the \( T_i \)-solvers (\( e_{ij} \)-deduction, case-split). [12, 13, 21]

- based on Boolean reasoning on interface equalities via CDCL (plus \( T \)-propagation)
- important improvements and evolutions [36, 9]
- feature wrt N.O. [20, 21]
  - do not require (possibly expensive) deduction capabilities from \( T_i \)-solvers
  - with non-convex theories, case-splits on \( e_{ij} \)’s handled by SAT
  - generate \( T_i \)-lemmas with interface equalities
    \[ \implies \] backjumping & learning from \( e_{ij} \)-reasoning
Efficient SMT solving
SMT for combinations of theories

DTC: example

\[ \mathcal{EUF} : \quad (v_3 = h(v_0)) \land (v_4 = h(v_1)) \land (v_6 = f(v_2)) \land (v_7 = f(v_5)) \land \]

\[ \mathcal{LA}(Q) : \quad (v_0 \geq v_1) \land (v_0 \leq v_1) \land (v_2 = v_3 - v_4) \land (\text{RESET}_5 \rightarrow (v_5 = 0)) \land \]

Both : \quad (\neg \text{RESET}_5 \rightarrow (v_5 = v_8)) \land \neg (v_6 = v_7).

Search for an assignment \( \mu \) propositionally satisfying \( \varphi \)

Search on \( e_{ij} \)'s:
check the \( T_1 \cup T_2 \)-satisfiability of \( \mu \)

\[ \mu_{\mathcal{EUF}} : \quad \{(v_3 = h(v_0)), (v_4 = h(v_1)), (v_6 = f(v_2)), (v_7 = f(v_5))\} \]

\[ \mu_{\mathcal{LA}(Q)} : \quad \{(v_0 \geq v_1), (v_0 \leq v_1), (v_2 = v_3 - v_4)\} \]

C\(_{01}\) : \( \mu'_{\mathcal{LA}(Q)} \rightarrow (v_0 = v_1) \)
C\(_{34}\) : \( \mu'_{\mathcal{EUF}} \land (v_0 = v_1) \rightarrow (v_3 = v_4) \)
C\(_{25}\) : \( \mu''_{\mathcal{LA}(Q)} \land (v_5 = 0) \land (v_3 = v_4) \rightarrow (v_2 = v_5) \)
C\(_{67}\) : \( \mu''_{\mathcal{EUF}} \land (v_2 = v_5) \rightarrow (v_6 = v_7) \)
DTC: example (non-convex theory)

\[\neg(f(v_1) = f(v_2))\]
\[\neg(f(v_2) = f(v_4))\]
\[f(v_3) = v_5\]
\[f(v_1) = v_6\]
\[\neg(v_1 = v_4)\]
\[\neg(v_1 = v_3)\]
\[\neg(v_5 = v_6)\]
\[\neg(v_2 = v_4)\]
\[\neg(v_2 = v_3)\]

**\(\mu_{EUF}\):**

- \(\neg(v_1 = v_4)\)
- \(\neg(v_1 = v_3)\)
- \(\neg(v_5 = v_6)\)
- \(\neg(v_2 = v_4)\)
- \(\neg(v_2 = v_3)\)

**\(\mu_{LA(Z)}\):**

- \(v_1 = v_3\)
- \(v_5 = v_6\)
- \(v_2 = v_4\)

**\(C_{13}\):**

\[(\mu'_{LA(Z)}) \rightarrow ((v_1 = v_3) \lor (v_1 = v_4))\]

**\(C_{56}\):**

\[(\mu'_{EUF}) \land (v_1 = v_3) \rightarrow (v_5 = v_6)\]

**\(C_{23}\):**

\[(\mu''_{LA(Z)}) \land (v_5 = v_6) \rightarrow ((v_2 = v_3) \lor (v_2 = v_4))\]

**\(C_{24}\):**

\[(\mu''_{EUF}) \land (v_1 = v_3) \land (v_2 = v_3) \rightarrow \bot\]

**\(C_{14}\):**

\[(\mu'''_{EUF}) \land (v_1 = v_3) \land (v_2 = v_4) \rightarrow \bot\]
Outline

1 Motivations and goals

2 Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities

3 Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories

4 Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization

5 Conclusions & current research directions
Advanced SMT functionalities (very important in FV):

- Building *proofs of* $T$-unsatisfiability
- Extracting $T$-unsatisfiable Cores
- Computing *Craig interpolants*
- Performing *All-SMT and Predicate Abstraction*
- Deciding/optimizing *SMT problems with costs*
Outline

1. Motivations and goals
2. Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities
3. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories
4. Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization
5. Conclusions & current research directions
Building (Resolution) Proofs of \( \mathcal{T} \)-Unsatisfiability

Resolution proof of \( \mathcal{T} \)-unsatisfiability

Very similar to building proofs with plain SAT:

- resolution proofs whose leaves are original clauses and \( \mathcal{T} \)-lemmas returned by the \( \mathcal{T} \)-solver (i.e., \( \mathcal{T} \)-conflict and \( \mathcal{T} \)-deduction clauses)
- built by backward traversal of implication graphs, as in CDCL SAT
- Sub-proofs of \( \mathcal{T} \)-lemmas can be built in some \( \mathcal{T} \)-specific deduction framework if requested

Important for:

- certifying \( \mathcal{T} \)-unsatisfiability results
- computing unsatisfiable cores
- computing interpolants
Building Proofs of $\mathcal{T}$-Unsatisfiability: example

\((x = 0 \lor \neg(x = 1) \lor A_1) \land (x = 0 \lor x = 1 \lor A_2) \land (\neg(x = 0) \lor x = 1 \lor A_2) \land \\
(\neg A_2 \lor y = 1) \land (\neg A_1 \lor x + y > 3) \land (y < 0) \land (A_2 \lor x - y = 4) \land (y = 2 \lor \neg A_1) \land (x \geq 0),\)

relevant original clauses, irrelevant original clauses, $\mathcal{T}$-lemmas
Extraction of $\mathcal{T}$-unsatisfiable cores

The problem

Given a $\mathcal{T}$-unsatisfiable set of clauses, extract from it a (possibly small/minimal/minimum) $\mathcal{T}$-unsatisfiable subset ($\mathcal{T}$-unsatisfiable core)

- wide literature in SAT
- Some implementations, very few literature for SMT [25, 53]
- We recognize three approaches:
  - **Proof-based** approach (CVClite, MathSAT):
    byproduct of finding a resolution proof
  - **Assumption-based** approach (Yices):
    use extra variables labeling clauses, as in the plain Boolean case
  - **Lemma-Lifting** approach [25]:
    use an external (possibly-optimized) Boolean unsat-core extractor
The proof-based approach to $\mathcal{T}$-unsat cores

Idea (adapted from [78])

Unsatisfiable core of $\varphi$:

- in SAT: the set of leaf clauses of a resolution proof of unsatisfiability of $\varphi$
- in SMT($\mathcal{T}$): the set of leaf clauses of a resolution proof of $\mathcal{T}$-unsatisfiability of $\varphi$, minus the $\mathcal{T}$-lemmas
Beyond Solving: advanced SMT functionalities

Proofs and unsatisfiable cores

The proof-based approach to $\mathcal{T}$-unsat cores: example

\[ (x = 0 \lor \neg(x = 1) \lor A_1) \land (x = 0 \lor x = 1 \lor A_2) \land (\neg(x = 0) \lor x = 1 \lor A_2) \land \\
(\neg A_2 \lor y = 1) \land (\neg A_1 \lor x + y > 3) \land (y < 0) \land (A_2 \lor x - y = 4) \land (y = 2 \lor \neg A_1) \land (x \geq 0), \]

Roberto Sebastiani

Satisfiability Modulo Theories

July 16th 2011
The assumption-based approach to $\mathcal{T}$-unsat cores

Let $\varphi$ be $\bigwedge_{i=1}^{n} C_i$ s.t. $\varphi$ inconsistent.

Idea (adapted from [55])

1. each clause $C_i$ in $\varphi$ is substituted by $\neg S_i \lor C_i$, s.t. $S_i$ fresh “selector” variable
2. the resulting formula is checked for satisfiability under the assumption of all $S_i$’s
3. final conflict clause at dec. level 0: $\bigvee_{j} \neg S_j$ $\implies \{C_j\}_j$ is the unsat core

- extends straightforwardly to SMT($\mathcal{T}$).
The assumption-based approach to \( \mathcal{T} \)-unsat cores: Example

\[
(S_1 \rightarrow (x = 0 \lor \neg(x = 1) \lor A_1)) \land (S_2 \rightarrow (x = 0 \lor x = 1 \lor A_2)) \land \\
(S_3 \rightarrow (\neg(x = 0) \lor x = 1 \lor A_2)) \land (S_4 \rightarrow (\neg A_2 \lor y = 1)) \land \\
(S_5 \rightarrow (\neg A_1 \lor x + y > 3)) \land (S_6 \rightarrow y < 0) \land \\
(S_7 \rightarrow (A_2 \lor x - y = 4)) \land (S_8 \rightarrow (y = 2 \lor \neg A_1)) \land (S_9 \rightarrow x \geq 0)
\]

Conflict analysis (Yices 1.0.6) returns:

\[
\neg S_1 \lor \neg S_2 \lor \neg S_3 \lor \neg S_4 \lor \neg S_6 \lor \neg S_7 \lor \neg S_8,
\]

corresponding to the unsat core in red.
The lemma-lifting approach to $\mathcal{T}$-unsat cores

**Idea [25, 29]**

(i) The $\mathcal{T}$-lemmas $D_i$ are valid in $\mathcal{T}$

(ii) The conjunction of $\varphi$ with all the $\mathcal{T}$-lemmas $D_1, \ldots, D_k$ is propositionally unsatisfiable: $\mathcal{T}2B(\varphi \land \bigwedge_{i=1}^{n} D_i) \models \bot$.

.interfaces with an external Boolean Unsat-core Extractor

$\implies$ benefits for free of all state-of-the-art size-reduction techniques

Roberto Sebastiani ()
Satisfiability Modulo Theories
July 16th 2011
102 / 132
The lemma-lifting approach to \( \mathcal{T} \)-unsat cores: example

\[
(x = 0 \lor \neg(x = 1) \lor A_1) \land (x = 0 \lor x = 1 \lor A_2) \land (\neg(x = 0) \lor x = 1 \lor A_2) \land \\
(\neg A_2 \lor y = 1) \land (\neg A_1 \lor x + y > 3) \land (y < 0) \land (A_2 \lor x - y = 4) \land (y = 2 \lor \neg A_1) \land (x \geq 0),
\]

1. The SMT solver generates the following set of \( \mathcal{L}A(\mathbb{Z}) \)-lemmas:

\[
\{(\neg(x = 1) \lor \neg(x = 0)), (\neg(y = 2) \lor \neg(y < 0)), (\neg(y = 1) \lor \neg(y < 0))\}.
\]

2. The following formula is passed to the external Boolean core extractor

\[
(B_0 \lor \neg B_1 \lor A_1) \land (B_0 \lor B_1 \lor A_2) \land (\neg B_0 \lor B_1 \lor A_2) \land \\
(\neg A_2 \lor B_2) \land (\neg A_1 \lor B_3) \land B_4 \land (A_2 \lor B_5) \land (B_6 \lor \neg A_1) \land B_7 \land \\
(\neg B_1 \lor \neg B_0) \land (\neg B_6 \lor \neg B_4) \land (\neg B_2 \lor \neg B_4)
\]

which returns the unsat core in red.

3. The unsat-core is mapped back, the three \( \mathcal{T} \)-lemmas are removed \( \implies \) the final \( \mathcal{T} \)-unsat core (in red above).
Computing (Craig) Interpolants in SMT

Craig Interpolant

Given an ordered pair \((A, B)\) of formulas such that \(A \land B \models_{T} \bot\), a Craig interpolant is a formula \(I\) s.t.:

a) \(A \models_{T} I\),

b) \(I \land B \models_{T} \bot\),

c) \(I \preceq A\) and \(I \preceq B\).

“\(I \preceq A\)” meaning that all uninterpreted (in \(T\)) symbols in \(I\) occur in \(A\).

Very important in many FV applications

A few works presented for various theories:

- \(EUF\) \([57, 69]\), \(DL\) \([26, 28]\), \(UTVPI\) \([27, 28]\), \(LA(\mathbb{Q})\) \([57, 69, 26, 28]\), \(LA(\mathbb{Z})\) \([50, 15, 47]\), \(BV\) \([51]\), ...
A General Algorithm

Algorithm: Interpolant generation for SMT($T$) [67, 57]

(i) Generate a resolution proof of $T$-unsatisfiability $\mathcal{P}$ for $A \land B$.

(ii) ...

(iii) For every original leaf clause $C$ in $\mathcal{P}$, set $I_C \overset{\text{def}}{=} C \downarrow B$ if $C \in A$, and $I_C \overset{\text{def}}{=} \top$ if $C \in B$.

(iv) For every inner node $C$ of $\mathcal{P}$ obtained by resolution from $C_1 \overset{\text{def}}{=} p \lor \phi_1$ and $C_2 \overset{\text{def}}{=} \neg p \lor \phi_2$, set $I_C \overset{\text{def}}{=} I_{C_1} \lor I_{C_2}$ if $p$ does not occur in $B$, and $I_C \overset{\text{def}}{=} I_{C_1} \land I_{C_2}$ otherwise.

(v) Output $I_{\bot}$ as an interpolant for $(A, B)$.

"$\eta \setminus B$" [resp. "$\eta \downarrow B$"] is the set of literals in $\eta$ whose atoms do not [resp. do] occur in $B$.

- row 2. only place where $T$ comes in to play

$\Longrightarrow$ Reduced to the problem of finding an interpolant for two sets of $T$-literals (Boolean and $T$-specific component decoupled)
A General Algorithm

Algorithm: Interpolant generation for SMT(\(\mathcal{T}\)) [67, 57]

(i) Generate a resolution proof of \(\mathcal{T}\)-unsatisfiability \(\mathcal{P}\) for \(A \land B\).

(ii) Foreach \(\mathcal{T}\)-lemma \(\neg \eta\) in \(\mathcal{P}\), generate an interpolant \(I_{\neg \eta}\) for \((\eta \setminus B, \eta \downarrow B)\).

(iii) For every original leaf clause \(C\) in \(\mathcal{P}\), set \(I_C \overset{\text{def}}{=} C \downarrow B\) if \(C \in A\), and \(I_C \overset{\text{def}}{=} \top\) if \(C \in B\).

(iv) For every inner node \(C\) of \(\mathcal{P}\) obtained by resolution from \(C_1 \overset{\text{def}}{=} p \lor \phi_1\) and \(C_2 \overset{\text{def}}{=} \neg p \lor \phi_2\), set \(I_C \overset{\text{def}}{=} I_{C_1} \lor I_{C_2}\) if \(p\) does not occur in \(B\), and \(I_C \overset{\text{def}}{=} I_{C_1} \land I_{C_2}\) otherwise.

(v) Output \(I_{\bot}\) as an interpolant for \((A, B)\).

“\(\eta \setminus B\)” [resp. “\(\eta \downarrow B\)“] is the set of literals in \(\eta\) whose atoms do not [resp. do] occur in \(B\).

- row 2. only place where \(\mathcal{T}\) comes in to play

\(\Rightarrow\Rightarrow\) Reduced to the problem of finding an interpolant for two sets of \(\mathcal{T}\)-literals (Boolean and \(\mathcal{T}\)-specific component decoupled)
Computing Craig Interpolants in SMT: example

\[ A \overset{\text{def}}{=} (B_1 \lor (0 \leq x_1 - 3x_2 + 1)) \land (0 \leq x_1 + x_2) \land (\neg B_2 \lor \neg (0 \leq x_1 + x_2)) \]

\[ B \overset{\text{def}}{=} (\neg (0 \leq x_3 - 2x_1 - 3) \lor (0 \leq 1 - 2x_3)) \land (\neg B_1 \lor B_2) \land (B_1 \lor (0 \leq x_3 - 2x_1 - 3)) \]

original proof

\[
\begin{align*}
\neg (0 \leq x_1 - 3x_2 + 1) & \lor \neg (0 \leq x_1 + x_2) \\
\neg (0 \leq x_3 - 2x_1 - 3) & \lor (0 \leq 1 - 2x_3) \\
\neg (0 \leq x_1 - 3x_2 + 1) & \lor \neg (0 \leq x_1 + x_2) \\
\neg (0 \leq x_3 - 2x_1 - 3) & \lor (0 \leq x_3 - 2x_1 - 3) \\
\neg (0 \leq x_1 + x_2) & \lor B_1 \\
\neg B_1 & \lor (0 \leq x_3 - 2x_1 - 3) \\
\neg B_2 & \lor \neg (0 \leq x_1 + x_2) \\
(0 \leq x_1 + x_2) & \lor \neg (0 \leq x_1 + x_2) \\
\bot & \\
\end{align*}
\]

interpolant proof

\[
\begin{align*}
(0 \leq 4x_1 + 1) & \\
T & \\
(0 \leq 4x_1 + 1) & \\
T & \\
(0 \leq 4x_1 + 1) & \\
B_1 & \\
B_1 & \\
B_1 & \\
\bot & \\
B_1 & \\
\bot & \\
\neg B_2 & \\
(\neg B_1 \lor (0 \leq 4x_1 + 1)) & \land \neg B_2
\end{align*}
\]
Example: interpolation algorithms for difference logic

- An inference-based algorithm [57]

\[
\frac{(0 \leq x_1 - x_2 + 1) \quad (0 \leq x_2 - x_3)}{\text{COMB} \quad (0 \leq x_1 - x_3 + 1) \quad (0 \leq x_4 - x_5 - 1)} \quad \frac{\text{COMB} \quad (0 \leq x_1 - x_3 + x_4 - x_5)}{(0 \leq 0)(0 \leq 0)}
\]

\[\implies\text{Interpolant: } (0 \leq x_1 - x_3 + x_4 - x_5) \text{ (not in } \mathcal{DL}, \text{ and weaker).}\]

- A graph-based algorithm [26, 28]

\[
A \overset{\text{def}}{=} \{(0 \leq x_1 - x_2 + 1), (0 \leq x_2 - x_3), (0 \leq x_4 - x_5 - 1)\}
\]

\[
B \overset{\text{def}}{=} \{(0 \leq x_5 - x_1), (0 \leq x_3 - x_4 - 1)\}.
\]

\[\implies\text{Interpolant: } (0 \leq x_1 - x_3 + 1) \land (0 \leq x_4 - x_5 - 1) \text{ (still in } \mathcal{DL})\]
Outline

1 Motivations and goals
2 Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities
3 Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories
4 Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization
5 Conclusions & current research directions
**All-SAT/All-SMT**

- **All-SAT**: enumerate all truth assignments satisfying $\varphi$
- **All-SMT**: enumerate all $T$-satisfiable truth assignments propositionally satisfying $\varphi$
- **All-SMT over an “important” subset of atoms $P \overset{\text{def}}{=} \{P_i\}_i$**: enumerate all assignments over $P$ which can be extended to $T$-satisfiable truth assignments propositionally satisfying $\varphi$  
  \[\Rightarrow\] *can compute predicate abstraction*

**Algorithms:**

- **BCLT** [52]
  each time a $T$-satisfiable assignment $\{l_1, ..., l_n\}$ is found, perform conflict-driven backjumping as if the restricted clause $(\lor_i \neg l_i) \downarrow P$ belonged to the clause set
- **MathSAT/NuSMV** [23]
  As above, plus the Boolean search of the SMT solver is driven by an OBDD.
Predicate Abstraction

Predicate abstraction

If \( \varphi(\mathbf{v}) \) is a SMT formula over the domain variables \( \mathbf{v} \overset{\text{def}}{=} \{v_j\}_j \), \( \{\gamma_i\}_i \) is a set of “relevant” predicates over \( \mathbf{v} \), and \( \mathbf{P} \overset{\text{def}}{=} \{P_i\}_i \) a set of Boolean labels, then:

\[
\operatorname{PredAbs}_{\mathbf{P}}(\varphi) \overset{\text{def}}{=} \exists \mathbf{v}. (\varphi(\mathbf{v}) \land \bigwedge_i P_i \leftrightarrow \gamma_i(\mathbf{v}))
\]

\[
= \bigvee \left\{ \mu \mid \mu \text{ truth assignment on } \mathbf{P} \text{ s.t. } \mu \land \varphi \land \bigwedge_i (P_i \leftrightarrow \gamma_i) \text{ is } \mathcal{T} \text{-satisfiable} \right\}
\]

- Projection of \( \varphi \) over (the Boolean abstraction of) the set \( \{\gamma_i\}_i \).
- Essential step in FV: extracts finite-state abstractions from an infinite state space.
Predicate Abstraction: example

\[ \varphi \overset{\text{def}}{=} (v_1 + v_2 > 12) \]
\[ \gamma_1 \overset{\text{def}}{=} (v_1 + v_2 = 2) \]
\[ \gamma_2 \overset{\text{def}}{=} (v_1 - v_2 < 10) \]

\[ \Downarrow \]

\[ \text{PreAbs}(\varphi)\{P_1, P_2\} \overset{\text{def}}{=} \exists v_1 v_2 . \left( (v_1 + v_2 > 12) \land (P_1 \iff v_1 + v_2 = 2) \land (P_2 \iff v_1 - v_2 < 10) \right) \]
\[ = (\neg P_1 \land \neg P_2) \lor (\neg P_1 \land P_2) \]
\[ = \neg P_1. \]
Outline

1 Motivations and goals

2 Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities

3 Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories

4 Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization

5 Conclusions & current research directions
SMT cost-minimization problems

The problem

SMT(\(\mathcal{T}\)) problem \(\varphi\) for some \(\mathcal{T}\), augmented with cost functions:

\[
\text{cost}^i = \sum_{j=1}^{N^i} \text{ite}(P^{ij}, c_1^{ij}, c_2^{ij}), \text{ s.t. } \text{cost}^i \in [l^i, u^i], c_{\{1,2}\} > 0
\]

- **Decision problem**: is there a model complying with cost ranges?
- **Optimization problem**: find model minimizing some \(\text{cost}^i\).

allows for encoding MaxSAT/MaxSMT and PseudoBoolean

Proposed solution: [63, 24]

- \(\text{SMT}(\mathcal{T} \cup \mathcal{C})\), \(\mathcal{C}\) is an ad-hoc “theory of costs”
- a specialized very-fast theory-solver for \(\mathcal{C}\) added to MathSAT
  - very fast & aggressive search pruning and theory-propagation
- cost minimization handled by linear or binary search
SMT cost-minimization problems

The problem

SMT($\mathcal{T}$) problem $\varphi$ for some $\mathcal{T}$, augmented with cost functions:

$$
cost^i = \sum_{j=1}^{N^i} \text{ite}(P^{ij}, c_1^{ij}, c_2^{ij}), \text{ s.t. } \text{cost}^i \in (l^i, u^i], \ c_{\{1,2}\} > 0
$$

- **Decision problem**: is there a model complying with cost ranges?
- **Optimization problem**: find model minimizing some $\text{cost}^i$.

- allows for encoding MaxSAT/MaxSMT and PseudoBoolean

Proposed solution: [63, 24]

- **SMT($\mathcal{T} \cup \mathcal{C}$)**, $\mathcal{C}$ is an ad-hoc “theory of costs”
- a specialized very-fast theory-solver for $\mathcal{C}$ added to MathSAT
  - very fast & aggressive search pruning and theory-propagation
- cost minimization handled by linear or binary search
SMT(\(\mathcal{T} \cup \mathcal{C}\)): main ideas

- A “theory of costs” \(\mathcal{C}\):
  - Cost variables \(cost^i\)
  - “bound cost” \(BC(cost^i, k)\): “\(cost^i \leq k\)”
  - “incur cost” \(IC(cost^i, j, k^i_j)\): “the \(j\)th addend of \(cost^i := k^i_j\)”
  - “\(cost^i = \sum_{j=1}^{N^i} \text{ite}(P^i_j, k^i_j, 0)\), s.t. \(cost^i \in (l^i, u^i]\)” encoded as
    \[-BC(cost^i, l^i) \land BC(cost^i, u^i) \land \land_{j=1}^{N^i}(P^i_j \leftrightarrow IC(cost^i, j, k^i_j))\]

- very-fast theory solver: \(\mathcal{C}\)-solver
  1. \(IC(cost^i, j, k^i_j) = \top \implies cost^i = cost^i + k^i_j\)
  2. \(cost^i > ub^i \implies \text{conflict}\)
  3. \(cost^i + \{\text{total cost of all unassigned IC's}\} \leq lb^i \implies \text{conflict}\)
  4. \(IC(cost^i, j, k^i_j) = \top \) causes 2. \(\implies \mathcal{C}\)-propagate \(\neg IC(cost^i, j, k^i_j)\)
  5. \(IC(cost^i, j, k^i_j) = \bot \) causes 3. \(\implies \mathcal{C}\)-propagate \(IC(cost^i, j, k^i_j)\)

- no symbol shared with \(\mathcal{T}\)
  \(\implies\) independent theory solvers for \(\mathcal{T}\) and \(\mathcal{C}\)
Ongoing work: SMT with linear cost functions

The problem

SMT($\mathcal{L} \mathcal{A}(\mathbb{Q}) \cup \bigcup_k \mathcal{T}_k$) problem $\varphi$ for some $\mathcal{T}_k$’s, augmented with cost functions:

\[
\text{cost}_i = \sum_{j=1}^{N_i} a_i x_i, \text{ s.t. } \text{cost}_i \in (l^i, u^i], \ x_i \in \mathbb{Q}
\]

- **Decision problem**: standard SMT($\mathcal{L} \mathcal{A}(\mathbb{Q}) \cup \bigcup_k \mathcal{T}_k$)
- **Optimization problem**: find model minimizing some cost$_i$.

Optimization along two orthogonal directions: $\mathit{BOOL}$ and $\mathbb{Q}$

Allows for encoding **generalized linear disjunctive programming**

Proposed solution:

- SMT($\mathcal{L} \mathcal{A}(\mathbb{Q}) \cup \bigcup_k \mathcal{T}_k$) augmented with a LP optimization routine
  - very fast & aggressive search pruning and theory-propagation
- cost minimization (branch&bound or binary search) **embedded** inside the CDCL search
Outline

1. Motivations and goals
2. Background: Modern SAT solving
   - Basics
   - Modern CDCL SAT solving
   - Other SAT functionalities
3. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for theories of interest
   - SMT for combinations of theories
4. Beyond Solving: advanced SMT functionalities
   - Proofs and unsatisfiable cores
   - Interpolants
   - All-SMT & Predicate Abstraction
   - SMT with cost optimization
5. Conclusions & current research directions
Conclusions

- SMT very popular, due to successful application in many domains
- Combines techniques from SAT, ATP and operational research
- Not only satisfiability, but also advanced functionalities
Conclusions & current research directions

Open/ongoing research directions

- **Solving:**
  - improve efficiency (e.g. $\mathcal{BV}$, $\mathcal{AR}$, $\mathcal{LA}(\mathbb{Z})$ & their combinations)
    - "a never-ending fight against the search-space explosion problem [E. Clarke, Turing-award winner 2007]"
  - develop efficient solvers for other theories ($\mathcal{NLA}(\mathbb{Q})$, $\mathcal{NLA}(\mathbb{Z})$)
  - develop new theories & solvers (e.g., floating-point arithmetic)
  - ... 

- **Functionalities**
  - **Interpolation** in some theories ($\mathcal{LA}(\mathbb{Z})$, $\mathcal{BV}$) still very challenging
  - **Predicate abstraction (AllSMT)** still a bottleneck in SMT-based FV
  - **SMT with costs/optimization** still in very early stage
  - ... 

- **Combination of SMT solvers and ATP** (SMT with quantifiers)
- Integration & customization of SMT solvers with (FV) tools
- See also [64]
Links I

- a 20-hour PhD course on SAT & SMT:
  http://disi.unitn.it/~rseba/DIDATTICA/SAT_BASED10/

- survey papers:
  http://disi.unitn.it/~rseba/publist.html

  - Roberto Sebastiani: "Lazy Satisfiability Modulo Theories".
    Journal on Satisfiability, Boolean Modeling and Computation, JSAT.

  - Clark Barrett, Roberto Sebastiani, Sanjit Seshia, Cesare Tinelli
    "Satisfiability Modulo Theories". Part II, Chapter 26, The Handbook
    of Satisfiability. 2009. ©IOS press.

- web links:

  - The SMT library SMT-LIB:
    http://goedel.cs.uiowa.edu/smtlib/

  - The SMT Competition SMT-COMP: http://www.smtcomp.org/

  - The 2011 MIT SAT/SMT School
    http://people.csail.mit.edu/vganesh/summerschool/
References I

Simplifying OBDDs in Decidable Theories.  

SAT-based procedures for temporal reasoning.  

A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions.  


Verifying Industrial Hybrid Systems with MathSAT.  

SAT-Based Bounded Model Checking for Timed Systems.  

A Generalization of Shostak’s Method for Combining Decision Procedures.  
In Proc. FROCOs’02, 2002.

Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT.  
In 14th International Conference on Computer-Aided Verification, 2002.
Splitting on Demand in SAT Modulo Theories.

Satisfiability Modulo Theories.

FDPLL - A First Order Davis-Putnam-Longeman-Loveland Procedure.

Efficient Satisfiability Modulo Theories via Delayed Theory Combination.

Efficient theory combination via boolean search.

Mathsat: Tight integration of sat and mathematical decision procedures.

An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic.

RTL-datapath verification using integer linear programming.
Lemmas on Demand for the Extensional Theory of Arrays.
*Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 6, 2009.

Boolector: An efficient smt solver for bit-vectors and arrays.

A Lazy and Layered SMT(\textit{BV}) Solver for Hard Industrial Verification Problems.


Automatic Verification of Pipelined Microprocessor Control.

Computing Predicate Abstractions by Integrating BDDs and SMT Solvers.

Satisfiability modulo the theory of costs: Foundations and applications.
A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories.

Efficient Interpolant Generation in Satisfiability Modulo Theories.

Interpolant Generation for UTVPI.

Efficient Generation of Craig Interpolants in Satisfiability Modulo Theories.

Computing small unsatisfiable cores in sat modulo theories.

Fast and Flexible Difference Logic Propagation for DPLL(T).

[31] M. Davis, G. Longemann, and D. Loveland.
A machine program for theorem proving.

A computing procedure for quantification theory.


Applying SMT in Symbolic Execution of Microcode.

[42] V. Ganesh and D. L. Dill.
A Decision Procedure for Bit-Vectors and Arrays.

DPLL(T): Fast decision procedures.
(To appear).

Building decision procedures for modal logics from propositional decision procedures - the case study of modal K.

Deciding array formulas with frugal axiom instantiation.
In *Proceedings of SMT’08/BPR’08*, pages 12–17, New York, NY, USA, 2008. ACM.

A Practical Approach to SMT(LA(Z)).

Efficient Interpolant Generation in Satisfiability Modulo Linear Integer Arithmetic.
Stochastic Local Search for SMT: Combining Theory Solvers with WalkSAT. 

FaCT and DLP. 

Efficient Craig Interpolation for Linear Diophantine (Dis)Equations and Linear Modular Equations. 

Lifting Propositional Interpolants to the Word-Level. 

SMT techniques for fast predicate abstraction. 


On Computing Minimum Unsatisfiable Cores. 

On computing minimum unsatisfiable cores. 
In SAT, 2004.
A Satisfiability Checker for Difference Logic.  

An interpolating theorem prover.  

Fully symbolic model checking of timed systems using difference decision diagrams.  

Chaff: Engineering an efficient SAT solver.  

Simplification by Cooperating Decision Procedures.  

Congruence closure with integer offsets.  

DPLL(T) with Exhaustive Theory Propagation and its Application to Difference Logic.  
On SAT Modulo Theories and Optimization Problems.

Challenges in Satisfiability Modulo Theories.

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

A Structure-preserving Clause Form Translation.

Lower bounds for resolution and cutting planes proofs and monotone computations.

Deconstructing Shostak.

Constraint Solving for Interpolation.

[70] R. Sebastiani.
Lazy Satisfiability Modulo Theories.
References X

[71] R. Shostak. 
Deciding Combinations of Theories.  

GRASP - A new Search Algorithm for Satisfiability.  
In *Proc. ICCAD’96*, 1996.

Backtracking algorithms for disjunctions of temporal constraints.  

[74] A. Stump, C. W. Barrett, and D. L. Dill.  
CVC: A Cooperating Validity Checker.  
In *Proc. CAV’02*, number 2404 in LNCS. Springer Verlag, 2002.

The LPSAT Engine & its Application to Resource Planning.  

A Library for Composite Symbolic Representation.  

Efficient conflict driven learning in boolean satisfiability solver.  

Extracting small unsatisfiable cores from unsatisfiable boolean formula.  
Disclaimer

The list of references above is by no means intended to be all-inclusive. I apologize both with the authors and with the readers for all the relevant works which are not cited here.