Satisfiability, Boolean Modeling and Computation
Spring 2016

Matti Järvisalo

Lecture 7: Minimal unsatisfiability, part II: MUS extraction.
February 9, 2016
On This Lecture

MUS Extraction

- SAT-based approaches to finding MUSes
- Group-MUS extraction
Basic concepts
# MUSes: Recap

## Minimal unsatisfiability

A CNF formula $F$ is *minimally unsatisfiable* (MU) if
- $F \in \text{UNSAT}$, and
- for each clause $C \in F$, the CNF formula $F \setminus \{C\} \in \text{SAT}$.

## Minimal unsatisfiable subset (MUS)

A set of clauses $M$ is an MUS of a CNF formula $F$ if
- $M \subseteq F$, and
- $M$ is an MU.

**MUS($F$):** the set of MUSes of a CNF formula $F$. 

Categorizing Clauses wrt MUS Extraction

- MUS extraction algorithms are based on iteratively identifying clauses that are guaranteed to belong to some MUS.
- Such clauses are called *necessary* or *transition* clauses.
- Different algorithms use different ways of identifying such clauses.
Categorizing Clauses wrt MUS Extraction

**Necessary clauses for a CNF formula** $F$

Clause $C \in F$ is necessary for $F$ if
(i) $F \in$ UNSAT, and (ii) $F \setminus \{C\} \in$ SAT.

$C \in F$ is necessary for $F$ iff
$\exists$ assignment $\tau$ s.t. $\tau(F \setminus \{C\}) = 1$ and $\tau(C) = 0$.

**Properties of necessary clauses**

Necessary clauses ...
- belong to *all* MUSes of $F$: in $\bigcap \text{MUS}(F)$.
- are needed in every Resolution proof of $F$.

If $F$ is MU, then all clauses in $F$ are necessary.

If $C$ is necessary for $F$, it is necessary for every unsatisfiable $F' \subset F$.

Deciding whether given $C$ is necessary for $F$ is NP–complete.
Categorizing Clauses wrt MUS Extraction

Example: necessary clauses

\[ C_1 : (x) \]
\[ C_2 : (\neg x \lor z) \]
\[ C_3 : (y) \]
\[ C_4 : (\neg x \lor \neg y) \]
\[ C_5 : (x \lor y) \]
\[ C_6 : (\neg y \lor \neg z) \]

\[ \text{MUS}(F) = \{ \{C_1, C_3, C_4\}, \{C_1, C_2, C_3, C_6\}\} \]

Necessary clauses:

“Necessary clauses” are a central concept in practical MUS extraction.
Categorizing Clauses wrt MUS Extraction

Example: necessary clauses

\[ C_1 : (x) \]
\[ C_2 : (\neg x \lor z) \]
\[ C_3 : (y) \]
\[ C_4 : (\neg x \lor \neg y) \]
\[ C_5 : (x \lor y) \]
\[ C_6 : (\neg y \lor \neg z) \]

\[ \text{MUS}(F) = \{ \{ C_1, C_3, C_4 \}, \{ C_1, C_2, C_3, C_6 \} \} \]

“Necessary clauses” are a central concept in practical MUS extraction
Categorizing Clauses wrt MUS Extraction

Example: necessary clauses

\begin{align*}
C_1 : & \quad (x) \\
C_2 : & \quad (\neg x \lor z) \\
C_3 : & \quad (y) \\
C_4 : & \quad (\neg x \lor \neg y) \\
C_5 : & \quad (x \lor y) \\
C_6 : & \quad (\neg y \lor \neg z)
\end{align*}

- \text{MUS}(F) = \{\{C_1, C_3, C_4\}, \{C_1, C_2, C_3, C_6\}\}
- \text{Necessary clauses: ?}

“Necessary clauses” are a central concept in practical MUS extraction
Example: necessary clauses

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$ : $(x)$</td>
<td></td>
</tr>
<tr>
<td>$C_2$ : $(\neg x \lor z)$</td>
<td></td>
</tr>
<tr>
<td>$C_3$ : $(y)$</td>
<td></td>
</tr>
<tr>
<td>$C_4$ : $(\neg x \lor \neg y)$</td>
<td>MUS($F$) = {${C_1, C_3, C_4}$, ${C_1, C_2, C_3, C_6}$}</td>
</tr>
<tr>
<td>$C_5$ : $(x \lor y)$</td>
<td>Necessary clauses: $C_1$</td>
</tr>
<tr>
<td>$C_6$ : $(\neg y \lor \neg z)$</td>
<td></td>
</tr>
</tbody>
</table>

“Necessary clauses” are a central concept in practical MUS extraction.
Categorizing Clauses wrt MUS Extraction

Example: necessary clauses

| C_1  | (x)                  |
| C_2  | (¬x ∨ z)             |
| C_3  | (y)                  |
| C_4  | (¬x ∨ ¬y)            |
| C_5  | (x ∨ y)              |
| C_6  | (¬y ∨ ¬z)            |

- \[ \text{MUS}(F) = \{ \{ C_1, C_3, C_4 \}, \{ C_1, C_2, C_3, C_6 \} \} \]
- Necessary clauses: \[ C_1, \ldots ? \]

“Necessary clauses” are a central concept in practical MUS extraction
Categorizing Clauses wrt MUS Extraction

Potentially necessary clauses

Clauses in some but not all MUSes of \( F \).

- May become necessary after removing some clauses in \( F \).
  \( \sim \) may become necessary for Resolution proofs of \( F \).

Example

\[
\begin{align*}
C_1 & : (x) \\
C_2 & : (\neg x \lor z) \\
C_3 & : (y) \\
C_4 & : (\neg x \lor \neg y) \\
C_5 & : (x \lor y) \\
C_6 & : (\neg y \lor \neg z)
\end{align*}
\]

- Remove \( C_4 \)
  \( \sim \) only core left is \( \{ C_1, C_2, C_3, C_6 \} \)

- Remove \( C_3, C_4 \) \( \sim \) SAT.

Unnecessary clauses / never necessary clauses

Clauses that can be removed without making the UNSAT CNF formula satisfiable.
Approaches to MUS Extraction

Main approaches to MUS extraction:
- Deletion-based
- Insertion-based
- Dichotomic (binary search)
- Hybrid

Several types of optimization for different types of algorithms:
- Redundancy checking
- Clause-set refinement
- (Recursive) model rotation
MUS Extraction Algorithms
Preliminaries

SAT($F$) denotes a SAT solver call on CNF formula $F$
- Returns UNSAT or SAT

Recall: necessary (or transition) clauses
- Clause $C$ is necessary for UNSAT $F$ if $F \setminus \{C\} \in \text{SAT}$

<table>
<thead>
<tr>
<th>Deletion-based MUS extraction:</th>
<th>destructive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteratively detect necessary clause on transition from SAT to UNSAT.</td>
<td></td>
</tr>
<tr>
<td>• MUS built by excluding unnecessary clauses.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Insertion-based MUS extraction:</th>
<th>constructive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteratively detect necessary clauses on transition from UNSAT to SAT.</td>
<td></td>
</tr>
<tr>
<td>• MUS built by including necessary clauses.</td>
<td></td>
</tr>
</tbody>
</table>
Approaches to MUS Extraction: Complexity

One metric: the number of SAT solver calls needed in the worst case

In terms of
- size of $F$, i.e., the number $|F|$ of clauses in input CNF formula $F$;
- size of the MUS $M$ of $F$ extracted.

- Deletion-based: $\Theta(|F|)$
- Insertion-based: $O(|M| \cdot |F|)$
- Dichotomic (binary search): $O(|M| \cdot \log |F|)$
Deletion-based MUS Extraction

**Input:** Unsatisfiable CNF formula $F$.

**Output:** An $M \in \text{MUS}(F)$.

1. $M \leftarrow F$
2. for each $C \in M$
   1. if not SAT($M \setminus \{C\}$) then $M \leftarrow M \setminus \{C\}$
3. return $M$

**During execution:**

$M$: over-approximation of an MUS of $F$. 
Insertion-based MUS Extraction

**Input:** Unsatisfiable CNF formula \( F \).
**Output:** An \( M \in \text{MUS}(F) \).

1. \( M \leftarrow \emptyset \)
2. while \( F \neq \emptyset \) do
   1. \( S \leftarrow \emptyset \)
   2. while \( \text{SAT}(M \cup S) \) do
      1. \( C \leftarrow \text{a clause in } F \)
      2. \( S \leftarrow S \cup \{C\} \)
      3. \( M \leftarrow M \cup \{C\} \)
      4. \( F \leftarrow S \setminus \{C\} \)
     \ // \ C \text{ is necessary for } M \cup S 
3. return \( M \)

**During execution:**

**\( M \):** under-approximation of an MUS of \( F \).
**\( S \):** working formula for detecting a necessary clause.
Comparison to deletion-based MUS extraction

- More SAT solver calls in the worst-case
- However:
  - SAT solver calls made starting with small subsets of $F$
  - Transition from SAT to UNSAT:
    SAT solver calls faster — SAT often easier than UNSAT for a solver
  - Potential much faster when MUSes of $F$ are small
Optimizations to Insertion-based MUS Extraction: Redundancy Checking

Definition: Redundant clause

Clause $C \in F$ is redundant in $F$ if $F \setminus \{C\} \models C$.

- Note: $C$ is redundant if $F$ and $F \setminus \{C\}$ are logically equivalent.

Observation

Necessary clauses are *irredundant* for $F \in \text{UNSAT}$.

$\sim$ Every clause in MUS of $F$ is irredundant.

Insertion-based MUS Extraction with Redundancy Checking

Add $C$ to $S$ only if $M \cup S \not\models \{C\}$.

- Check redundancy with SAT solver.
Insertion-based MUS Extraction w/Redundancy Checking

**Input:** Unsatisfiable CNF formula \( F \).

**Output:** An \( M \in \text{MUS}(F) \).

1. \( M \leftarrow \emptyset \)
2. while \( F \neq \emptyset \) do
   1. \( S \leftarrow \emptyset \)
   2. for each \( C \in F \) do
      1. if \( \text{SAT}(M \cup S \cup \{\neg C\}) \) then
         \( S \leftarrow S \cup \{C\} \)
         \( C_{\text{nec}} \leftarrow C \)
      3. \( M \leftarrow M \cup \{C_{\text{nec}}\} \)
      4. \( F \leftarrow S \setminus \{C_{\text{nec}}\} \)
3. return \( M \)

During execution:

\( M \): under-approximation of an MUS of \( F \).
\( S \): working formula for detecting a necessary clause.
\( \{\neg C\} \): \( \bigcup_{l \in C} \{\neg l\} \) (i.e., negations of literals in \( C \) as unit clauses)
Dichotomic MUS Extraction

Idea:
- Build an MUS via under-approximation $M$
- Maintain upper bound $U$ and lower bound $L$ on the size of an MUS
- Initialize $L = 1$, $U = |F|$ (number of clauses in $F$)
- For $middle = \lceil (L + U)/2 \rceil$:
  - If $\text{SAT}\left(\{C_1, \ldots, C_{middle}\}\right)$, let $L \leftarrow middle + 1$
  - Otherwise let $U \leftarrow middle$.
- When $L = U$: $C_L$ is necessary for $M \cup \{C_1, \ldots, C_L\}$
  - $\leadsto$ let $M \leftarrow M \cup \{C_L\}$.
- Return when $M$ becomes unsatisfiable.
Approaches to MUS Extraction: Complexity

One metric: the number of SAT solver calls needed in the worst case

- Deletion-based: $\Theta(|F|)$
- Insertion-based: $O(|M| \cdot |F|)$
- Dichotomic (binary search): $O(|M| \cdot \log |F|)$

However: not the whole story!

- Overall performance also depends on how difficult the SAT solver calls are
Refining MUS Extraction Algorithms
Refinements to MUS Extraction Algorithms

- Redundancy checking
  - Aims at making SAT solver calls easier

- Clause-set refinement
  - Aims at reducing the number of unsatisfiable SAT solver calls

- (Recursive) model rotation
  - Aims at reducing the number of satisfiable SAT solver calls
Hybrid MUS Extraction

**Input:** Unsatisfiable CNF formula $F$.

**Output:** An $M \in \text{MUS}(F)$.

1. $M \leftarrow \emptyset$ // under-approximation

2. while $F \neq \emptyset$ do
   1. $C \leftarrow$ a clause in $F$
   2. $\text{res} \leftarrow \text{SAT}(M \cup (F \setminus \{C\}))$
   3. if $\text{res} = \text{"SAT"}$ then $M \leftarrow M \cup \{C\}$ // $C$ necessary for $M \cup F$
   4. else $F \leftarrow F \setminus \{C\}$ // $C$ not necessary, remove from $F$

3. return $M$

**During execution:**

$M$: under-approximation of an MUS of $F$.

Combines the ideas of refining $F$ and $M$ from deletion and insertion based MUS extraction
Clause-set Refinement for Hybrid MUS Extraction

Takes advantage of the fact that SAT solvers can return UNSAT cores

When the necessity check \( \text{SAT}(M \cup (F \setminus \{C\})) \) fails:

- Obtain also an UNSAT core \( U \) of \( M \cup (F \setminus \{C\}) \)
- Use \( U \) to potentially rule out *multiple* unnecessary clauses, not just \( C \):
  let \( F \leftarrow U \setminus M \).

*Focuses on the obtained UNSAT core*
Hybrid MUS Extraction w/Clause-set Refinement

Input: Unsatisfiable CNF formula $F$.
Output: An $M \in \text{MUS}(F)$.

1. $M \leftarrow \emptyset$ \hspace{1cm} // under-approximation
2. while $F \neq \emptyset$ do
   1. $C \leftarrow$ a clause in $F$
   2. $(\text{res}, U) \leftarrow \text{SAT}(M \cup (F \setminus \{C\}))$
   3. if $\text{res} = "\text{SAT}"$ then $M \leftarrow M \cup \{C\}$ \hspace{1cm} // $C$ necessary for $M \cup F$
   4. else $F \leftarrow U \setminus M$ \hspace{1cm} // Clause-set refinement
3. return $M$
Model Rotation

Fact

Clause $C$ is necessary for $F$ if and only if

- $F \in \text{UNSAT}$, and
- there is an assignment $\tau$ s.t. $\tau(F \setminus \{C\}) = 1$ and $\tau(C) = 0$.

$\tau$ is a witness for the necessity of $C$ for $F$.

Model Rotation

Given witness $\tau$ for $C$:

attempt to locally modify $\tau$ to obtain a different witness for some other clause $C'$.
Model Rotation

Given witness $\tau$ for $C$: attempt to locally modify $\tau$ to obtain a different witness for some other clause $C'$.

In practice

- Try individually flipping the value assigned by $\tau$ to each variable in $C$.
- If one of thus obtained $\tau'$ is a witness for some other clause $C'$ in $F$: $C'$ is also necessary.
  - Very cheap to evaluate each $\tau'$!
- Recurse: try flipping value assigned by $\tau'$ to each variable in $C'$, etc.
- Stop when no new necessary clause is found.

Greatly improves performance in practice by lowering the number of needed SAT solver calls.
MUS Extractors

- MUSer2  most algorithmic variants
  https://bitbucket.org/anton_belov/muser2
- HaifaMUC  deletion-based, manipulating resolution proofs
- MoUsSaka  deletion-based
  http://www-pr.informatik.uni-tuebingen.de/?site=forschung/sat/algo_engineering#Software
- picomus  deletion-based
  http://fmv.jku.at/picosat/
- DMUSer  formula-trimming based on clausal proofs
  https://bitbucket.org/anton_belov/dmuser
- SAT4J  Java-based
  http://www.sat4j.org
  ...

SAT (Lecture 7)  Spring 2016  Feb 9  27 / 32
Group-MUS Extraction
Group MUSes: Recap

Example

\( G_1 = \begin{cases} 
C_1 : (x) \\
C_2 : (y) 
\end{cases} \)

\( G_2 = \begin{cases} 
C_3 : (\neg x \lor \neg y) \\
C_4 : (x \lor y) 
\end{cases} \)

\( G_3 = \begin{cases} 
C_5 : (\neg x \lor z) \\
C_6 : (\neg y \lor \neg z) 
\end{cases} \)

\( F = G_1 \cup G_2 \cup G_3, \)

where \( G_i \)s are clause-groups:

\( G_1 = \{ C_1, C_2 \}, \)
\( G_2 = \{ C_3, C_4 \}, \)
\( G_3 = \{ C_5, C_6 \}, \)

\( \{ G_1, G_2 \} \) is a group-MUS of \( F: \)

\( G_1 \cup G_2 \in \text{UNSAT} \)
\( G_1, G_2 \in \text{SAT}. \)

\( \{ G_1, G_3 \} \) is a group-MUS, too.
Group MUSes: Recap

Example

\[ G_1 = \begin{cases} C_1 : (x) \\ C_2 : (y) \end{cases} \]

\[ G_2 = \begin{cases} C_3 : (\neg x \lor \neg y) \\ C_4 : (x \lor y) \end{cases} \]

\[ G_3 = \begin{cases} C_5 : (\neg x \lor z) \\ C_6 : (\neg y \lor \neg z) \end{cases} \]

- \( F = G_1 \cup G_2 \cup G_3 \),
  where \( G_i \)'s are clause-groups:
  \( G_1 = \{ C_1, C_2 \} \), \( G_2 = \{ C_3, C_4 \} \),
  \( G_3 = \{ C_5, C_6 \} \).

- \( \{ G_1, G_2 \} \) is a group-MUS of \( F \):
  \( G_1 \cup G_2 \in \text{UNSAT} \)
  \( G_1, G_2 \in \text{SAT} \).

- \( \{ G_1, G_3 \} \) is a group-MUS, too.
Group MUSes: Recap

Example

\[ G_1 = \begin{cases} 
C_1 : (x) \\
C_2 : (y) 
\end{cases} \]

\[ G_2 = \begin{cases} 
C_3 : (\neg x \vee \neg y) \\
C_4 : (x \vee y) 
\end{cases} \]

\[ G_3 = \begin{cases} 
C_5 : (\neg x \vee z) \\
C_6 : (\neg y \vee \neg z) 
\end{cases} \]

- \( F = G_1 \cup G_2 \cup G_3 \), where \( G_i \)'s are clause-groups:
  - \( G_1 = \{ C_1, C_2 \} \)
  - \( G_2 = \{ C_3, C_4 \} \)
  - \( G_3 = \{ C_5, C_6 \} \)

- \( \{ G_1, G_2 \} \) is a group-MUS of \( F \):
  - \( G_1 \cup G_2 \in \text{UNSAT} \)
  - \( G_1, G_2 \in \text{SAT} \)

- \( \{ G_1, G_3 \} \) is a group-MUS, too.
Definition
Given a group-partitioned CNF formula $\mathcal{F} = G_0 \cup G_1 \cup \cdots \cup G_m$, a group-MUS of $\mathcal{F}$ is a subset $\{G_{i_1}, \ldots, G_{i_k}\} \subseteq \{G_1 \cup \cdots \cup G_m\}$ such that

- $\mathcal{F}' = G_0 \cup \bigcup_{j=1}^k G_{i_j} \in \text{UNSAT}$, and
- $\mathcal{F}' \setminus G_{i_j} \in \text{SAT}$ for each $j = 1..k$.

Example: a group = CNF representation of a gate in a circuit.
Note the special role of $G_0$ : the “background” clauses.
Example: output constraint in circuit.
$G_0 \in \text{UNSAT} \iff$ unique GMUS $\emptyset$.
MUS extraction: a special case of group-MUS extraction
Group-MUS extraction can be reduced to MUS extraction
Group-MUS Extraction via MUS Extraction

Given a group-partitioned CNF formula \( F = G_0 \cup G_1 \cup \cdots \cup G_m \):

- Consider the CNF formula
  \[
  F = G_0 \cup \bigcup_{i=1}^{m} \{(C \lor a_i) \mid C \in G_i\},
  \]

where each \( a_i \) is a new (assumption) variable.
  - One-to-one mapping between \( a_i \) and \( G_i \) (a group of clauses)!

Use the assumption interface of a SAT solver to find UNSAT cores of \( F \) over the assumption variables \( a_i \):
  - Can essentially implement any MUS extraction algorithm using assumptions
Summary

Take-home message
- SAT solvers central in MUS extraction
- CDCL SAT solvers used incrementally via the assumption interface
- Also applicable to extracting “MUSes” of high-level

Study goals
- Concepts: necessary clauses
- The central MUS extraction algorithms
- Extracting group-MUSes with “plain” MUS extractors

Next time:
Boolean optimization: maximum satisfiability (part I)