MiniSat

SAT Algorithms and Applications

Niklas Sörensson
nik@cs.chalmers.se

Chalmers University of Technology and Göteborg University

Empirically Successful Classical Automated Reasoning
a CADE-20 Workshop
22nd - 23th July, 2005
**MiniSat** is a SAT solver with the following features:

- Simple, well documented implementation suitable for educational purposes
- Incremental SAT-solving
- Well-defined interface for general boolean constraints
Empirically Successful

SAT competition 2005 results:

- **MINISAT** won all industrial categories
- **MINISAT** also performed best overall
Empirically Successful

Niklas Sörensson

MINISAT
Overview

- Algorithms for SAT
- Applications
DPLL SAT solving

- Branching
- Unit propagation
- Backtracking
DPLL SAT solving

- Branching
- Unit propagation
- Backtracking
- Learning
DPLL SAT solving

- Branching
- Unit propagation
- Backtracking
- Learning

\[ \neg a \quad a \]
\[ \neg b \quad b \]
\[ \neg c \quad c \]
\[ \bot \]

\[ C_1 \]
DPLL SAT solving

- Branching
- Unit propagation
- Backtracking
- Learning

Analyze the conflict to infer a clause $C_1$ that is a logical consequence of the problem.
DPLL SAT solving

- Branching
- Unit propagation
- Backtracking
- Learning

Analyze the conflict to infer a clause $C_1$ that is a logical consequence of the problem.
Propagation & Conflicts

Unit propagation

Is realized by particular datastructures rather than explicit inferences. No clauses are changed.
Introduction

Background

Propagation & Conflicts

Unit propagation

Is realized by particular datastructures rather than explicit inferences. No clauses are changed.

Conflict

A conflict is a situation where at least one conflicting clause is falsified by unit propagation
Conflict Clause

- Infered by conflict analysis
- Helps prune future parts of the search space
- Actually drives backtracking

Requirements

- Consequence of the clause set
- Falsified by current assignment
- Contains exactly one literal implied by last assumption
### Conflict Analysis Example

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e = F</td>
<td>assumption</td>
</tr>
<tr>
<td>f = F</td>
<td>¬f ∨ e</td>
</tr>
<tr>
<td>g = F</td>
<td>¬g ∨ f</td>
</tr>
<tr>
<td>h = F</td>
<td>¬h ∨ g</td>
</tr>
<tr>
<td>a = T</td>
<td>assumption</td>
</tr>
<tr>
<td>b = T</td>
<td>b ∨ ¬a ∨ e</td>
</tr>
<tr>
<td>c = T</td>
<td>c ∨ e ∨ f</td>
</tr>
<tr>
<td>d = T</td>
<td>d ∨ ¬b ∨ h</td>
</tr>
</tbody>
</table>

![Diagram of conflict analysis example with variables a, b, c, d, e, f, g, h and their truth values.](image-url)
## Conflict Analysis Example

### Assumptions

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>F</td>
</tr>
<tr>
<td>f</td>
<td>F</td>
</tr>
<tr>
<td>g</td>
<td>F</td>
</tr>
<tr>
<td>h</td>
<td>F</td>
</tr>
<tr>
<td>a</td>
<td>T</td>
</tr>
<tr>
<td>b</td>
<td>T</td>
</tr>
<tr>
<td>c</td>
<td>T</td>
</tr>
<tr>
<td>d</td>
<td>T</td>
</tr>
</tbody>
</table>

### Formulas

- \( \neg e \)
- \( a \)
- \( \neg b \lor \neg c \lor \neg d \)
### Conflict Analysis Example

<table>
<thead>
<tr>
<th>Assumption</th>
<th>( \neg b \lor \neg c \lor \neg d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a = T )</td>
<td>( a \lor \neg b \lor \neg c \lor \neg d )</td>
</tr>
<tr>
<td>( b = T )</td>
<td>( b \lor \neg a \lor e \lor f \lor \neg b \lor h )</td>
</tr>
<tr>
<td>( c = T )</td>
<td>( c \lor e \lor f \lor \neg b \lor h )</td>
</tr>
<tr>
<td>( d = T )</td>
<td>( d \lor \neg b \lor h )</td>
</tr>
<tr>
<td>( e = F )</td>
<td>( \top )</td>
</tr>
<tr>
<td>( f = F )</td>
<td>( \neg f \lor e )</td>
</tr>
<tr>
<td>( g = F )</td>
<td>( \neg g \lor f )</td>
</tr>
<tr>
<td>( h = F )</td>
<td>( \neg h \lor g )</td>
</tr>
</tbody>
</table>

Niklas Sörensson

MINI-SAT
### Conflict Analysis Example

<table>
<thead>
<tr>
<th>e = F</th>
<th>assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>f = F</td>
<td>¬f ∨ e</td>
</tr>
<tr>
<td>g = F</td>
<td>¬g ∨ f</td>
</tr>
<tr>
<td>h = F</td>
<td>¬h ∨ g</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a = T</th>
<th>assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>b = T</td>
<td>b ∨ ¬a ∨ e</td>
</tr>
<tr>
<td>c = T</td>
<td>c ∨ e ∨ f</td>
</tr>
<tr>
<td>d = T</td>
<td>d ∨ ¬b ∨ h</td>
</tr>
</tbody>
</table>

\[ ¬b ∨ ¬c ∨ ¬d \]
\[ ¬b ∨ ¬c ∨ h \]
Conflict Analysis Example

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e=F</td>
<td>assumption</td>
</tr>
<tr>
<td>f=F</td>
<td>¬f ∨ e</td>
</tr>
<tr>
<td>g=F</td>
<td>¬g ∨ f</td>
</tr>
<tr>
<td>h=F</td>
<td>¬h ∨ g</td>
</tr>
<tr>
<td>a=T</td>
<td>assumption</td>
</tr>
<tr>
<td>b=T</td>
<td>b ∨ ¬a ∨ e</td>
</tr>
<tr>
<td>c=T</td>
<td>c ∨ e ∨ f</td>
</tr>
<tr>
<td>d=T</td>
<td>d ∨ ¬b ∨ h</td>
</tr>
</tbody>
</table>

¬b ∨ ¬c ∨ ¬d
¬b ∨ ¬c ∨ h
## Conflict Analysis Example

<table>
<thead>
<tr>
<th>:</th>
<th>:</th>
</tr>
</thead>
<tbody>
<tr>
<td>e=F</td>
<td>assumption</td>
</tr>
<tr>
<td>f=F</td>
<td>( \neg f \lor e )</td>
</tr>
<tr>
<td>g=F</td>
<td>( \neg g \lor f )</td>
</tr>
<tr>
<td>h=F</td>
<td>( \neg h \lor g )</td>
</tr>
<tr>
<td>a=T</td>
<td>assumption</td>
</tr>
<tr>
<td>b=T</td>
<td>( b \lor \neg a \lor e )</td>
</tr>
<tr>
<td>c=T</td>
<td>( c \lor e \lor f )</td>
</tr>
<tr>
<td>d=T</td>
<td>( d \lor \neg b \lor h )</td>
</tr>
</tbody>
</table>

\[ \neg b \lor \neg c \lor \neg d \]
\[ \neg b \lor \neg c \lor h \]
\[ \neg b \lor h \lor e \lor f \]
Conflict Analysis Example

- \( e = F \)
- \( f = F \)
- \( g = F \)
- \( h = F \)
- \( a = T \)
- \( b = T \)
- \( c = T \)
- \( d = T \)

Assumption

- \( e \)
- \( f \)
- \( g \)
- \( h \)

Negation

- \( \neg b \lor \neg c \lor \neg d \)
- \( \neg b \lor \neg c \lor h \)
- \( \neg b \lor h \lor e \lor f \)
Conflict Analysis Algorithm

1. Begin with conflicting clause
2. Resolve on the most recently propagated literal, using the clause that caused the propagation as side clause
3. Repeat until the candidate clause contains exactly one literal implied by the last assumption
Alternative Conflict Analyses

- Traditional Conflict Analysis is minimal in the number of derivations
- Balance between time spent and usefulness of the conflict clause
- Is a shorter clause always better?
Subsumption Resolution

\[
\frac{a \lor C \quad \neg a \lor D}{D} \quad C \subseteq D
\]

Conclusion clause subsumes one of the antecedents
## Basic Conflict Minimizing Example

<table>
<thead>
<tr>
<th>:</th>
<th>:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e = F$</td>
<td><em>assumption</em> $\neg f \lor e$</td>
</tr>
<tr>
<td>$f = F$</td>
<td>$\neg g \lor f$</td>
</tr>
<tr>
<td>$g = F$</td>
<td>$\neg h \lor g$</td>
</tr>
<tr>
<td>$h = F$</td>
<td>$\neg b \lor \neg c \lor \neg d$</td>
</tr>
<tr>
<td>$a = T$</td>
<td><em>assumption</em> $b \lor \neg a \lor e$</td>
</tr>
<tr>
<td>$b = T$</td>
<td>$\neg b \lor \neg c \lor h$</td>
</tr>
<tr>
<td>$c = T$</td>
<td>$\neg b \lor h \lor e \lor f$</td>
</tr>
<tr>
<td>$d = T$</td>
<td>$d \lor \neg b \lor h$</td>
</tr>
</tbody>
</table>

*Diagram*:

- $\neg e$
- $a$
- $\neg b$
- $\bot$

- $\neg b \lor \neg c \lor \neg d$
- $\neg b \lor \neg c \lor h$
- $\neg b \lor h \lor e \lor f$
Basic Conflict Minimizing Example

\[ e = F \]
\[ f = F \]
\[ g = F \]
\[ h = F \]
\[ a = T \]
\[ b = T \]
\[ c = T \]
\[ d = T \]

assumption

\[ \neg b \lor \neg c \lor \neg d \]
\[ \neg b \lor \neg c \lor h \]
\[ \neg b \lor h \lor e \lor f \]
# Basic Conflict Minimizing Example

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e=F</td>
<td><strong>assumption</strong></td>
</tr>
<tr>
<td>f=F</td>
<td>¬f ∨ e</td>
</tr>
<tr>
<td>g=F</td>
<td>¬g ∨ f</td>
</tr>
<tr>
<td>h=F</td>
<td>¬h ∨ g</td>
</tr>
<tr>
<td>a=T</td>
<td><strong>assumption</strong></td>
</tr>
<tr>
<td>b=T</td>
<td>b ∨ ¬a ∨ e</td>
</tr>
<tr>
<td>c=T</td>
<td>c ∨ e ∨ f</td>
</tr>
<tr>
<td>d=T</td>
<td>d ∨ ¬b ∨ h</td>
</tr>
</tbody>
</table>

\[
\neg b \lor \neg c \lor \neg d \\
\neg b \lor \neg c \lor h \\
\neg b \lor h \lor e \lor f \\
\neg b \lor h \lor e 
\]
Basic Conflict Minimizing

- Start from ordinary conflict clause
- Apply subsumption resolution greedily
- Works because there are no cyclic dependencies
- Also uses reason clauses from other levels

Very cheap, almost for free.
Generalized Subsumption Resolution

\[
\frac{C_1}{C_2} \quad \frac{D_1}{C_2} \quad \frac{C_2}{C_3} \quad \frac{D_2}{C_3} \quad \ldots \quad \frac{C_{n-1}}{C_n} \quad \frac{D_{n-1}}{C_n}
\]

where \( C_n \) subsumes \( C_1 \)
Full Conflict Minimizing Example

\[
\begin{align*}
& e = F \\
& f = F \\
& g = F \\
& h = F \\
& a = T \\
& b = T \\
& c = T \\
& d = T \\
\end{align*}
\]

\[
\begin{align*}
& \text{assumption} \\
& \neg f \lor e \\
& \neg g \lor f \\
& \neg h \lor g \\
\end{align*}
\]

\[
\begin{align*}
& \neg b \lor \neg c \lor \neg d \\
& \neg b \lor \neg c \lor h \\
& \neg b \lor h \lor e \lor f \\
\end{align*}
\]
Full Conflict Minimizing Example

- \( e = \neg F \)
- \( f = \neg F \)
- \( g = \neg F \)
- \( h = \neg F \)
- \( a = T \)
- \( b = T \)
- \( c = T \)
- \( d = T \)

- \( e = \text{assumption} \) \( \neg f \lor e \)
- \( f = \neg F \)
- \( g = \neg F \)
- \( h = \neg F \)
- \( a = \text{assumption} \) \( \neg b \lor \neg c \lor \neg d \)
- \( b = T \)
- \( c = T \)
- \( d = T \)
- \( b \lor \neg a \lor e \)
- \( c \lor e \lor f \)
- \( b \lor h \lor e \lor f \)

Niklas Sörensson

MINISAT
**Full Conflict Minimizing Example**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e=F</td>
<td>assumption</td>
</tr>
<tr>
<td>f=F</td>
<td>¬f ∨ e</td>
</tr>
<tr>
<td>g=F</td>
<td>¬g ∨ f</td>
</tr>
<tr>
<td>h=F</td>
<td>¬h ∨ g</td>
</tr>
<tr>
<td>a=T</td>
<td>assumption</td>
</tr>
<tr>
<td>b=T</td>
<td>b ∨ ¬a ∨ e</td>
</tr>
<tr>
<td>c=T</td>
<td>c ∨ e ∨ f</td>
</tr>
<tr>
<td>d=T</td>
<td>d ∨ ¬b ∨ h</td>
</tr>
</tbody>
</table>

```
¬b ∨ ¬c ∨ ¬d
¬b ∨ ¬c ∨ h
¬b ∨ h ∨ e ∨ f
¬b ∨ e ∨ f ∨ g
```
### Full Conflict Minimizing Example

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$e = F$</td>
<td>assumption</td>
</tr>
<tr>
<td>$f = F$</td>
<td>$\neg f \lor e$</td>
</tr>
<tr>
<td>$g = F$</td>
<td>$\neg g \lor f$</td>
</tr>
<tr>
<td>$h = F$</td>
<td>$\neg h \lor g$</td>
</tr>
<tr>
<td>$a = T$</td>
<td>assumption</td>
</tr>
<tr>
<td>$b = T$</td>
<td>$b \lor \neg a \lor e$</td>
</tr>
<tr>
<td>$c = T$</td>
<td>$c \lor e \lor f$</td>
</tr>
<tr>
<td>$d = T$</td>
<td>$d \lor \neg b \lor h$</td>
</tr>
</tbody>
</table>

```
¬b ∨ ¬c ∨ ¬d
¬b ∨ ¬c ∨ h
¬b ∨ h ∨ e ∨ f
¬b ∨ e ∨ f ∨ g
```
## Full Conflict Minimizing Example

<table>
<thead>
<tr>
<th>e = F</th>
<th>assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>f = F</td>
<td>¬f ∨ e</td>
</tr>
<tr>
<td>g = F</td>
<td>¬g ∨ f</td>
</tr>
<tr>
<td>h = F</td>
<td>¬h ∨ g</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a = T</th>
<th>assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>b = T</td>
<td>b ∨ ¬a ∨ e</td>
</tr>
<tr>
<td>c = T</td>
<td>c ∨ e ∨ f</td>
</tr>
<tr>
<td>d = T</td>
<td>d ∨ ¬b ∨ h</td>
</tr>
</tbody>
</table>

| ¬b ∨ ¬c ∨ ¬d |
| ¬b ∨ ¬c ∨ h |
| ¬b ∨ h ∨ e ∨ f |
| ¬b ∨ e ∨ f ∨ g |
| ¬b ∨ e ∨ f |
## Full Conflict Minimizing Example

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e=F</td>
<td>assumption</td>
</tr>
<tr>
<td>f=F</td>
<td>¬f ∨ e</td>
</tr>
<tr>
<td>g=F</td>
<td>¬g ∨ f</td>
</tr>
<tr>
<td>h=F</td>
<td>¬h ∨ g</td>
</tr>
<tr>
<td>a=T</td>
<td>assumption</td>
</tr>
<tr>
<td>b=T</td>
<td>b ∨ ¬a ∨ e</td>
</tr>
<tr>
<td>c=T</td>
<td>c ∨ e ∨ f</td>
</tr>
<tr>
<td>d=T</td>
<td>d ∨ ¬b ∨ h</td>
</tr>
</tbody>
</table>

\[
\neg b \lor \neg c \lor \neg d \\
\neg b \lor \neg c \lor h \\
\neg b \lor h \lor e \lor f \\
\neg b \lor e \lor f \lor g \\
\neg b \lor e \lor f
\]
Full Conflict Minimizing Example

\[
\begin{align*}
\text{e} &= F \\
\text{f} &= F \\
\text{g} &= F \\
\text{h} &= F \\
\text{a} &= T \\
\text{b} &= T \\
\text{c} &= T \\
\text{d} &= T \\
\end{align*}
\]

\[
\begin{align*}
\text{assumption} & \quad \neg f \lor e \\
\text{assumption} & \quad b \lor \neg a \lor e \\
\text{c} & \quad e \lor f \\
\text{d} & \quad d \lor \neg b \lor h \\
\neg b \lor \neg c \lor \neg d \\
\neg b \lor \neg c \lor h \\
\neg b \lor h \lor e \lor f \\
\neg b \lor e \lor f \lor g \\
\neg b \lor e \lor f \\
\neg b \lor e \\
\end{align*}
\]
Full Conflict Minimizing

- Search for applications of generalized subsumption
- Caching of subresults is necessary to make it efficient

Is relatively expensive, but simplifies a lot
## Effect of Minimization

<table>
<thead>
<tr>
<th></th>
<th>LPC</th>
<th>Time</th>
<th></th>
<th>LPC</th>
<th>Time</th>
<th></th>
<th>LPC</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Orig</strong></td>
<td>47.0</td>
<td>4.90</td>
<td><strong>Orig</strong></td>
<td>37.9</td>
<td>182.0</td>
<td><strong>Orig</strong></td>
<td>174.1</td>
<td>33.273</td>
</tr>
<tr>
<td><strong>Basic</strong></td>
<td>31.3</td>
<td>(7.8%)</td>
<td><strong>Basic</strong></td>
<td>36.2</td>
<td>(11.5%)</td>
<td><strong>Basic</strong></td>
<td>92.1</td>
<td>(12.8%)</td>
</tr>
<tr>
<td><strong>Full</strong></td>
<td>13.2</td>
<td>(4.6%)</td>
<td><strong>Full</strong></td>
<td>16.5</td>
<td>(29.6%)</td>
<td><strong>Full</strong></td>
<td>39.0</td>
<td>(42.1%)</td>
</tr>
</tbody>
</table>
SatELite

- A pre-processing tool developed by Niklas Een
- Uses simplification techniques
  - Variable elimination
  - Subsumption
  - Subsumption resolution
- Works well for simplifying formulas translated from circuits
Effects of using SatElite
Incremental SAT

- Allows to solve a sequence of related problems
- Between runs clauses can be added or unit clauses can be retracted
- Learned clauses are valid in later runs
Safety Property Verification

Symbolic Finite State Machine

- $T(s, s')$ (transition relation)
- $I(s)$ (initial states)

Can a state satisfying a property $\neg P(s)$ be reached?
Temporal Induction

\[ I(s_0) \land \neg P(s_0) \quad \text{(base 1)} \]

\[ P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \quad \text{(ind. 1)} \]

\[ I(s_1) \land P(s_1) \land T(s_1, s_0) \land P(s_0) \quad \text{(base 2)} \]

\[ P(s_2) \land T(s_2, s_1) \land P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \quad \text{(ind. 2)} \]

\[ I(s_2) \land P(s_2) \land T(s_2, s_1) \land P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \quad \text{(base 3)} \]

...
Temporal Induction

\[ I(s_0) \land \neg P(s_0) \quad (base \ 1) \]
**Temporal Induction**

\[
I(s_0) \land \neg P(s_0) \quad (base \ 1)
\]

\[
P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \quad (ind. \ 1)
\]
Temporal Induction

\[ I(s_0) \land \neg P(s_0) \quad \text{(base 1)} \]

\[ P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \quad \text{(ind. 1)} \]

\[ I(s_1) \land P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \quad \text{(base 2)} \]
Temporal Induction

\[ I(s_0) \land \neg P(s_0) \]  \hspace{1cm} (base 1)

\[ P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \]  \hspace{1cm} (ind. 1)

\[ I(s_1) \land P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \]  \hspace{1cm} (base 2)

\[ P(s_2) \land T(s_2, s_1) \land P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \]  \hspace{1cm} (ind. 2)
Temporal Induction

\[ I(s_0) \land \neg P(s_0) \quad (base \ 1) \]
\[ P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \quad (ind. \ 1) \]
\[ I(s_1) \land P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \quad (base \ 2) \]
\[ P(s_2) \land T(s_2, s_1) \land P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \quad (ind. \ 2) \]
\[ I(s_2) \land P(s_2) \land T(s_2, s_1) \land P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \quad (base \ 3) \]
Temporal Induction

\[ I(s_0) \land \neg P(s_0) \]  \hspace{1cm} \text{(base 1)}

\[ P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \]  \hspace{1cm} \text{(ind. 1)}

\[ I(s_1) \land P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \]  \hspace{1cm} \text{(base 2)}

\[ P(s_2) \land T(s_2, s_1) \land P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \]  \hspace{1cm} \text{(ind. 2)}

\[ I(s_2) \land P(s_2) \land T(s_2, s_1) \land P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \]  \hspace{1cm} \text{(base 3)}

\[ \vdots \]
Coding Unique States

For completeness all states in a path is required to be different
Coding Unique States

For completeness all states in a path is required to be different

Problem

Encoding the unique states requirement gives a formula quadratic in number of states
Coding Unique States

For completeness all states in a path is required to be different

**Problem**

Encoding the unique states requirement gives a formula quadratic in number of states

**Solution**

Overhead avoided by lazy addition of uniqueness constraints
Paradox - a First Order Model Finder

- Searches for models with a **finite domain**.
- Uses MINISAT in an incremental way
- Won the SAT* category of CASC 2003 and 2004
When searching for models of size $k$ it is enough to choose the first $k$ natural numbers as the domain.
First Order Model Finding Using SAT

- When searching for models of size $k$ it is enough to choose the first $k$ natural numbers as the domain.

- In an interpretation each ground atom is identified with a propositional variable. Example:

  $$ P(0), \ P(1), \ f(0) = 0, \ f(0) = 1, \ f(1) = 0, \ f(1) = 1 $$
When searching for models of size $k$ it is enough to choose the first $k$ natural numbers as the domain.

In an interpretation each ground atom is identified with a propositional variable. Example:

$$P(0), \quad P(1), \quad f(0) = 0, \quad f(0) = 1, \quad f(1) = 0, \quad f(1) = 1$$

To transform a first order problem to SAT we need to get rid of complex terms, and variables. This is done by function flattening, and ground instantiation.
Function flattening

Shallow Literals are of one of the following forms:

1. $P(x_1, \ldots, x_m)$, or $\neg P(x_1, \ldots, x_m)$,
2. $f(x_1, \ldots, x_n) = y$, or $f(x_1, \ldots, x_n) \neq y$,
3. $x = y$.

Rewrite rules

1. $C[t] \quad \rightarrow \quad t \neq x \lor C[x]$
2. $x \neq y \lor C[x, y] \quad \rightarrow \quad C[x, x]$

Terminates with a shallow clause set, if applied exhaustively
Ground instantiation of a clause set $F$ is done for the particular domain $D = \{1, \ldots, k\}$

1. Instances $F_\sigma$ with respect to $D$
Ground instantiation of a clause set $F$ is done for the particular domain $D = \{1, \ldots, k\}$

1. Instances $F_\sigma$ with respect to $D$
2. Functionality constraints: each function has at most one value in $D$ for each input

Example: $f(1) \neq 1 \lor f(1) \neq 2$
Ground instantiation of a clause set $F$ is done for the particular domain $D = \{1, \ldots, k\}$

1. Instances $F_\sigma$ with respect to $D$
2. Functionality constraints: each function has at most one value in $D$ for each input
   Example: $f(1) \neq 1 \lor f(1) \neq 2$
3. Totality constraints: each function has at least one value in $D$ for each input
   Example: $f(1) = 1 \lor f(1) = 2 \lor f(1) = 3$, for $D = \{1, 2, 3\}$
Non Ground Splitting

Grounding is exponential in the number of variables in a clause. Number of variables can be reduced by splitting.
Grounding is exponential in the number of variables in a clause. Number of variables can be reduced by splitting.

Example: \( P(x, y) \lor Q(y, z) \rightarrow \)

1. \( P(x, y) \lor S(y) \)
2. \( Q(y, z) \lor \neg S(y) \)
Non Ground Splitting

Grounding is exponential in the number of variables in a clause. Number of variables can be reduced by splitting.

Example: $P(x, y) \lor Q(y, z) \rightarrow$

1. $P(x, y) \lor S(y)$
2. $Q(y, z) \lor \neg S(y)$

- Optimally splitting a clause is NP hard
- We use a simple greedy heuristic that works well
Relatively expensive techniques pay off in computing conflict clauses. Variable elimination-based preprocessing helps with industrial problems. Incremental SAT allows solving a sequence of related problems more efficiently. Use incremental architecture for lazy encoding of otherwise infeasible (or expensive) constraints.
Relatively expensive techniques pay off in computing conflict clauses
Summary

- Relatively expensive techniques pay off in computing conflict clauses.
- Variable elimination based preprocessing helps with industrial problems.
Summary

- Relatively expensive techniques pay off in computing conflict clauses
- Variable elimination based preprocessing helps with industrial problems
- Incremental SAT allows to solve a sequence of related problems more efficiently
Summary

- Relatively expensive techniques pay off in computing conflict clauses
- Variable elimination based preprocessing helps with industrial problems
- Incremental SAT allows to solve a sequence of related problems more efficiently
- Use incremental architecture for lazy encoding of otherwise infeasible (or expensive) constraints