Proofs in Satisfiability Modulo Theories

Pascal Fontaine (Inria, Loria, U. Lorraine)

SAT/SMT Summer School 2015

Stanford

July 17, 2015
Outline

Introduction

Proofs for SAT
  Prerequisites
  SAT and proofs
  Proof formats
Introduction

Why proofs in SAT/SMT?

▶ as a debugging facility
▶ as a part of the reasoning framework (e.g. conflict clauses)
▶ to check the result for unsatisfiable/valid formulas
▶ to extract cores
▶ to compute interpolants
▶ for solver/prover cooperation
▶ for evaluation purposes (how good is the algorithm?)

---

Outline

Introduction

Proofs for SAT

Prerequisites
SAT and proofs
Proof formats
Introduction

Proofs for SAT
  Prerequisites
  SAT and proofs
  Proof formats
Outline

Introduction

Proofs for SAT
  Prerequisites
  SAT and proofs
  Proof formats
CNF: from formulas to clauses

- Boolean formulas: built with variables, ¬, ∧, ∨, ⇒, ...  
- Clause: disjunctive set of literals (ACI of ∨ used implicitly)  
- Conjunctive Normal Form (CNF): (conjunctive) set of clauses  
- Disjunctive Normal Form (DNF): (disjunctive) set of cubes

Every formula is logically equivalent to a CNF (DNF)

Remark:

- Converting to DNF, then finding one satisfiable cube is a trivial satisfiability procedure
- checking the satisfiability of a (set of) cube(s) is linear
- so DNF conversion cannot be polynomial (otherwise P = NP)
- computing DNF of formula: negation of CNF of negation of formula
- so CNF conversion cannot be efficient
**CNF: efficient computation**

Consider

\[ \varphi = (p_{1,1} \land p_{1,2} \land p_{1,3}) \lor \cdots \lor (p_{n,1} \land p_{n,2} \land p_{n,3}) \]

**Equivalent CNF (distributivity laws)**

\[ \varphi \iff \bigwedge_{i_1=1}^{3} \cdots \bigwedge_{i_n=1}^{3} (p_{1,i_1} \lor \cdots \lor p_{n,i_n}) \]

**Equisatisfiable CNF**

\[ (X_1 \lor \cdots \lor X_n) \land \bigwedge_{i=1}^{n} (X_i \iff (p_{i,1} \land p_{i,2} \land p_{i,3})) \]

where \( X_i \iff (p_{i,1} \land p_{i,2} \land p_{i,3}) \) can be represented as a conjunction of clauses (Exercise).

**Formulas can be transformed in linear time into equisatisfiable CNF**

Plaisted, Greenbaum, Tseitin: definitional, p-definitional

Doesn’t it contradict the previous slide?

---

Resolution

Resolution rule

\[
\frac{A \lor \ell \quad B \lor \overline{\ell}}{A \lor B}
\]

Antecedents: \(A \lor \ell, B \lor \overline{\ell}\)

Pivot: \(\ell\) or \(\overline{\ell}\)

Resolvent: \(A \lor B = (A \lor \ell) \diamond (B \lor \overline{\ell})\)

Resolution: complete method for propositional logic

Extensions for FOL logic, also with equality

Some proofs systems are much stronger

E.g. pigeon hole: exponential resolution proofs\(^3\)

\[\text{polynomial extended resolution proofs}^4\]

Refutation proof

- first formula is input
- subsequent formulas consequences of previous ones by application of rule (checked “easily”)
- last formula is \(\square\)

SAT/SMT proof: resolutions (chains), input formulas and tautologies


introducing new variables: does not preserve equivalence

no new variables: subformulas stand for variables

Assume $\varphi_{in} = a \land (b \lor (c \land d))$

Rules: adding tautologies of the form

- $\neg(X_1 \land \cdots \land X_n) \lor X_i$
- $\neg X_1 \lor \cdots \lor \neg X_n \lor (X_1 \land \cdots \land X_n)$
- other rules for other connectors

Proof:

- $\varphi_2 = \neg(a \land (b \lor (c \land d))) \lor a$ by resolution of $\varphi_{in}$ and $\varphi_2$
- $\varphi_3 = a$ by resolution of $\varphi_{in}$ and $\varphi_4$
- $\varphi_4 = \neg(a \land (b \lor (c \land d))) \lor (b \lor (c \land d))$
- $\varphi_5 = b \lor (c \land d)$ by resolution of $\varphi_{in}$ and $\varphi_4$
- $\varphi_6 = \neg(c \land d) \lor c$
- $\varphi_7 = \neg(c \land d) \lor d$

$\varphi_{in}, \psi$ equivalent to $\varphi_3, \varphi_5, \varphi_6, \varphi_7, \psi$
**CNF: Proofs**

- introducing new variables: does not preserve equivalence
- no new variables: subformulas stand for variables

Assume $\varphi_{in} = a \land (b \lor (c \land d))$

Rules: adding tautologies of the form

- $\neg (X_1 \land \cdots \land X_n) \lor X_i$
- $\neg X_1 \lor \cdots \lor \neg X_n \lor (X_1 \land \cdots \land X_n)$
- other rules for other connectors

Proof:

- $\varphi_2 = \neg (a \land (b \lor (c \land d))) \lor a$
- $\varphi_3 = a$ by resolution of $\varphi_{in}$ and $\varphi_2$
- $\varphi_4 = \neg (a \land (b \lor (c \land d))) \lor (b \lor (c \land d))$
- $\varphi_5 = b \lor (c \land d)$ by resolution of $\varphi_{in}$ and $\varphi_4$
- $\varphi_6 = \neg (c \land d) \lor c$
- $\varphi_7 = \neg (c \land d) \lor d$

$\varphi_{in}, \psi$ equisatisfiable to $\varphi_3, \varphi_5, \varphi_6, \varphi_7, \psi$
even with subformulas abstracted in $\varphi_is
Resolution chain  
a.k.a. hyper-resolution

Sequence of resolution  \( C_1 \diamond C_2 \diamond C_3 \diamond C_4 \)

\[ \frac{C_1}{C_2} \]
\[ \frac{C_1 \diamond C_2}{C_3} \]
\[ \frac{C_1 \diamond C_2 \diamond C_3}{C_4} \]
\[ C_1 \diamond C_2 \diamond C_3 \diamond C_4 \]

\( \diamond \) is commutative, but not associative

\[ (\bar{a} \lor \bar{b} \lor c \diamond \bar{a} \lor b) \diamond a \lor c \]
\[ \frac{\bar{a} \lor \bar{b} \lor c \quad \bar{a} \lor b}{\bar{a} \lor c} \]
\[ \frac{\bar{a} \lor b}{c} \]
\[ a \lor c \]

\[ \bar{a} \lor \bar{b} \lor c \diamond (\bar{a} \lor b \diamond a \lor c) \]
\[ \frac{\bar{a} \lor \bar{b} \lor c}{\bar{a} \lor b} \]
\[ \frac{a \lor c}{b \lor c} \]
\[ \frac{\bar{a} \lor b}{\bar{a} \lor c} \]
Resolution chain
a.k.a. hyper-resolution

Sequence of resolution \(((C_1 \diamond C_2) \diamond C_3) \diamond C_4\)

\[
\begin{array}{c}
C_1 \\ (C_1 \diamond C_2) \\ (C_1 \diamond C_2) \diamond C_3 \\ ((C_1 \diamond C_2) \diamond C_3) \diamond C_4
\end{array}
\]

\(\diamond\) is commutative, but not associative

\[
\begin{array}{c}
(\overline{a} \lor \overline{b} \lor c \diamond \overline{a} \lor b) \diamond a \lor c \\
\overline{a} \lor \overline{b} \lor c \diamond \overline{a} \lor b \\
\overline{a} \lor \overline{b} \lor c \diamond \overline{a} \lor b \\
(\overline{a} \lor b \diamond a \lor c) \\
\overline{a} \lor \overline{b} \lor c \\
\overline{a} \lor \overline{b} \lor c \\
\overline{a} \lor b \\
\overline{a} \lor \overline{b} \lor c \\
\end{array}
\]

\[
\begin{array}{c}
(\overline{a} \lor b \diamond a \lor c) \\
\overline{a} \lor \overline{b} \lor c \diamond (\overline{a} \lor b \diamond a \lor c) \\
\overline{a} \lor b \\
\overline{a} \lor \overline{b} \lor c \\
\overline{a} \lor b \diamond a \lor c \\
\overline{a} \lor \overline{b} \lor c \\
\end{array}
\]
Resolution chain
a.k.a. hyper-resolution

Sequence of resolution \(((C_1 \diamond C_2) \diamond C_3) \diamond C_4\)

\[
\begin{align*}
\frac{C_1 \quad C_2}{(C_1 \diamond C_2) \quad C_3} \\
\frac{(C_1 \diamond C_2) \quad C_3}{((C_1 \diamond C_2) \diamond C_3) \diamond C_4}
\end{align*}
\]

\(\diamond\) is commutative, but not associative. \(\diamond\) left associative

\[
\begin{align*}
(\bar{a} \vee \bar{b} \vee c \diamond \bar{a} \vee b) \diamond a \vee c \\
\frac{\bar{a} \vee \bar{b} \vee c \quad \bar{a} \vee b}{\bar{a} \vee c} \quad a \vee c \\
\frac{\bar{a} \vee \bar{b} \vee c}{c} \quad a \vee c
\end{align*}
\]

\[
\begin{align*}
\bar{a} \vee \bar{b} \vee c \diamond (\bar{a} \vee b \diamond a \vee c) \\
\frac{\bar{a} \vee b}{\bar{a} \vee c} \quad a \vee c
\end{align*}
\]

\[
\begin{align*}
\bar{a} \vee \bar{b} \vee c \diamond (\bar{a} \vee b \diamond a \vee c) \\
\frac{\bar{a} \vee b}{\bar{a} \vee c}
\end{align*}
\]
Outline

Introduction

Proofs for SAT

Prerequisites

SAT and proofs

Proof formats
SAT solving

\[ C_1 = \overline{b} \lor c, \quad C_2 = a \lor c, \quad C_3 = \overline{a} \lor b, \quad C_4 = \overline{a} \lor \overline{b}, \quad C_5 = a \lor \overline{b}, \quad C_6 = b \lor \overline{c} \]

1: `procedure SAT(C)`
2: \ `while T do`
3: \ \ `if PROPAGATE then`
4: \ \ \ `if \neg DECIDE then`
5: \ \ \ \ `return SAT`
6: \ \ \ `continue`
7: \ \ `if level = 0 then`
8: \ \ \ `return UNSAT`
9: \ `ANALYSE`
10: `BACKTRACK`

Stack:
SAT solving

\[ C_1 = \overline{b} \lor c, \ C_2 = a \lor c, \ C_3 = \overline{a} \lor b, \ C_4 = \overline{a} \lor \overline{b}, \ C_5 = a \lor \overline{b}, \ C_6 = b \lor \overline{c} \]

1: procedure SAT(C)  
2: while T do  
3: \hspace{1em} if PROPAGATE then  
4: \hspace{1em} \hspace{1em} if \ \neg \text{Decide} then  
5: \hspace{1em} \hspace{1em} \hspace{1em} return SAT  
6: \hspace{1em} \hspace{1em} continue  
7: \hspace{1em} if level = 0 then  
8: \hspace{1em} \hspace{1em} return UNSAT  
9: \hspace{1em} Analyse  
10: \hspace{1em} Backtrack

No propagation possible

Stack:
SAT solving

\[ C_1 = \overline{b} \lor c, \quad C_2 = a \lor c, \quad C_3 = \overline{a} \lor b, \quad C_4 = \overline{a} \lor \overline{b}, \quad C_5 = a \lor \overline{b}, \quad C_6 = b \lor \overline{c} \]

1: procedure SAT($C$)
2: while $\top$ do
3: if PROPAGATE then
4: if $\neg$DECIDE then
5: return SAT
6: continue
7: if level = 0 then
8: return UNSAT
9: ANALYSE
10: BACKTRACK

\[ \overline{c} \]

No propagation possible
Decision: $\overline{c}$

Stack: $\overline{c}$
SAT solving

\[ C_1 = \overline{b} \lor c, \ C_2 = a \lor c, \ C_3 = \overline{a} \lor b, \ C_4 = \overline{a} \lor \overline{b}, \ C_5 = a \lor \overline{b}, \ C_6 = b \lor \overline{c} \]

1: procedure SAT\((C)\)
2: while \(\top\) do
3: \hspace{1em} if \text{PROPAGATE} then
4: \hspace{2em} if \text{\lnot} \text{DECIDE} then
5: \hspace{3em} return SAT
6: \hspace{2em} continue
7: \hspace{1em} if level = 0 then
8: \hspace{2em} return UNSAT
9: \hspace{1em} \text{ANALYSE}
10: \hspace{1em} \text{BACKTRACK}

Stack: \(\Box \overline{c}, \overline{b}, a\)

- No propagation possible
- Decision: \(\overline{c}\)
- Propagation: \(C_1 : \overline{b}, C_2 : a\)
SAT solving

\[ C_1 = \overline{b} \lor c, \quad C_2 = a \lor c, \quad C_3 = \overline{a} \lor b, \quad C_4 = \overline{a} \lor \overline{b}, \quad C_5 = a \lor \overline{b}, \quad C_6 = b \lor \overline{c} \]

1: procedure SAT(C) 2: while ⊤ do 3: if PROPAGATE then 4: if ¬DECIDE then 5: return SAT 6: continue 7: if level = 0 then 8: return UNSAT 9: Analyse 10: Backtrack

Stack: \[ \overline{c}, \overline{b}, a \]

- No propagation possible
- Decision: \( \overline{c} \)
- Propagation: \( C_1 : \overline{b}, \quad C_2 : a \)
- Conflict: \( C_3 \)
SAT solving

\[ C_1 = \overline{b} \lor c, \quad C_2 = a \lor c, \quad C_3 = \overline{a} \lor b, \quad C_4 = \overline{a} \lor \overline{b}, \quad C_5 = a \lor \overline{b}, \quad C_6 = b \lor \overline{c} \]

1: \textbf{procedure} SAT(C)
2: \hspace{1em} \textbf{while } \top \textbf{ do}
3: \hspace{2em} \textbf{if } \text{PROPAGATE} \textbf{ then}
4: \hspace{3em} \textbf{if } \overline{\text{DECIDE}} \textbf{ then}
5: \hspace{4em} \textbf{return} SAT
6: \hspace{3em} \textbf{continue}
7: \hspace{2em} \textbf{if } \text{level} = 0 \textbf{ then}
8: \hspace{3em} \textbf{return} UNSAT
9: \hspace{2em} \textbf{Analyse}
10: \hspace{2em} \textbf{Backtrack}

\begin{itemize}
  \item No propagation possible
  \item Decision: \( \overline{c} \)
  \item Propagation: \( C_1 : \overline{b}, \quad C_2 : a \)
  \item Conflict: \( C_3 \)
  \item Analyze: learn \( c \) and backtrack
\end{itemize}

Stack: \( c \)
SAT solving

\[ C_1 = \overline{b} \lor c, \ C_2 = a \lor c, \ C_3 = \overline{a} \lor b, \ C_4 = \overline{a} \lor \overline{b}, \ C_5 = a \lor \overline{b}, \ C_6 = b \lor \overline{c} \]

1: procedure SAT(\(C\))
2: \textbf{while} \(\top\) \textbf{do}
3: \textbf{if} PROPAGATE \textbf{then}
4: \textbf{if} \neg\text{Decide} \textbf{then}
5: \textbf{return} SAT
6: \textbf{continue}
7: \textbf{if} level = 0 \textbf{then}
8: \textbf{return} UNSAT
9: \textbf{Analyse}
10: \textbf{Backtrack}

► No propagation possible
► Decision: \(\overline{c}\)
► Propagation: \(C_1 : \overline{b}, \ C_2 : a\)
► Conflict: \(C_3\)
► Analyze: learn \(c\) and backtrack
► Propagation: \(C_6 : b, \ C_3 : \overline{a}\)

Stack: \(c, b, \overline{a}\)
SAT solving

\[ C_1 = \overline{b} \lor c, \ C_2 = a \lor c, \ C_3 = \overline{a} \lor b, \ C_4 = \overline{a} \lor \overline{b}, \ C_5 = a \lor \overline{b}, \ C_6 = b \lor \overline{c} \]

1: procedure SAT(C)                                • No propagation possible
2:     while ⊤ do                                   • Decision: \( \overline{c} \)
3:       if PROPAGATE then                         • Propagation: \( C_1 : \overline{b}, \ C_2 : a \)
4:         if ¬DECIDE then                        • Conflict: \( C_3 \)
5:           return SAT                           • Analyze: learn \( c \) and backtrack
6:         continue                               • Propagation: \( C_6 : b, \ C_3 : \overline{a} \)
7:       if level = 0 then                        • Conflict: \( C_4 \)
8:         return UNSAT                           •
9:     Analyse                                     •
10:    Backtrack                                  •

Stack: \( c, b, \overline{a} \)

\[ \begin{array}{c}
\text{c} \quad \text{C}_6 \\
\downarrow \quad \quad \quad \\
\text{b} \\
\quad \quad \quad \\
\text{a} \quad \text{C}_5 \\
\end{array} \]

\[ \begin{array}{c}
\text{c} \quad \text{C}_4 \\
\downarrow \\
\overline{a} \\
\end{array} \]
SAT solving

\[ C_1 = \overline{b} \lor c, \ C_2 = a \lor c, \ C_3 = \overline{a} \lor b, \ C_4 = \overline{a} \lor \overline{b}, \ C_5 = a \lor \overline{b}, \ C_6 = b \lor \overline{c} \]

1: \textbf{procedure} SAT(\(C\))
2: \hspace{1em} \textbf{while} \(\top\) \textbf{do}
3: \hspace{2em} \textbf{if} \ PROPAGATE \textbf{then}
4: \hspace{3em} \textbf{if} \ \neg\text{Decide} \textbf{then}
5: \hspace{4em} \text{return} \ SAT
6: \hspace{3em} \text{continue}
7: \hspace{2em} \textbf{if} \ level = 0 \textbf{then}
8: \hspace{3em} \text{return} \ UNSAT
9: \hspace{2em} \textbf{Analyse}
10: \hspace{2em} \textbf{Backtrack}

Stack: \(c, b, \overline{a}\)

- No propagation possible
- Decision: \(\overline{c}\)
- Propagation: \(C_1 : \overline{b}, C_2 : a\)
- Conflict: \(C_3\)
- Analyze: learn \(c\) and backtrack
- Propagation: \(C_6 : b, C_3 : \overline{a}\)
- Conflict: \(C_4\)
- No decisions (level = 0)
  Return unsat
Analysis and proofs

1: **procedure** Analyse
2: \( n \leftarrow 0 \)
3: \( C \leftarrow \text{conflicting clause} \)
4: \( P \leftarrow C \)
5: repeat
6: \hspace{1em} for all \( \ell \in C \) s.t. \( \neg \text{MARKED}(\ell) \) do
7: \hspace{2em} \text{MARK}(\ell)
8: \hspace{2em} if \( \text{LEVEL}(\ell) < \text{current level} \) then
9: \hspace{3em} \( R \leftarrow R \cup \{\ell\} \)
10: \hspace{2em} else
11: \hspace{3em} \( n \leftarrow n + 1 \)
12: \hspace{1em} repeat
13: \hspace{2em} \( \ell \leftarrow \text{POP} \)
14: \hspace{1em} until \( \text{MARKED}(\ell) \)
15: \text{UNMARK}(\ell)
16: \( n \leftarrow n - 1 \)
17: if \( n > 1 \) then
18: \hspace{1em} \( C \leftarrow \text{REASON}(\ell) \setminus \{\ell\} \)
19: \hspace{1em} \( P \leftarrow P \odot \text{REASON}(\ell) \)
20: until \( n = 1 \)
21: return \( (R \cup \{\ell\}, P) \)
SAT solvers use many processing techniques to improve efficiency

- Weakening techniques: removing clause. No impact on proof
- Some techniques introduce new clauses easily derivable by resolution
- Some techniques do not preserve logical equivalence. E.g. symmetry breaking
Outline

Introduction

Proofs for SAT
- Prerequisites
- SAT and proofs
- Proof formats
**SAT proof format: TraceCheck**

\[ C_1 = \overline{b} \lor c, \ C_2 = a \lor c, \ C_3 = \overline{a} \lor b, \ C_4 = \overline{a} \lor \overline{b}, \ C_5 = a \lor \overline{b}, \ C_6 = b \lor \overline{c} \]

\( a \rightarrow 1, \ b \rightarrow 2, \ c \rightarrow 3 \)

<table>
<thead>
<tr>
<th>DIMACCS</th>
<th>TraceCheck</th>
<th>TraceCheck format</th>
</tr>
</thead>
<tbody>
<tr>
<td>p cnf 3 6</td>
<td>1 -2 3 0 0</td>
<td>( \langle \text{trace} \rangle ::= { \langle \text{clause} \rangle } )</td>
</tr>
<tr>
<td></td>
<td>-2 3 0</td>
<td>( \langle \text{clause} \rangle ::= \langle \text{pos} \rangle \langle \text{literals} \rangle \langle \text{antecedents} \rangle )</td>
</tr>
<tr>
<td></td>
<td>1 3 0</td>
<td>( \langle \text{literals} \rangle ::= \ast \mid { \langle \text{lit} \rangle } \ '0' )</td>
</tr>
<tr>
<td></td>
<td>-1 2 0</td>
<td>( \langle \text{antecedents} \rangle ::= { \langle \text{pos} \rangle } \ '0' )</td>
</tr>
<tr>
<td></td>
<td>-1 -2 0</td>
<td>( \langle \text{lit} \rangle = \langle \text{pos} \rangle \mid \langle \text{neg} \rangle )</td>
</tr>
<tr>
<td></td>
<td>1 -2 0</td>
<td>( \langle \text{lit} \rangle = \langle \text{pos} \rangle \mid \langle \text{neg} \rangle )</td>
</tr>
<tr>
<td></td>
<td>2 -3 0</td>
<td>( \langle \text{lit} \rangle = \langle \text{pos} \rangle \mid \langle \text{neg} \rangle )</td>
</tr>
<tr>
<td></td>
<td>7 -2 0 4 5 0</td>
<td>( \langle \text{lit} \rangle = \langle \text{pos} \rangle \mid \langle \text{neg} \rangle )</td>
</tr>
<tr>
<td></td>
<td>8 3 0 1 2 3 0</td>
<td>( \langle \text{lit} \rangle = \langle \text{pos} \rangle \mid \langle \text{neg} \rangle )</td>
</tr>
<tr>
<td></td>
<td>9 0 6 7 8 0</td>
<td>( \langle \text{lit} \rangle = \langle \text{pos} \rangle \mid \langle \text{neg} \rangle )</td>
</tr>
<tr>
<td></td>
<td>8 3 0 1 2 3 0</td>
<td>( \langle \text{lit} \rangle = \langle \text{pos} \rangle \mid \langle \text{neg} \rangle )</td>
</tr>
</tbody>
</table>

- Input clause: \( \langle \text{antecedents} \rangle \) is ‘0’
- Learned clause: \( \langle \text{literals} \rangle \) can be ‘\ast’
  can be computed anyway
- lines and \( \langle \text{antecedents} \rangle \) not sorted

learned clause \( C_8 = c \) is derived from \( C_1, \ C_2 \) and \( C_3 \): \( C_8 = C_3 \lor C_2 \lor C_1 \)

\[
\begin{align*}
C_3 &= \overline{a} \lor b \\
C_2 &= a \lor c \\
\therefore C_1 &= \overline{b} \lor c \\
\therefore C_8 &= c
\end{align*}
\]
Reverse Unit Propagation (RUP)

Clause $C = \ell_1 \lor \cdots \lor \ell_n$ is Reverse Unit Propagation w.r.t. set of clause $S$ if unit propagation alone can show that $S \cup \{\overline{\ell_1}, \ldots, \overline{\ell_n}\}$ is unsatisfiable.

Hence $S \land \neg C \models \Box$, i.e. $S \models C$.

Then there is $R_1 \diamond \cdots \diamond R_m = \Box$ with $R_i$s all different, and each $R_i$ either in $S$ or negated literal of $C$.

\[
\begin{array}{c}
R_1 & R_2 & R_3 & R_4 & R_5 & R_6 \\
C_1 & C_2 & C_3 & C_4 & & \\
\hline
\end{array}
\]
Reverse Unit Propagation (RUP)

Clause $C = \ell_1 \lor \cdots \lor \ell_n$ is Reverse Unit Propagation w.r.t. set of clause $S$ if unit propagation alone can show that $S \cup \{\overline{\ell_1}, \ldots, \overline{\ell_n}\}$ is unsatisfiable.

Hence $S \land \neg C \models \Box$, i.e. $S \models C$.

Then there is $R_1 \diamond \cdots \diamond R_m = \Box$ with $R_i$'s all different, and each $R_i$ either in $S$ or negated literal of $C$.

\[
\begin{array}{c}
R_1 \\
\hline
C_1 \\
R_2 \\
\hline
C_2 \\
R_3 \\
\hline
C_3 \\
\hline
C_4 \\
\hline
R_5 \\
\hline
R_6 \\
\hline
\Box
\end{array}
\]
Reverse Unit Propagation (RUP)

Clause \( C = \ell_1 \lor \cdots \lor \ell_n \) is Reverse Unit Propagation w.r.t. set of clause \( S \) if unit propagation alone can show that \( S \cup \{\overline{\ell_1}, \ldots, \overline{\ell_n}\} \) is unsatisfiable.

Hence \( S \land \neg C \models \square \), i.e. \( S \models C \).

Then there is \( R_1 \diamond \ldots \diamond R_m = \square \) with \( R_i \)s all different, and each \( R_i \) either in \( S \) or negated literal of \( C \).

\[
\begin{array}{c}
R_1 \\
R_2 \\
\hline
C_1 \\
\hline
C_3 \lor \ell_1 \\
\hline
\overline{\ell_1} \\
\hline
C_3 \\
\hline
C_4 \\
\hline
\square
\end{array}
\]
Reverse Unit Propagation (RUP)

Clause $C = \ell_1 \lor \cdots \lor \ell_n$ is Reverse Unit Propagation w.r.t. set of clause $S$ if unit propagation alone can show that $S \cup \{\overline{\ell_1}, \ldots, \overline{\ell_n}\}$ is unsatisfiable.

Hence $S \land \neg C \models \square$, i.e. $S \models C$.

Then there is $R_1 \diamond \cdots \diamond R_m = \square$ with $R_i$s all different, and each $R_i$ either in $S$ or negated literal of $C$.

\[
\begin{array}{c}
R_1 \\
R_2 \\
C_1 \\
R_3 \\
C_3 \lor \ell_1 \\
C_3 \\
C_4 \\
R_5 \\
R_6 \\
\square
\end{array}
\]
Reverse Unit Propagation (RUP)

Clause $C = \ell_1 \lor \cdots \lor \ell_n$ is Reverse Unit Propagation w.r.t. set of clause $S$ if unit propagation alone can show that $S \cup \{\overline{\ell_1}, \ldots, \overline{\ell_n}\}$ is unsatisfiable.

Hence $S \land \neg C \models \square$, i.e. $S \models C$.

Then there is $R_1 \Diamond \cdots \Diamond R_m = \square$ with $R_i$s all different, and each $R_i$ either in $S$ or negated literal of $C$.

\[
\begin{array}{c}
R_1 & R_2 \\
\hline
C_1 & R_3 \\
\hline
C_3 \lor \ell_1 & R_5 \\
\hline
C_4 \lor \ell_1 & \ell_1 \\
\hline
R_6
\end{array}
\]

Resolution proof of RUP can be computed by analysis of propagation graph.
SAT proof format: (D)RUP

\[ C_1 = \overline{b} \lor c, \ C_2 = a \lor c, \ C_3 = \overline{a} \lor b, \ C_4 = \overline{a} \lor \overline{b}, \ C_5 = a \lor \overline{b}, \ C_6 = b \lor \overline{c} \]

\[ a \rightarrow 1, \ b \rightarrow 2, \ c \rightarrow 3 \]

<table>
<thead>
<tr>
<th>DIMACS</th>
<th>RUP</th>
<th>RUP format</th>
</tr>
</thead>
<tbody>
<tr>
<td>p cnf 3 6</td>
<td>-2 0</td>
<td>Clauses, like DIMACS</td>
</tr>
<tr>
<td>-2 3 0</td>
<td>3 0</td>
<td>Checking ( \overline{b} ) is RUP: unit propagation for ( b )</td>
</tr>
<tr>
<td>1 3 0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>-1 2 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1 -2 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 -2 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 -3 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{c}
\overline{a} \\
C_4 \\
\end{array}
\quad
\begin{array}{c}
C_4 \\
\overline{b} \\
C_5 \\
\end{array}

\begin{array}{c}
b \\
C_5 \\
\end{array}
\quad
\begin{array}{c}
\overline{b} \\
C_4 \\
\overline{C_5} \\
\end{array}
\]

\[ \square \]
Proof checking RUP

- Each RUP clause consequence of input and previous RUP clauses
- Proof checking more resource consuming than SAT solving
  - SAT solving: clauses are regularly cleaned out
  - Proof Checking: clauses accumulate
  - SAT solving: propagations changed incrementally
  - Proof Checking: every RUP is an entire new work
- Solution (for the first point above): DRUP, notify deleted clauses
  - File format: \textit{d} followed by clause in DIMACS format
Other extension DRAT: stronger proof system
Outline

Introduction

Proofs for SAT
  Prerequisites
  SAT and proofs
  Proof formats
Outline

Introduction

Proofs for SAT
  Prerequisites
  SAT and proofs
  Proof formats
Satisfiability Modulo Theories $\approx$ SAT + expressiveness

Satisfiability of first-order formulas with interpreted and non-interpreted predicates and functions

Interpreted: Axioms (e.g. arrays) or Structure (e.g. linear arithmetic)

- **SAT solvers**

  $$\neg \left[ (p \Rightarrow q) \Rightarrow \left[ (\neg p \Rightarrow q) \Rightarrow q \right] \right]$$

- **congruence closure (uninterpreted symbols + equality)**

  $$a = b \land \left[ f(a) \neq f(b) \lor (p(a) \land \neg p(b)) \right]$$

- **in combination with arithmetic**

  $$a \leq b \land b \leq a + x \land x = 0 \land \left[ f(a) \neq f(b) \lor (p(a) \land \neg p(b + x)) \right]$$

- **quantifiers**

- ...
From propositional SAT to SMT

Input: \( a \leq b \land b \leq a + x \land x = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(b + x))] \)
From propositional SAT to SMT

Input: \( a \leq b \land b \leq a + x \land x = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(b + x))] \)

To SAT solver: \( p_{a \leq b} \land p_{b \leq a + x} \land p_{x = 0} \land [\neg p_{f(a) = f(b)} \lor (p_{q(a)} \land \neg p_{q(b + x)})] \)
From propositional SAT to SMT

Input: \( a \leq b \land b \leq a + x \land x = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(b + x))] \)

To SAT solver: \( p_{a \leq b} \land p_{b \leq a + x} \land p_{x=0} \land [\neg p_{f(a)=f(b)} \lor (p_{q(a)} \land \neg p_{q(b+x)})] \)

Boolean model: \( p_{a \leq b}, p_{b \leq a + x}, p_{x=0}, \neg p_{f(a)=f(b)} \)
From propositional SAT to SMT

**Input:** $a \leq b \land b \leq a + x \land x = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(b + x))]$

**To SAT solver:** $p_{a \leq b} \land p_{b \leq a + x} \land p_{x=0} \land [\neg p_f(a) = f(b) \lor (p_{q(a)} \land \neg p_{q(b+x)})]$

**Boolean model:** $p_{a \leq b}, p_{b \leq a + x}, p_{x=0}, \neg p_f(a) = f(b)$

**Theory reasoner:** $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable
From propositional SAT to SMT

Input: \( a \leq b \land b \leq a + x \land x = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(b + x))] \)

To SAT solver: \( p_{a \leq b} \land p_{b \leq a + x} \land p_{x=0} \land [\neg p_{f(a)=f(b)} \lor (p_{q(a)} \land \neg p_{q(b+x)})] \)

Boolean model: \( p_{a \leq b}, p_{b \leq a + x}, p_{x=0}, \neg p_{f(a)=f(b)} \)

Theory reasoner: \( a \leq b, b \leq a + x, x = 0, f(a) \neq f(b) \) unsatisfiable

New theory clause: \( \neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{x=0} \lor p_{f(a)=f(b)} \)
From propositional SAT to SMT

Input: \( a \leq b \land b \leq a + x \land x = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(b + x))] \)
To SAT solver: \( p_{a \leq b} \land p_{b \leq a + x} \land p_{x = 0} \land [\neg p_{f(a) = f(b)} \lor (p_{q(a)} \land \neg p_{q(b + x)})] \)
Boolean model: \( p_{a \leq b}, p_{b \leq a + x}, p_{x = 0}, \neg p_{f(a) = f(b)} \)
Theory reasoner: \( a \leq b, b \leq a + x, x = 0, f(a) \neq f(b) \) unsatisfiable
New theory clause: \( \neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{x = 0} \lor p_{f(a) = f(b)} \)
New theory clause: \( \neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{q(a)} \lor p_{q(b + x)} \)
From propositional SAT to SMT

Input: \( a \leq b \land b \leq a + x \land x = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(b + x))] \)

To SAT solver: \( p_{a \leq b} \land p_{b \leq a + x} \land p_{x=0} \land [\neg p f(a) = f(b) \lor (p_q(a) \land \neg p_{q(b+x)})] \)

Boolean model: \( p_{a \leq b}, p_{b \leq a + x}, p_{x=0}, \neg p f(a) = f(b) \)

Theory reasoner: \( a \leq b, b \leq a + x, x = 0, f(a) \neq f(b) \) unsatisfiable

New theory clause: \( \neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{x=0} \lor p f(a) = f(b) \)

New theory clause: \( \neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_q(a) \lor p_{q(b+x)} \)
From propositional SAT to SMT

Input: \( a \leq b \land b \leq a + x \land x = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(b + x))] \)

To SAT solver: \( p_{a \leq b} \land p_{b \leq a + x} \land p_{x = 0} \land [\neg p_f(a) = f(b) \lor (p_{q(a)} \land \neg p_{q(b + x)})] \)

Boolean model: \( p_{a \leq b}, p_{b \leq a + x}, p_{x = 0}, \neg p_f(a) = f(b) \)

Theory reasoner: \( a \leq b, b \leq a + x, x = 0, f(a) \neq f(b) \) unsatisfiable

New theory clause: \( \neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{x = 0} \lor p_f(a) = f(b) \)

New theory clause: \( \neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{q(a)} \lor p_{q(b + x)} \)
From propositional SAT to SMT

Input: \(a \leq b \land b \leq a + x \land x = 0 \land \left[ f(a) \neq f(b) \lor (q(a) \land \neg q(b + x)) \right]\)

To SAT solver: \(p_{a \leq b} \land p_{b \leq a + x} \land p_{x=0} \land \left[ \neg p_{f(a)=f(b)} \lor (p_{q(a)} \land \neg p_{q(b+x)}) \right]\)

Boolean model: \(p_{a \leq b}, p_{b \leq a + x}, p_{x=0}, \neg p_{f(a)=f(b)}\)

Theory reasoner: \(a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)\) unsatisfiable

New theory clause: \(\neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{x=0} \lor p_{f(a)=f(b)}\)

New theory clause: \(\neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{q(a)} \lor p_{q(b+x)}\)
SMT in practice

- online decision procedures
  theory checks propositional assignment on the fly

- small explanations
  unsat core of propositional assignment
  discard classes of propositional assignments (not one by one)

- theory propagation
  instead of guessing propositional variable assignments, SAT solver
  assigns theory-entailed literals

- ackermannization, simplifications, and other magic

Challenge: collect enough information
SMT in practice

- online decision procedures
  - theory checks propositional assignment on the fly
  - No influence on proof

- small explanations
  - unsat core of propositional assignment
  - discard classes of propositional assignments (not one by one)

- theory propagation
  - instead of guessing propositional variable assignments, SAT solver assigns theory-entailed literals

- ackermannization, simplifications, and other magic

Challenge: collect enough information
SMT in practice

- online decision procedures
  theory checks propositional assignment on the fly
  \textit{No influence on proof}

- small explanations
  unsat core of propositional assignment
  discard classes of propositional assignments (not one by one)
  \textit{No influence on proof (small theory clauses)}

- theory propagation
  instead of guessing propositional variable assignments, SAT solver assigns theory-entailed literals

- ackermannization, simplifications, and other magic

Challenge: collect enough information
SMT in practice

- online decision procedures
  theory checks propositional assignment on the fly
  *No influence on proof*

- small explanations
  *unsat core of propositional assignment*
  discard classes of propositional assignments (not one by one)
  *No influence on proof (small theory clauses)*

- theory propagation
  instead of guessing propositional variable assignments, SAT solver
  assigns theory-entailed literals
  *May need explanation (theory clause)*

- ackermannization, simplifications, and other magic

Challenge: collect enough information
SMT in practice

- online decision procedures
  - theory checks propositional assignment on the fly
  - *No influence on proof*

- small explanations
  - *unsat core of propositional assignment*
  - *discard classes of propositional assignments (not one by one)*
  - *No influence on proof (small theory clauses)*

- theory propagation
  - instead of guessing propositional variable assignments, SAT solver assigns theory-entailed literals
  - *May need explanation (theory clause)*

- ackermannization, simplifications, and other magic
  - *Sometimes cumbersome to prove*

Challenge: collect enough information
Outline

Introduction

Proofs for SAT

Prerequisites
SAT and proofs
Proof formats
Theory reasoning proofs
Congruence closure (1/3)

Axioms of equality

- **Reflexivity:** \( \forall x . x = x \)
- **Symmetry:** \( \forall x, y . x = y \Rightarrow y = x \)
- **Transitivity:** \( \forall x, y, z . (x = y \land y = z) \Rightarrow x = z \)
- **Congruence (schema):**
  \[
  \forall x_1, \ldots, x_n, y_1, \ldots, y_n .
  \quad (x_1 = y_1 \land \cdots \land x_n = y_n) \Rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)
  \]

Reflexivity and Symmetry used silently
Transitivity chains of arbitrary length
Consider the terms: $a, b, c, f(a), f(b)$
Theory reasoning proofs
Congruence closure (2/3)

Consider the terms: \( a, b, c, f(a), f(b) \)

\[
\begin{array}{ccc}
  f(a) & f(b) \\
  a & c & b \\
\end{array}
\]

each term in its equivalence class
Theory reasoning proofs

Congruence closure (2/3)

Consider the terms: $a$, $b$, $c$, $f(a)$, $f(b)$
And literals: $a = c$

$\begin{array}{c|c|c|c|}
\hline
f(a) & f(b) \\
\hline
\end{array}$

- each term in its equivalence class
- equality $\rightarrow$ class merge

$\begin{array}{c|c|c|c|}
\hline
a & a = c & c & b \\
\hline
\end{array}$
Consider the terms: $a, b, c, f(a), f(b)$
And literals: $a = c, c = b$

- each term in its equivalence class
- equality $\rightarrow$ class merge

$\begin{align*}
  f(a) & \quad f(b) \\
  a & \quad a = c \\
  c & \quad c = b \\
  b & 
\end{align*}$
Consider the terms: $a, b, c, f(a), f(b)$
And literals: $a = c$, $c = b$

$f(a) \sim f(b)$

$a \overset{a = c}{\sim} c \overset{c = b}{\sim} b$

- each term in its equivalence class
- equality $\rightarrow$ class merge
- congruence $\rightarrow$ class merge
Theory reasoning proofs
Congruence closure (2/3)

Consider the terms: \( a, b, c, f(a), f(b) \)
And literals: \( a = c, c = b, f(a) \neq f(b) \)

\[
\begin{align*}
&f(a) \not= f(b) \\
&a = c & c = b & b
\end{align*}
\]

▶ each term in its equivalence class
▶ equality \( \rightarrow \) class merge
▶ congruence \( \rightarrow \) class merge
▶ detect conflicts
Theory reasoning proofs

Congruence closure (2/3)

Consider the terms: \(a, b, c, f(a), f(b)\)
And literals: \(a = c, c = b, f(a) \neq f(b)\)

\[
f(a) \neq f(b)\]

Each term in its equivalence class

Equality \(\rightarrow\) class merge

Congruence \(\rightarrow\) class merge

Detect conflicts

In practice: efficient (merge, congruence and conflict detection)
Theory reasoning proofs

Congruence closure (2/3)

Consider the terms: $a, b, c, f(a), f(b)$

And literals: $a = c, c = b, f(a) \neq f(b)$

$f(a) \neq f(b)$

- each term in its equivalence class
- equality $\rightarrow$ class merge
- congruence $\rightarrow$ class merge
- detect conflicts

In practice: efficient (merge, congruence and conflict detection)

Theory reasoning proof, from graph:
Theory reasoning proofs

Congruence closure (2/3)

Consider the terms: \( a, b, c, f(a), f(b) \)
And literals: \( a = c, c = b, f(a) \neq f(b) \)

\[
f(a) \neq f(b)
\]

▶ each term in its equivalence class
▶ equality \( \rightarrow \) class merge
▶ congruence \( \rightarrow \) class merge
▶ detect conflicts

In practice: efficient (merge, congruence and conflict detection)

Theory reasoning proof, from graph:
▶ conflict \( f(a) \neq f(b) \) with an implied literal
Theory reasoning proofs

Congruence closure (2/3)

Consider the terms: \(a, b, c, f(a), f(b)\)
And literals: \(a = c, c = b, f(a) \neq f(b)\)

\[
f(a) \neq f(b) \quad \Rightarrow \quad a = c, c = b\]

▶ each term in its equivalence class
▶ equality \(\rightarrow\) class merge
▶ congruence \(\rightarrow\) class merge
▶ detect conflicts

In practice: efficient (merge, congruence and conflict detection)

Theory reasoning proof, from graph:

▶ conflict \(f(a) \neq f(b)\) with an implied literal
▶ entailed by congruence: \(a \neq b \lor f(a) = f(b)\)
Consider the terms: \( a, b, c, f(a), f(b) \)

And literals: \( a = c, c = b, f(a) \neq f(b) \)

\[
f(a) \neq f(b)
\]

- each term in its equivalence class
- equality \( \rightarrow \) class merge
- congruence \( \rightarrow \) class merge
- detect conflicts

In practice: efficient (merge, congruence and conflict detection)

Theory reasoning proof, from graph:

- conflict \( f(a) \neq f(b) \) with an implied literal
- entailed by congruence: \( a \neq b \lor f(a) = f(b) \)
- and \( a = b \) comes from transitivity: \( a \neq c \lor c \neq b \lor a = b \)
Theory reasoning proofs

Congruence closure (2/3)

Consider the terms: \( a, b, c, f(a), f(b) \)
And literals: \( a = c, c = b, f(a) \neq f(b) \)

\[
\begin{align*}
\text{\( f(a) \neq f(b) \)} & \quad \text{each term in its equivalence class} \\
\quad & \quad \text{equality } \longrightarrow \text{ class merge} \\
\quad & \quad \text{congruence } \longrightarrow \text{ class merge} \\
\quad & \quad \text{detect conflicts}
\end{align*}
\]

In practice: efficient (merge, congruence and conflict detection)

Theory reasoning proof, from graph:

- conflict \( f(a) \neq f(b) \) with an implied literal
- entailed by congruence: \( a \neq b \lor f(a) = f(b) \)
- and \( a = b \) comes from transitivity: \( a \neq c \lor c \neq b \lor a = b \)
- \textit{resolution} compute the theory clause: \( a \neq c \lor c \neq b \lor f(a) = f(b) \)
Theory reasoning proofs

Congruence closure (3/3)

Congruence closure proofs

- Use the same data-structures as for conflict computation
- Transitivity + Congruence glued together with resolution
- Smallest conflict computation/smallest proof: NP complete (teaser for the SMT workshop 2015)
Theory reasoning proofs

Combination of theories

Theory reasoning proof, with combination of theories:

- conflict $f(a) \neq f(b)$ with an implied literal
Theory reasoning proofs

Combination of theories

Theory reasoning proof, with combination of theories:

- conflict $f(a) \neq f(b)$ with an implied literal
- entailed by congruence: $a \neq b \lor f(a) = f(b)$
Theory reasoning proofs

Combination of theories

Theory reasoning proof, with combination of theories:

- conflict $f(a) \neq f(b)$ with an implied literal
- entailed by congruence: $a \neq b \lor f(a) = f(b)$
- and $a = b$ comes from another theory clause:
  $$\neg a \leq b \lor \neg b \leq a + x \lor x \neq 0 \lor a = b$$
Theory reasoning proofs
Combination of theories

Theory reasoning proof, with combination of theories:

▶ conflict \( f(a) \neq f(b) \) with an implied literal

▶ entailed by congruence: \( a \neq b \lor f(a) = f(b) \)

▶ and \( a = b \) comes from another theory clause:
\[
\neg a \leq b \lor \neg b \leq a + x \lor x \neq 0 \lor a = b
\]

▶ resolution compute the theory clause:
\[
\neg a \leq b \lor \neg b \leq a + x \lor x \neq 0 \lor f(a) = f(b)
\]
Theory reasoning proofs
Combination of theories

Theory reasoning proof, with combination of theories:

- conflict $f(a) \neq f(b)$ with an implied literal
- entailed by congruence: $a \neq b \lor f(a) = f(b)$
- and $a = b$ comes from another theory clause:
  $\neg a \leq b \lor \neg b \leq a + x \lor x \neq 0 \lor a = b$

- *resolution* compute the theory clause:
  $\neg a \leq b \lor \neg b \leq a + x \lor x \neq 0 \lor f(a) = f(b)$

Over-simplification:

- delayed theory combination
- model-based combination

Both are combination techniques within the underlying SAT solver.
Pretty trivial for proof production (same for splitting on demand)
Theory reasoning proofs

Linear arithmetic

- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate

$L_1, \ldots, L_n$ unsatisfiable set of linear constraints?

There exists a trivially unsatisfiable linear combination $\Sigma_i c_i L_i$
Theory reasoning proofs
Linear arithmetic

- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate

$L_1, \ldots, L_n$ unsatisfiable set of linear constraints?
There exists a trivially unsatisfiable linear combination $\sum_i c_i L_i$

$y > 1$
Theory reasoning proofs

Linear arithmetic

- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate

$L_1, \ldots, L_n$ unsatisfiable set of linear constraints?
There exists a trivially unsatisfiable linear combination $\Sigma_i c_i L_i$

- $y > 1$, $x < 1$
Theory reasoning proofs

Linear arithmetic

- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate

$L_1, \ldots, L_n$ unsatisfiable set of linear constraints?
There exists a trivially unsatisfiable linear combination $\sum_i c_i L_i$

- $y > 1$, $x < 1$, $y \leq x$

![Diagram with constraints](image)

![Diagram with constraints](image)
Theory reasoning proofs

Linear arithmetic

▶ Many linear arithmetic decision procedures based on simplex
▶ Simplex detects inconsistency
▶ Farkas lemma can be used to provide certificate

$L_1, \ldots, L_n$ unsatisfiable set of linear constraints?
There exists a trivially unsatisfiable linear combination $\sum_i c_i L_i$

▶ $y > 1$, $x < 1$, $y \leq x$
▶ inconsistency
Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate

$L_1, \ldots, L_n$ unsatisfiable set of linear constraints?
There exists a trivially unsatisfiable linear combination $\sum_i c_i L_i$

- $y > 1$, $x < 1$, $y \leq x$
- inconsistency
  - $x < 1$
  - $y \leq x$
  - $y > 1$
  - $0 < 0$
Theory reasoning proofs

Linear arithmetic

- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate

$L_1, \ldots, L_n$ unsatisfiable set of linear constraints?
There exists a trivially unsatisfiable linear combination $\sum_i c_i L_i$

- $\quad y > 1, \quad x < 1, \quad y \leq x$
- inconsistency
  
  $\quad \begin{align*}
  x &< 1 \\
  + & \quad y \leq x \\
  - & \quad y > 1 \\
  \hline
  0 &< 0
  \end{align*}$

- Clause: $\neg y > 1 \lor \neg x < 1 \lor \neg y \leq x$
Theory reasoning proofs

Linear arithmetic

- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate

$L_1, \ldots, L_n$ unsatisfiable set of linear constraints?

There exists a trivially unsatisfiable linear combination $\sum_i c_i L_i$

- $y > 1$, $x < 1$, $y \leq x$
- Inconsistency
  
  $\begin{align*}
  x &< 1 \\
  + & y \leq x \\
  - & y > 1 \\
  \hline
  0 &< 0
  \end{align*}$

- Clause: $\neg y > 1 \lor \neg x < 1 \lor \neg y \leq x$

And also

- integers: branches, cuts
- simplifications, bound propagations…
Quantifiers and proofs

- Quantifiers mainly come from instantiation
- Proof is simply
  \[ \neg \forall x \varphi(x) \lor \varphi(t) \]
- \( \forall x \varphi(x) \) is an abstract Boolean variable for the SAT solver
- Resolution, again
- Skolemization is a problem though
Other theories

- arrays
  Not that different from uninterpreted symbols with equality
- (co-)inductive data types
- bit-vectors
  Often rely on rewriting and bit-blasting
- strings
- Floating point arithmetic
- non-linear arithmetic: Can be quite non-trivial (e.g. CAD)
Outline

Introduction

Proofs for SAT
  Prerequisites
  SAT and proofs
  Proof formats
SMT Proof formats

- No standard format (yet)
- Proofs (CVC4) vs. Proof traces (veriT, z3)
- CVC4: LFSC, generic, flexible, extendable
- z3: SMT-LIB terms
- veriT: simple
In SMT-LIB 2.0 format:

(set-logic QF_UFLRA)
(set-info :source | Example formula in SMT-LIB 2.0 |)
(set-info :smt-lib-version 2.0)
(declare-fun f (Real) Real)
(declare-fun q (Real) Bool)
(declare-fun a () Real)
(declare-fun b () Real)
(declare-fun x () Real)
(assert (and (<= a b) (<= b (+ a x)) (= x 0)
    (or (not (= (f a) (f b)))
        (and (q a) (not (q (+ b x)))))
    ))
(check-sat)
(exit)
(set .c1 (input :conclusion ((and (<= a b) (<= b (+ a x)) (= x 0))
   (or (not (= (f b) (f a))) (and (q a) (not (q (+ b x))))))))
(set .c2 (and :clauses (.c1) :conclusion ((<= a b))))
(set .c3 (and :clauses (.c1) :conclusion ((<= b (+ a x))))
(set .c4 (and :clauses (.c1) :conclusion ((= x 0))))
(set .c5 (and :clauses (.c1) :conclusion
   ((or (not (= (f b) (f a))) (and (q a) (not (q (+ b x)))))))
(set .c6 (and_pos :conclusion ((not (and (q a) (not (q (+ b x)))))) (q a))))
(set .c7 (and_pos :conclusion ((not (and (q a) (not (q (+ b x)))))) (not (q (+ b x))))
(set .c8 (or :clauses (.c5) :conclusion
   ((not (= (f b) (f a))) (and (q a) (not (q (+ b x)))))))
(set .c9 (eq_congruent :conclusion ((not (= a b)) (= (f b) (f a))))
(set .c10 (la_disequality :conclusion ((or (= a b) (not (<= a b)) (not (<= b a))))))
(set .c11 (or :clauses (.c10) :conclusion ((= a b) (not (<= a b)) (not (<= b a))))
(set .c12 (resolution :clauses (.c11 .c2) :conclusion ((= a b) (not (<= b a))))
(set .c13 (la_generic :conclusion ((not (<= b (+ a x)) (<= b a) (not (= x 0))))
(set .c14 (resolution :clauses (.c13 .c3 .c4) :conclusion ((<= b a)))))
(set .c15 (resolution :clauses (.c12 .c14) :conclusion ((= a b))))
(set .c16 (resolution :clauses (.c9 .c15) :conclusion ((= (f b) (f a))))
(set .c17 (resolution :clauses (.c8 .c16) :conclusion ((and (q a) (not (q (+ b x))))))
(set .c18 (resolution :clauses (.c6 .c17) :conclusion ((q a))))
(set .c19 (resolution :clauses (.c7 .c17) :conclusion ((not (q (+ b x))))))
(set .c20 (eq_congruent_pred :conclusion ((not (= a (+ b x))) (not (q a)) (q (+ b x)))))

(set .c21 (resolution :clauses (.c20 .c18 .c19) :conclusion ((not (= a (+ b x)))))

(set .c22 (la_disequality :conclusion

  ((or (= a (+ b x)) (not (<= a (+ b x))) (not (<= (+ b x) a))))

(set .c23 (or :clauses (.c22) :conclusion

  ((= a (+ b x)) (not (<= a (+ b x))) (not (<= (+ b x) a))))

(set .c24 (resolution :clauses (.c23 .c21) :conclusion

  ((not (<= a (+ b x))) (not (<= (+ b x) a)))))

(set .c25 (eq_congruent_pred :conclusion

  ((not (= a b)) (not (= (+ a x) (+ b x))) (<= a (+ b x)) (not (<= b (+ a x))))))

(set .c26 (eq_congruent :conclusion ((not (= a b)) (not (= x x)) (= (+ a x) (+ b x))))

(set .c27 (eq_reflexive :conclusion ((= x x))))

(set .c28 (resolution :clauses (.c26 .c27) :conclusion ((not (= a b)) (= (+ a x) (+ b x))))

(set .c29 (resolution :clauses (.c25 .c28) :conclusion

  ((not (= a b)) (<= a (+ b x)) (not (<= b (+ a x))))))

(set .c30 (resolution :clauses (.c29 .c3 .c15) :conclusion (((<= a (+ b x)))))

(set .c31 (resolution :clauses (.c24 .c30) :conclusion ((not (<= (+ b x) a))))

(set .c32 (la_generic :conclusion (((<= (+ b x) a) (not (= a b)) (not (= x 0)))))

(set .c33 (resolution :clauses (.c32 .c4 .c15 .c31) :conclusion ())))
z3 proof (1/2)

(let ((q b) (? (q b)) (?x49 (* (- 1.0) b)) (?x50 (+ a ?x49))
    (?x51 (<= ?x50 0.0)) (?x35 (f b)) (?x34 (f a))
    (?x36 (= ?x34 ?x35)) (?x37 (not ?x36))
    (?x43 (or ?x37 (and (q a) (not (q (+ b x)))))
    (?x33 (= x 0.0)) (?x57 (+ a ?x49 x)) (?x56 (>= ?x57 0.0))
    (?x44 (and (<= a b) (<= b (+ a x)) $x33 $x43))
    (@x60 (monotonicity (rewrite (= (<= a b) $x51))
        (rewrite (= (<= b (+ a x)) $x56))
        (= $x44 (and $x51 $x56 $x33 $x43))))
    (@x61 (mp (asserted $x44) @x60 (and $x51 $x56 $x33 $x43)))
    (@x62 (and-elim @x61 $x56))
    (let ((@x70 (trans (monotonicity (and-elim @x61 $x33) (= ?x57 (+ a ?x49 0.0)))
        (rewrite (= (+ a ?x49 0.0) ?x50)) (= ?x57 ?x50)))))
    (@x63 (mp (and-elim @x61 $x56) (monotonicity @x70 (= $x56 $x71) $x71)))
    (let ((@x121 (monotonicity (symm ((_ th-lemma arith eq-propagate 1 1) @x74 @x62 (= a b)) (= b a))
        (= $x82 (q a))))
    (let ((@x115 (monotonicity (symm ((_ th-lemma arith eq-propagate 1 1) @x74 @x62 (= a b)) (= b a))
        (= ?x35 ?x34))))
    (let ((@x100 (or $x37 $x97)))
    (let ((@x102 (monotonicity (rewrite (= (and $x38 (not $x82)) $x97))
        (= (or $x37 (and $x38 (not $x82)) $x100))))
    (let ((@x90 (not $x82)))
    (let ((@x88 (and $x38 $x85)))
    (let ((@x91 (or $x37 $x88)))
    (let ((@x81 (trans (monotonicity (and-elim @x61 $x33) (= (+ b x) (+ b 0.0)))
        (rewrite (= (+ b 0.0) b)) (= (+ b x) b))))
    (let ((@x87 (monotonicity (monotonicity @x81 (= (q (+ b x) $x82)) (= (not (q (+ b x)) $x85))))))
(let ((@x93 (monotonicity (monotonicity @x87 (= (and $x38 (not (q (+ b x)))) $x88))
  (= $x43 $x91)))))
(let ((@x103 (mp (mp (and-elim @x61 $x43) @x93 $x91) @x102 $x100)))
(let ((@x119 (unit-resolution (def-axiom (or $x96 $x38))
  (unit-resolution @x103 (symm @x115 $x36) $x97) $x38)))
(let ((@x118 (unit-resolution (def-axiom (or $x96 $x85))
  (unit-resolution @x103 (symm @x115 $x36) $x97) $x85)))
  (unit-resolution @x118 (mp @x119 (symm @x121 (= $x38 $x82)) $x82) false))))))))}}}
(check
(\% a var_real
(\% b var_real
(\% x var_real
(\% f (term (arrow Real Real))
(\% q (term (arrow Real Bool))
(\% @F1 (th_holds (<=_Real (a_var_real a) (a_var_real b)))
(\% @F2 (th_holds (<=_Real (a_var_real b) (+_Real (a_var_real a) (a_var_real x))))
(\% @F3 (th_holds (= Real (a_var_real x) (a_real 0/1)))
(\% @F4 (th_holds (or (not (= Real (apply _ _ f (a_var_real a)) (apply _ _ f (a_var_real b))))
    (and (= Bool (apply _ _ q (a_var_real a)) btrue)
    (= Bool (apply _ _ q (+_Real (a_var_real b) (a_var_real x))) bfalse))))
(: (holds cln)

(decl_atom (<=_Real (a_var_real a) (a_var_real b)) (\ v1 (\ a1
(decl_atom (<=_Real (a_var_real b) (+_Real (a_var_real a) (a_var_real x))) (\ v2 (\ a2
(decl_atom (= Real (a_var_real x) (a_real 0/1)) (\ v3 (\ a3
(decl_atom (= Real (a_var_real a) (a_var_real b)) (\ v4 (\ a4
(decl_atom (= Real (apply _ _ f (a_var_real a)) (apply _ _ f (a_var_real b))) (\ v5 (\ a5
(decl_atom (= Bool (apply _ _ q (a_var_real a)) btrue)) (\ v6 (\ a6
(decl_atom (= Bool (apply _ _ q (+_Real (a_var_real b) (a_var_real x))) bfalse)) (\ v7 (\ a7
(decl_atom (<=_Real (a_var_real b) (a_var_real a)) (\ v8 (\ a8
(decl_atom (= Real (a_var_real a) (+_Real (a_var_real b) (a_var_real x))) (\ v9 (\ a9
(decl_atom (and (= Bool (apply _ _ q (a_var_real a)) btrue)
    (= Bool (apply _ _ q (+_Real (a_var_real b) (a_var_real x))) bfalse))
    (\ v10 (\ a10
CVC4 proof (2/3)

; CNFication
(satlem _ _ (asf _ _ _ a1 (\ l1 (clausify_false (contra _ @F1 l1))))) (\ C1
(satlem _ _ (asf _ _ _ a2 (\ l2 (clausify_false (contra _ @F2 l2))))) (\ C2
(satlem _ _ (asf _ _ _ a3 (\ l3 (clausify_false (contra _ @F3 l3))))) (\ C3
(satlem _ _ (ast _ _ _ a5 (\ l5 (asf _ _ _ a6 (\ l6 (clausify_false (contra _
 (and_elim_1 _ _ (or_elim_1 _ _ (not_not_intro _ @F1 l1)))))))) (\ C4
(satlem _ _ (ast _ _ _ a5 (\ l5 (asf _ _ _ a7 (\ l7 (clausify_false (contra _
 (and_elim_2 _ _ (or_elim_1 _ _ (not_not_intro _ @F1 l1)))))))))) (\ C5

; Theory lemmas
; ~a4 ^ a1 ^ a8 => false
(satlem _ _ (asf _ _ _ a4 (\ l4 (ast _ _ _ a1 (\ l1 (ast _ _ _ a8 (\ l8
(clausify_false (contra _ l1
(or_elim_1 _ _ (not_not_intro _ (<=_to_>_Real _ _ l8)) (not_=_to>_=_<Real _ _ 18)))))))))) (\ C6
; a2 ^ a3 ^ ~a8 => false
(satlem _ _ (ast _ _ _ a2 (\ l2 (ast _ _ _ a3 (\ l3 (asf _ _ _ a8 (\ l8
(poly_norm_> _ _ _ (=to>_=Real _ _ 12) (pn_ _ _ _ _ _ _ (pn_ _ _ _ _ _
 (pn_var a) (pn_var x)) (pn_var b)) (pn2
(poly_norm=_ _ _ (symm _ _ _ _ _ _ (pn_const 0/1) (pn_var x)) (pn3
(poly_norm>_ _ _ (not<=to>_Real _ _ 18) (pn_ _ _ _ _ _ (pn_var b) (pn_var a)) (pn8
(lra_contra>_ _ (lra_add>_ _ _ _ _ _ (lra_add_= _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (pn3 pn2)))))))))))(\ C7
; a4 ^ ~a5 => false
(satlem _ _ (ast _ _ _ a4 (\ l4 (asf _ _ _ a5 (\ l5 (clausify_false
(contra _ (cong _ _ _ _ _ _ (refl _ f) l4) l5))))) (\ C8
CVC4 proof (3/3)

; a3 ^ a4 ^ ~a9 => false
(satlem _ _ (ast _ _ _ a3 \ 13 (ast _ _ _ a4 \ 14 (asf _ _ _ a9 \ 19 (clausify_false
(poly_norm_=_ _ _ (symm _ _ _ 13) (pn_=_ _ _ _ _ _ (pn_const 0/1) (pn_var x)) \ pn3
(poly_norm_=_ _ _ 14 (pn_=_ _ _ _ _ _ (pn_var a) (pn_var b)) \ pn4
(poly_norm_distinct _ _ _ 19 (pn_=_ _ _ _ _ _ (pn_+_ _ _ _ _ _
(pn_var b) (pn_var x)) (pn_var a)) \ pn9
(lra_contra_distinct _ (lra_add_=_distinct _ _
(lra_add_=_ _ _ _ _ _ (pn3 pn4) pn9)))))))))) ) (\ C9
; a9 ^ a6 ^ a7 => false
(satlem _ _ (ast _ _ _ a9 \ 19 (ast _ _ _ a6 \ 16 (ast _ _ _ a7 \ 17 (clausify_false
(contra _ (trans _ _ _ _ _ (trans _ _ _ _ _ (symm _ _ _ 16) (cong _ _ _ _ _ _
(refl _ q) 19)) 17) b_true_not_false)))) ) (\ C10

; Resolution proof
(satlem_simplify _ _ _ (R _ _ (Q _ _ (Q _ _ C6 C1 v1) (Q _ _ (Q _ _ C7 C2 v2) C3 v3) v8)
(Q _ _ (Q _ _ (Q _ _ (R _ _ C9 C10 v9) C3 v3) C4 v6) C5 v7) C8 v5) v4)
(\ x x)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))}
Outline

Introduction

Proofs for SAT
  Prerequisites
  SAT and proofs
  Proof formats
Compression (1/2)

Why?

- Smaller footprint for proof objects (certificates)
- Smaller cores
- Better interpolants
- Easier proof exchange

SAT: around 20% to 40% size reduction (after pruning, DAGification)
SMT: first attempts, but nothing convincing (yet?)
Some techniques

▶ Recycle Pivot with Intersection\(^5,6\)
▶ Lower Units\(^6\)
▶ Splitting\(^7\)

Some tools

▶ Skeptik\(^8\)
▶ PeRIPLO\(^9\)

---


\(^7\) Cotton. *Two Techniques for Minimizing Resolution Proofs*. SAT’10.


Regularity and redundancy

- Irregularity: same pivot several times on one path.
- Irregularity = redundancy
- Irregular proof can be made regular
- Regular resolution proofs in worst case exponentially larger than unrestricted resolution (duplications of part of the DAG)
- Compression idea: partial regularization, when inexpensive
Regularization: Recycle Pivot (with Intersection) RP(I)

- Find irregular node (i.e. same pivot $\ell$ closer to root)
- Replace it by one of its parents
- Fix the proof
  - New node contains $\ell$ or $\bar{\ell}$
    - Resolved further down
  - Many other literals may have disappeared
- Linear with some care
- First bottom-up traversal to
  -annotate nodes with pivots below
  -delete unnecessary parent
- Top-down traversal to fix
  -replace irregular node by preserved parent
  -fix the proof downwards
Regularization: RPI, an exercise

\[
\begin{align*}
\overline{a} \lor \overline{c} \lor \overline{d} & \quad \overline{a} \lor \overline{c} \lor d \\
\overline{a} \lor \overline{c} & \quad \overline{a} \lor \overline{c} \\
a \lor \overline{c} & \quad a \lor c \\
\overline{c} & \\

da \lor \overline{b} \lor c & \quad a \lor \overline{b} \\
\overline{a} & \quad a \lor \overline{b} \\
\overline{b} & \quad \overline{a} \lor b \\
a \lor c & \\
\overline{a} & \quad a \\
\overline{b} & \quad \overline{a} \\
b & \quad \overline{b} \\
\square & \\
\end{align*}
\]
Regularization: RPI, an exercise

\[ \overline{a} \lor c \lor \overline{d} \quad \overline{a} \lor c \lor d \]

\[ \overline{a} \lor \overline{c} \quad a \lor \overline{c} \]

\[ a \lor \overline{b} \lor c \quad \overline{c}^a \]

\[ a \lor \overline{b}^{a,\overline{b}} \quad a \lor b \]

\[ a \quad a^{a,b} \]

\[ \overline{a} \quad \overline{b} \quad \overline{b} \]

\[ b \quad b \]
Regularization: RPI, an exercise
Lower Units

\[
\begin{align*}
\bar{b} \lor \bar{c} & \quad c & \quad a \lor b \lor \bar{c} \\
\bar{b} & \quad a \lor b \\
\bar{a} \lor \bar{b} & \quad a & \quad \bar{a} \lor b \\
\bar{b} & \quad b
\end{align*}
\]
Lower Units

\[ \overline{b} \lor \overline{c} \]
\[ \overline{b} \]
\[ a \lor b \]
\[ a \]
\[ b \]
\[ a \lor b \]
\[ \overline{a} \lor b \]
\[ \overline{b} \]
\[ b \]
\[ a \lor b \]
\[ a \lor b \]
Lower Units

\[
\begin{array}{c}
\overline{b} \lor \overline{c} \quad \ast \quad a \lor b \lor \overline{c} \\
\overline{b} \quad \ast \quad a \lor b \\
\overline{a} \lor \overline{b} \quad \ast \quad \overline{a} \lor b \\
\overline{b} \quad \ast \quad b \\
\end{array}
\]
Lower Units

c

\[ \overline{b} \lor \overline{c} \quad a \lor b \lor \overline{c} \]

\[ a \lor \overline{c} \]

\[ \overline{a} \lor \overline{b} \quad * \quad \overline{a} \lor b \]

\[ \overline{b} \quad * \quad b \]

\[ \Box \]
Lower Units

\[\begin{align*}
\bar{c} & \quad \bar{b} \lor \bar{c} & \quad a \lor b \lor \bar{c} \\
\bar{a} \lor \bar{b} & \quad \bar{a} \lor b & \quad \bar{a} \lor b \\
\bar{a} & \quad \bar{a} \\
\end{align*}\]
Lower Units
Splitting

\[ \overline{a} \lor \overline{c} \quad a \lor \overline{c} \]

\[ \overline{c} \quad a \lor c \]

\[ \overline{a} \lor \overline{b} \quad a \quad \overline{a} \lor b \]

\[ \overline{b} \quad b \]
Splitting

\[ \overline{a} \lor \overline{c} \quad a \lor \overline{c} \]
\[ \overline{c} \quad a \lor c \]
\[ \overline{a} \lor \overline{b} \quad a \quad \overline{a} \lor b \]
\[ \overline{b} \quad b \]

\[ \overline{a} \lor \overline{c} \quad a \lor \overline{c} \]
\[ \overline{c} \quad a \lor c \]
\[ \overline{a} \lor \overline{b} \quad a \quad \overline{a} \lor b \]
\[ \overline{b} \quad b \]

□
Splitting

Keep the $\overline{a}$ branch

Keep the $a$ branch
Splitting

Keep the $\overline{a}$ branch

$\overline{a} \lor \overline{b}$

$\overline{b}$

$*$

$b$

Keep the $a$ branch

$\overline{a} \lor \overline{c}$

$a \lor \overline{c}$

$\overline{c}$

$a \lor c$

$a$

$\overline{a} \lor b$

$\overline{b}$

$*$

$b$

$\overline{a} \lor b$

$a$

$b$
Splitting

Keep the $\overline{a}$ branch

Keep the $a$ branch

\[
\begin{align*}
\overline{a} \lor \overline{b} & \quad \Downarrow \quad \star \quad \overline{a} \lor b \\
\overline{b} & \quad \Downarrow \quad b \quad \Downarrow \quad \square
\end{align*}
\]

\[
\begin{align*}
\star & \quad \Downarrow \quad a \lor \overline{c} \\
\overline{c} & \quad \Downarrow \quad a \lor c \\
\star & \quad \Downarrow \quad a \quad \Downarrow \quad \square
\end{align*}
\]
Splitting

\[ \bar{a} \lor \bar{b} \]

\[ \bar{b} \]

\[ b \]

\[ \square \]

\[ \bar{a} \lor b \]

\[ a \lor \bar{c} \]

\[ \bar{c} \]

\[ a \lor c \]

\[ a \]

\[ \bar{b} \]

\[ \square \]

\[ b \]

\[ \square \]
Splitting

\[ \overline{a} \lor \overline{b} \]

\[ \overline{a} \lor b \]

\[ \overline{a} \]

[Diagram of repair process with logical expressions]
Splitting

\[ \overline{a} \lor \overline{b} \]
\[ \overline{a} \lor b \]
\[ \overline{a} \]

\[ a \lor \overline{c} \]
\[ a \lor c \]
\[ \overline{b} \]
\[ b \]
\[ \square \]
\[ * \]
Splitting

\[
\overline{a} \lor \overline{b} \quad \overline{a} \lor b
\]

\[
\begin{align*}
\overline{a} \\
&\searrow \\
&\swarrow
\end{align*}
\]
Splitting

Combine

\[ \overline{a} \lor \overline{b} \quad \overline{a} \lor b \quad a \lor \overline{c} \quad a \lor c \]
Core compression

Core compression by successive computation of cores\textsuperscript{10}

1: \textbf{function} \textsc{get\_small\_core}(S')
2: \hspace{1em} \textbf{repeat}
3: \hspace{2em} S ← S'
4: \hspace{2em} S' ← \textsc{unsat\_core}(S)
5: \hspace{1em} \textbf{until} S' = S
6: \hspace{1em} \textbf{return} S

Minimal core computation\textsuperscript{11}

1: \textbf{function} \textsc{get\_min\_core}(S)
2: \hspace{1em} \textbf{while} \exists C ∈ S ∧ ¬\textsc{marked}(C) \textbf{do}
3: \hspace{2em} \textbf{choose} C ∈ S ∧ ¬\textsc{marked}(C)
4: \hspace{2em} \textbf{if} \textsc{sat}(S \setminus \{C\}) \textbf{then}
5: \hspace{3em} \textsc{mark}(C)
6: \hspace{2em} \textbf{else}
7: \hspace{3em} S ← \textsc{unsat\_core}(S \setminus \{C\})
8: \hspace{1em} \textbf{return} S

\textsuperscript{10}Zhang, Malik. \textit{Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications}. DATE 2003.

\textsuperscript{11}Dershowitz, Hanna, Nadel. \textit{A Scalable Algorithm for Minimal Unsatisfiable Core Extraction}. SAT 2006.
Outline

Introduction

Proofs for SAT
  Prerequisites
  SAT and proofs
  Proof formats
Applications

- Proof reconstruction within skeptical proof assistants \(^{12, 13, 14}\)
- Interpolant generation \(^{15, 16, 17}\)
- Unsat core computation \(^{18}\)

---


\(^{13}\) Armand, Faure, Grégoire, Keller, Thery, Werner. *A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses*. CPP '11.

\(^{14}\) Böhme. *Proof Reconstruction for Z3 in Isabelle/HOL*. SMT'09.

\(^{15}\) Reynolds, Tinelli, Hadarean. *Certified Interpolant Generation for EUF*. SMT '11.


\(^{17}\) McMillan. *Interpolants from Z3 Proofs*. FMCAD '11.

Proofs and SMT: a history
First Attempts

- Cooperating Validity Checker (CVC), 2002\textsuperscript{19}
  - First SMT solver to attempt proof-production
  - Motivation: independently certify results
  - Tool to find and correct bugs
  - Highly beneficial side effect: improvement in conflict clause production

\textsuperscript{19}Stump, Barrett, Dill. \textit{CVC: A Cooperating Validity Checker}. CAV ’02.
Proofs and SMT: a history
Communication with skeptical proof assistants

- **CVC Lite, 2005**
  - Successor to CVC, ad hoc proof format
  - Translator from proof format to HOL Light
  - Provide access to efficient decision procedures within HOL Light
  - And enable use of HOL Light as a proof-checker for CVC Lite

- **haRVey, 2006**
  - Integration with Isabelle/HOL

- **CVC3, 2008**
  - Effort to certify SMT-LIB benchmark library
  - Found benchmarks with incorrect status
  - Found bug in CVC3

---

20 McLaughlin, Barrett, Ge. *Cooperating Theorem Provers: A Case Study Combining HOL-Light and CVC Lite*. PDPAR '05.

21 Fontaine, Marion, Merz, Nieto, Tiu. *Expressiveness + Automation + Soundness: Towards Combining SMT Solvers and Interactive Proof Assistants*. TACAS '06.

Proofs and SMT: a history
Additional solvers support proofs

- Fx7, 2008<sup>23</sup>
  - Quantified reasoning, custom proof-checker
- MathSAT4, 2008<sup>24</sup>
  - Internal proof engine for unsat cores and interpolants
- Z3, 2008<sup>25</sup>
  - Proof traces - single rule for theory lemmas
- veriT, 2009<sup>26</sup>
  - Proof production a primary goal in veriT

---

<sup>25</sup> de Moura, Bjørner. *Proofs and Refutations, and Z3*. LPAR '08.
<sup>26</sup> Bouton, de Oliveira, Déharbe, Fontaine. *veriT: An Open, Trustable and Efficient SMT-Solver*. CADE '09.
Proofs and SMT: a history

Current Status

- No agreed-upon format for proofs in SMT
- Solvers targeting self-contained, independently-checkable proofs
  - CVC4, veriT
- Proof traces
  - Z3
- Solvers using proof technology to drive other features (e.g. interpolants)
  - MathSAT, SMTInterpol
Challenges

- Challenge to collect and store proof information efficiently
- Producing proofs for sophisticated preprocessing techniques
- Producing proofs for modules that use external tools
- Standardizing a proof format
Outline

Introduction

Proofs for SAT
  Prerequisites
  SAT and proofs
  Proof formats
Reason to believe in proofs

Proofs will play a crucial role

- IC3 seems to be quite a step forward in the state of the art of model checking
- Big step forward for proof assistants also, thanks to ATP: Sledgehammer\textsuperscript{27}, SMTCoq\textsuperscript{28}
- Framework to exchange proofs
- Proofs are required for SAT (SAT competition)
- “Proofs” are required for FOL theorem provers (CASC)
- Bugs regularly remind us: proofs are important, we should (only) believe in proofs

\textsuperscript{27}Blanchette, Böhme, Paulson. \textit{Extending Sledgehammer with SMT solvers.} JAR 2013.

\textsuperscript{28}Armand, Faure, Grégoire, Keller, Théry, Werner. \textit{A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses.} CPP 2011.
To know more

- APPA 2014
- Dagstuhl seminar 15381, September 2015
  
  http://www.dagstuhl.de/de/programm/kalender/semhp/?seminr=15381
Thanks

Some material in these slides from:

- Marijn Heule. *Satisfiability Solvers*. APPA 2014
- Offer Strichman. *SAT/SMT Summer School*. 2014