Minimal Unsatisfiability: Theory, Algorithms and Applications

Anton Belov

Complex and Adaptive Systems Laboratory
School of Computer Science and Informatics
University College Dublin, Ireland

EPCL Basic Training Camp
December 10-21, 2012
Dresden, Germany
Abstract

Minimal unsatisfiability and minimally unsatisfiable subformulas (MUSes) find a wide range of practical applications, including hardware and software design and verification, product configuration and knowledge-based validation. This course provides an introduction to the topic of minimal unsatisfiability in classical propositional logic and some of its extensions. The course covers the basic theory of minimal unsatisfiability and related concepts, algorithms and optimization techniques for computing and approximating MUSes, and some of the concrete applications of minimal unsatisfiability. The course assumes familiarity with classical propositional logic, propositional satisfiability (SAT) and the basics of SAT solving.
\( \mathcal{F} = \{ C_1, \ldots, C_6 \} \)

\[
\begin{align*}
C_1 &= (p) & C_3 &= (\neg p \lor \neg q) & C_5 &= (p \lor q) \\
C_2 &= (q) & C_4 &= (\neg p \lor r) & C_6 &= (\neg q \lor \neg r)
\end{align*}
\]
\[ \mathcal{F} = \{ C_1, \ldots, C_6 \} \]

\[
\begin{align*}
C_1 &= (p) & C_3 &= (\neg p \lor \neg q) & C_5 &= (p \lor q) \\
C_2 &= (q) & C_4 &= (\neg p \lor r) & C_6 &= (\neg q \lor \neg r)
\end{align*}
\]

UNSAT

Can we narrow down the source(s) of \( \mathcal{F} \)'s inconsistency?
\( \mathcal{F} = \{ C_1, \ldots, C_6 \} \)

Can we narrow down the source(s) of \( \mathcal{F} \)'s inconsistency?

The set \( \{ C_1, C_2, C_3 \} \in \text{UNSAT} \), and is \textit{minimal} w.r.t. to UNSAT.
Can we narrow down the source(s) of \( \mathcal{F} \)'s inconsistency?

The set \( \{ C_1, C_2, C_3 \} \in \text{UNSAT} \), and is minimal w.r.t. to UNSAT.

\( \{ C_1, C_2, C_3 \} \) is \textit{minimally unsatisfiable subformula (MUS)} of \( \mathcal{F} \).
Can we narrow down the source(s) of $\mathcal{F}$’s inconsistency?

The set $\{C_1, C_2, C_3\} \in \text{UNSAT}$, and is \textit{minimal} w.r.t. to UNSAT.

$\{C_1, C_2, C_3\}$ is \textit{minimally unsatisfiable subformula (MUS)} of $\mathcal{F}$.

$\{C_1, C_2, C_4, C_6\}$ is another MUS of $\mathcal{F}$.
Introduction

Why do we care about minimal unsatisfiability (MU) and MUSes?
Introduction

Why do we care about minimal unsatisfiability (MU) and MUSes?

Theory

▶ Proof complexity (e.g. known hard examples for resolution are MU).
▶ MU-decision problem is \( D^P \)-complete (related to various “criticality” problems).

Applications

▶ Identification and repair of sources of inconsistency:
   ▶ circuit error diagnosis;
   ▶ type debugging in programming languages;
   ▶ error localization in (automotive) product configuration.

▶ Identification of relevant/important features of systems:
   ▶ abstraction in model checking;
   ▶ environmental assumptions in formal equivalence checking;
   ▶ interpolation-based model checking;
   ▶ logic synthesis (Boolean function decomposition).
Introduction

Why do we care about minimal unsatisfiability (MU) and MUSes?

Theory

▶ Proof complexity (e.g. known hard examples for resolution are MU).
▶ MU-decision problem is $D^P$-complete (related to various “criticality” problems).

Applications

▶ Identification and repair of sources of inconsistency:
  ▶ circuit error diagnosis;
  ▶ type debugging in programming languages;
  ▶ error localization in (automotive) product configuration.
Introduction

Why do we care about minimal unsatisfiability (MU) and MUSes?

Theory
- Proof complexity (e.g. known hard examples for resolution are MU).
- MU-decision problem is $D^P$-complete (related to various “criticality” problems).

Applications
- Identification and repair of sources of inconsistency:
  - circuit error diagnosis;
  - type debugging in programming languages;
  - error localization in (automotive) product configuration.
- Identification of relevant/important features of systems:
  - abstraction in model checking;
  - environmental assumptions in formal equivalence checking;
  - interpolation-based model checking;
  - logic synthesis (Boolean function decomposition).
Introduction

Why do we care about minimal unsatisfiability (MU) and MUSes?

Theory
- Proof complexity (e.g. known hard examples for resolution are MU).
- MU-decision problem is \( \mathsf{D}^\mathsf{P} \)-complete (related to various “criticality” problems).

Applications
- Identification and repair of sources of inconsistency:
  - circuit error diagnosis;
  - type debugging in programming languages;
  - error localization in (automotive) product configuration.
- Identification of relevant/important features of systems:
  - abstraction in model checking;
  - environmental assumptions in formal equivalence checking;
  - interpolation-based model checking;
  - logic synthesis (Boolean function decomposition).

Interesting and difficult problem
Course outline

1. Preliminaries: CNF formulas, notation, SAT solvers.
2. Minimal Unsatisfiability: example, definition.
3. Structure of UNSAT formulas: MUSes, MCSes, hitting sets duality, MaxSAT.
5. Towards the algorithms: categorization of clauses.
7. Applications.
Preliminaries

Propositional variables: $\text{Vars} = \{p, q, r, \ldots\}$. 
Preliminaries

Propositional variables: \( \text{Vars} = \{p, q, r, \ldots \} \).

Literal: \( l \) — a variable (e.g. \( p \)) or its negation (e.g. \( \neg p \)).
Preliminaries

Propositional variables: \( \text{Vars} = \{p, q, r, \ldots \} \).

- **Literal**: \( l \) — a variable (e.g. \( p \)) or its negation (e.g. \( \neg p \)).

- **Clause**: \( C \) — a disjunction of literals (e.g. \( C = (p \lor \neg q) \)); empty clause \( \bot \); tautological clause — contains a literal and its negation.

Assume no tautologies.

- \( \left| F \right| \) — number of clauses.
- \( \text{Var}(F) \) — the set of variables that occur in \( F \).

Assignments: \( \tau: \text{Vars} \mapsto \{0, 1\} \); assume complete assignments, i.e. for a formula \( F \), \( \tau: \text{Var}(F) \mapsto \{0, 1\} \).

Assignments are extended to literals, clauses and formulas according to the semantics of classical propositional logic.

Note: \( \tau(\bot) \) is always 0, \( \tau(\emptyset) \) is always 1.
Preliminaries

Propositional variables: \( \text{Vars} = \{p, q, r, \ldots \} \).

**Literal**: \( l \) — a variable (e.g. \( p \)) or its negation (e.g. \( \neg p \)).

**Clause**: \( C \) — a disjunction of literals (e.g. \( C = (p \lor \neg q) \)); empty clause \( \perp \); tautological clause — contains a literal and its negation.

**CNF formula**: \( \mathcal{F} \) — a conjunction of clauses; written/treated as a set of clauses (e.g. \( \mathcal{F} = \{C_1, C_2, C_3\} \)).

- Assume no tautologies.
- \( |\mathcal{F}| \) — number of clauses.
- \( \text{Var}(\mathcal{F}) \) — the set of variables that occur in \( \mathcal{F} \).
Preliminaries

Propositional variables: \( Vars = \{p, q, r, \ldots \} \).

**Literal**: \( l \) — a variable (e.g. \( p \)) or its negation (e.g. \( \neg p \)).

**Clause**: \( C \) — a disjunction of literals (e.g. \( C = (p \lor \neg q) \)); empty clause \( \bot \); tautological clause — contains a literal and its negation.

**CNF formula**: \( F \) — a conjunction of clauses; written/treated as a set of clauses (e.g. \( F = \{C_1, C_2, C_3\} \)).

- Assume no tautologies.
- \( |F| \) — number of clauses.
- \( \text{Var}(F) \) — the set of variables that occur in \( F \).

**Assignment**: \( \tau : Vars \to \{0, 1\} \); assume complete assignments, i.e. for a formula \( F \), \( \tau : \text{Var}(F) \to \{0, 1\} \).
Preliminaries

Propositional variables: \( Vars = \{p, q, r, \ldots\} \).

**Literal**: \( l \) — a variable (e.g. \( p \)) or its negation (e.g. \( \neg p \)).

**Clause**: \( C \) — a disjunction of literals (e.g. \( C = (p \lor \neg q) \)); empty clause \( \bot \); tautological clause — contains a literal and its negation.

**CNF formula**: \( F \) — a conjunction of clauses; written/treated as a set of clauses (e.g. \( F = \{C_1, C_2, C_3\} \)).

- Assume no tautologies.
- \( |F| \) — number of clauses.
- \( \text{Var}(F) \) — the set of variables that occur in \( F \).

**Assignment**: \( \tau : Vars \rightarrow \{0, 1\} \); assume complete assignments, i.e. for a formula \( F \), \( \tau : \text{Var}(F) \rightarrow \{0, 1\} \).

Assignments are extended to literals, clauses and formulas according to the semantics of classical propositional logic.

**Note**: \( \tau(\bot) \) is always 0, \( \tau(\emptyset) \) is always 1.
Def: \( \tau \) is a model of CNF formula \( \mathcal{F} \) if \( \tau(\mathcal{F}) = 1 \).
**Preliminaries**

**Def:** \( \tau \) is a *model* of CNF formula \( F \) if \( \tau(F) = 1 \).

**Def:** \( F \) is *satisfiable* if it has a model, *unsatisfiable* otherwise.
**Preliminaries**

**Def:** \( \tau \) is a *model* of CNF formula \( \mathcal{F} \) if \( \tau(\mathcal{F}) = 1 \).

**Def:** \( \mathcal{F} \) is *satisfiable* if it has a model, *unsatisfiable* otherwise.

**SAT** (resp. **UNSAT**) — the set of all satisfiable (resp. unsatisfiable) CNF formulas.

**Theorem** [cf Cook’71]: SAT is NP-complete.
**Preliminaries**

**Def:** \( \tau \) is a *model* of CNF formula \( \mathcal{F} \) if \( \tau(\mathcal{F}) = 1 \).

**Def:** \( \mathcal{F} \) is *satisfiable* if it has a model, *unsatisfiable* otherwise.

**SAT** (resp. **UNSAT**) — the set of all satisfiable (resp. unsatisfiable) CNF formulas.

**Theorem** [cf Cook'71]: SAT is NP-complete.

**Notation:** \( \text{Unsat}(\mathcal{F}, \tau) = \{ C \in \mathcal{F} \mid \tau(C) = 0 \} \) — clauses of \( \mathcal{F} \) falsified by the assignment \( \tau \).
**Def:** \( \tau \) is a **model** of CNF formula \( \mathcal{F} \) if \( \tau(\mathcal{F}) = 1 \).

**Def:** \( \mathcal{F} \) is **satisfiable** if it has a model, **unsatisfiable** otherwise.

**SAT** (resp. **UNSAT**) — the set of all satisfiable (resp. unsatisfiable) CNF formulas.

**Theorem** [cf Cook’71]: SAT is NP-complete.

**Notation:** \( \text{Unsat}(\mathcal{F}, \tau) = \{ C \in \mathcal{F} \mid \tau(C) = 0 \} \) — clauses of \( \mathcal{F} \) falsified by the assignment \( \tau \).

Despite NP-completeness of SAT, SAT solvers are capable of handling very large practical instances.
Preliminaries: SAT solvers

Best-performing SAT solvers for practical applications are CDCL-based.


  1. When $F \in \text{UNSAT}$, solvers can produce an unsatisfiable core $U \subseteq F$, s.t. $U \in \text{UNSAT}$.
  
    *Note:* $U$ is the support of a resolution refutation of $F$ implicitly built by a CDCL-based SAT solver.

  2. SAT solvers are **incremental** — solvers allow for multiple invocations, on the same, or larger, formula, and re-use previously learned information.

  3. SAT solving with assumptions [Eén and Sörensson, '03]: let $A \subseteq \text{Vars}(F)$, and let $\tau: A \mapsto \{0, 1\}$. SAT solvers determine satisfiability of $F|_A$ directly, i.e. without modifying the input formula.

  4. Incrementality and assumptions allow to **program** SAT solvers: adding/removing and enabling/disabling clauses.
Best-performing SAT solvers for practical applications are CDCL-based.


Important features of modern CDCL-based SAT solvers:

1. When $\mathcal{F} \in \text{UNSAT}$, solvers can produce an *unsatisfiable core* — a sub-formula $\mathcal{U} \subseteq \mathcal{F}$, s.t. $\mathcal{U} \in \text{UNSAT}$. **Note**: $\mathcal{U}$ is the support of a resolution refutation of $\mathcal{F}$ implicitly built by a CDCL-based SAT solver.
Preliminaries: SAT solvers

Best-performing SAT solvers for practical applications are CDCL-based.


Important features of modern CDCL-based SAT solvers:

1. When $\mathcal{F} \in \text{UNSAT}$, solvers can produce an unsatisfiable core — a sub-formula $\mathcal{U} \subseteq \mathcal{F}$, s.t. $\mathcal{U} \in \text{UNSAT}$. Note: $\mathcal{U}$ is the support of a resolution refutation of $\mathcal{F}$ implicitly built by a CDCL-based SAT solver.

2. SAT solvers are incremental — solvers allow for multiple invocations, on the same, or larger, formula, and re-use previously learned information.
Preliminaries: SAT solvers

Best-performing SAT solvers for practical applications are CDCL-based.


Important features of modern CDCL-based SAT solvers:

1. When $\mathcal{F} \in \text{UNSAT}$, solvers can produce an *unsatisfiable core* — a sub-formula $\mathcal{U} \subseteq \mathcal{F}$, s.t. $\mathcal{U} \in \text{UNSAT}$. **Note:** $\mathcal{U}$ is the support of a resolution refutation of $\mathcal{F}$ implicitly built by a CDCL-based SAT solver.

2. SAT solvers are *incremental* — solvers allow for multiple invocations, on the same, or larger, formula, and re-use previously learned information.

3. SAT solving with assumptions [Eén and Sörensson, '03]: let $A \subseteq \text{Vars}(\mathcal{F})$, and let $\tau : A \mapsto \{0, 1\}$. SAT solvers determine satisfiability of $\mathcal{F}|_A$ directly, i.e. without modifying the input formula.
Preliminaries: SAT solvers

Best-performing SAT solvers for practical applications are CDCL-based.


Important features of modern CDCL-based SAT solvers:

1. When $\mathcal{F} \in \text{UNSAT}$, solvers can produce an \textit{unsatisfiable core} — a sub-formula $\mathcal{U} \subseteq \mathcal{F}$, s.t. $\mathcal{U} \in \text{UNSAT}$. \textbf{Note:} $\mathcal{U}$ is the support of a resolution refutation of $\mathcal{F}$ implicitly built by a CDCL-based SAT solver.

2. SAT solvers are \textit{incremental} — solvers allow for multiple invocations, on the same, or larger, formula, and re-use previously learned information.

3. SAT solving with assumptions [Eén and Sörensson, '03]: let $A \subseteq \text{Vars}(\mathcal{F})$, and let $\tau : A \mapsto \{0, 1\}$. SAT solvers determine satisfiability of $\mathcal{F}|_A$ directly, i.e. without modifying the input formula.

4. Incrementality and assumptions allow to \textit{program} SAT solvers: adding/removing and enabling/disabling clauses.
Course outline

1. Preliminaries: CNF formulas, notation, SAT solvers.
2. Minimal Unsatisfiability: example, definition.
3. Structure of UNSAT formulas: MUSes, MCSes, hitting sets duality, MaxSAT.
5. Towards the algorithms: categorization of clauses.
7. Applications.
Minimal Unsatisfiability

**Def:** A CNF formula \( \mathcal{F} \) is *minimally unsatisfiable (MU)* if \( \mathcal{F} \in UNSAT \), and \( \forall C \in \mathcal{F}, \mathcal{F} \setminus \{C\} \in SAT \).
Def: A CNF formula $\mathcal{F}$ is \textit{minimally unsatisfiable (MU)} if $\mathcal{F} \in \text{UNSAT}$, and $\forall C \in \mathcal{F}, \mathcal{F} \setminus \{C\} \in \text{SAT}$.

\begin{align*}
C_1 &= (p) \quad C_2 = (\neg p \lor q) \quad C_3 = (\neg p \lor \neg q) \\
\text{UNSAT}
\end{align*}
Def: A CNF formula $\mathcal{F}$ is \textit{minimally unsatisfiable (MU)} if $\mathcal{F} \in UNSAT$, and $\forall C \in \mathcal{F}, \mathcal{F} \setminus \{C\} \in SAT$.

$C_1 = (p)$ \quad $C_2 = (\neg p \lor q)$ \quad $C_3 = (\neg p \lor \neg q)$

UNSAT
Def: A CNF formula $\mathcal{F}$ is *minimally unsatisfiable (MU)* if $\mathcal{F} \in \text{UNSAT}$, and $\forall C \in \mathcal{F}, \mathcal{F} \setminus \{C\} \in \text{SAT}$.

\[ C_1 = (p) \quad C_2 = (\neg p \lor q) \quad C_3 = (\neg p \lor \neg q) \]

SAT
**Def:** A CNF formula $\mathcal{F}$ is *minimally unsatisfiable (MU)* if $\mathcal{F} \in \text{UNSAT}$, and $\forall C \in \mathcal{F}, \mathcal{F} \setminus \{C\} \in \text{SAT}$.

\[
C_1 = (p) \quad C_2 = (\neg p \lor q) \quad C_3 = (\neg p \lor \neg q)
\]

SAT
Def: A CNF formula $\mathcal{F}$ is \textit{minimally unsatisfiable (MU)} if $\mathcal{F} \in \text{UNSAT}$, and $\forall C \in \mathcal{F}, \mathcal{F} \setminus \{C\} \in \text{SAT}$.

\[
\begin{align*}
C_1 &= (p) \quad C_2 = (\neg p \lor q) \quad C_3 = (\neg p \lor \neg q) \\
\text{SAT}&
\end{align*}
\]
**Def:** A CNF formula $\mathcal{F}$ is *minimally unsatisfiable (MU)* if $\mathcal{F} \in \text{UNSAT}$, and $\forall C \in \mathcal{F}, \mathcal{F} \setminus \{C\} \in \text{SAT}$.

\[\begin{align*}
C_1 &= (p) \\
C_2 &= (\neg p \lor q) \\
C_3 &= (\neg p \lor \neg q)
\end{align*}\]

**UNSAT**
Minimal Unsatisfiability

**Def:** A CNF formula $\mathcal{F}$ is *minimally unsatisfiable (MU)* if $\mathcal{F} \in \text{UNSAT}$, and $\forall C \in \mathcal{F}, \mathcal{F} \setminus \{C\} \in \text{SAT}$.

$C_1 = (p)$  $\quad C_2 = (\neg p \lor q)$  $\quad C_3 = (\neg p \lor \neg q)$

**Notation:** MU — class of all minimally unsatisfiable CNF formulas.
Minimal Unsatisfiability

**Def:** A CNF formula $\mathcal{F}$ is *minimally unsatisfiable (MU)* if $\mathcal{F} \in \text{UNSAT}$, and $\forall C \in \mathcal{F}, \mathcal{F} \setminus \{C\} \in \text{SAT}$.

\[
C_1 = (p) \quad C_2 = (\neg p \lor q) \quad C_3 = (\neg p \lor \neg q)
\]

UNSAT

**Notation:** MU — class of all minimally unsatisfiable CNF formulas.

$\mathcal{F}_1 = \{(p), (\neg p \lor q), (\neg p \lor \neg q)\} \in \text{MU}$
### Minimal Unsatisfiability

**Def:** A CNF formula $\mathcal{F}$ is **minimally unsatisfiable (MU)** if $\mathcal{F} \in \text{UNSAT}$, and $\forall C \in \mathcal{F}, \mathcal{F} \setminus \{C\} \in \text{SAT}$.

<table>
<thead>
<tr>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(p)$</td>
<td>$(-p \lor q)$</td>
<td>$(-p \lor \neg q)$</td>
</tr>
</tbody>
</table>

**UNSAT**

**Notation:** MU — class of all minimally unsatisfiable CNF formulas.

\[ \mathcal{F}_1 = \{(p), (\neg p \lor q), (\neg p \lor \neg q)\} \in \text{MU} \]

\[ \mathcal{F}_2 = \{(p), (\neg p \lor q)\} \notin \text{MU} \text{ (because } \notin \text{ UNSAT)} \]
**Def:** A CNF formula $\mathcal{F}$ is *minimally unsatisfiable (MU)* if $\mathcal{F} \in \text{UNSAT}$, and $\forall C \in \mathcal{F}, \mathcal{F} \setminus \{C\} \in \text{SAT}$.

\[
C_1 = (p) \quad C_2 = (\neg p \lor q) \quad C_3 = (\neg p \lor \neg q)
\]

\[\text{UNSAT}\]

**Notation:** MU — class of all minimally unsatisfiable CNF formulas.

$\mathcal{F}_1 = \{(p), (\neg p \lor q), (\neg p \lor \neg q)\} \in \text{MU}$

$\mathcal{F}_2 = \{(p), (\neg p \lor q)\} \notin \text{MU}$ (because $\notin \text{UNSAT}$)

$\mathcal{F}_3 = \{(p), (\neg p \lor q), (\neg p \lor \neg q), (p \lor q)\} \notin \text{MU}$ (because $\mathcal{F}_3 \setminus \{(p \lor q)\} \in \text{UNSAT}$)
Claim: For every $\mathcal{F} \in \text{UNSAT}$, $\exists \mathcal{M} \subseteq \mathcal{F}$, such that $\mathcal{M} \in \text{MU}$. 
Structure of UNSAT Formulas: MUSes

**Claim:** For every $F \in \text{UNSAT}$, $\exists M \subseteq F$, such that $M \in \text{MU}$.

**Pf:**

1. Let $M = F$.
2. If $M \notin \text{MU}$, $\exists C \in M$ such that $M \setminus \{C\} \in \text{UNSAT}$.
3. Let $M = M \setminus \{C\}$, goto 2.
Structure of UNSAT Formulas: MUSes

**Claim:** For every $\mathcal{F} \in \text{UNSAT}$, $\exists \mathcal{M} \subseteq \mathcal{F}$, such that $\mathcal{M} \in \text{MU}$.

**Pf:**
1. Let $\mathcal{M} = \mathcal{F}$.
2. If $\mathcal{M} \notin \text{MU}$, $\exists \mathcal{C} \in \mathcal{M}$ such that $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$.
3. Let $\mathcal{M} = \mathcal{M} \setminus \{C\}$, goto 2.

**Def:** $\mathcal{M}$ is **minimally unsatisfiable subformula (MUS)** of $\mathcal{F}$ if $\mathcal{M} \subseteq \mathcal{F}$ and $\mathcal{M} \in \text{MU}$.

**Notation:** $\text{MUS}(\mathcal{F})$ — the set of all MUSes of $\mathcal{F}$.
**Claim:** For every \( F \in \text{UNSAT} \), \( \exists M \subseteq F \), such that \( M \in \text{MU} \).

**Pf:**

1. Let \( M = F \).
2. If \( M \notin \text{MU} \), \( \exists C \in M \) such that \( M \setminus \{C\} \in \text{UNSAT} \).
3. Let \( M = M \setminus \{C\} \), goto 2.

**Def:** \( M \) is **minimally unsatisfiable subformula (MUS)** of \( F \) if \( M \subseteq F \) and \( M \in \text{MU} \).

**Notation:** \( \text{MUS}(F) \) — the set of all MUSes of \( F \).

\[
\begin{align*}
C_1 &= (p) \\
C_2 &= (q) \\
C_3 &= (\neg p \lor \neg q) \\
C_4 &= (\neg p \lor r) \\
C_5 &= (p \lor q) \\
C_6 &= (\neg q \lor \neg r)
\end{align*}
\]
Claim: For every $F \in \text{UNSAT}$, $\exists M \subseteq F$, such that $M \in \text{MU}$.

Pf:
1. Let $M = F$.
2. If $M \not\in \text{MU}$, $\exists C \in M$ such that $M \setminus \{C\} \in \text{UNSAT}$.
3. Let $M = M \setminus \{C\}$, goto 2.

Def: $M$ is a minimally unsatisfiable subformula (MUS) of $F$ if $M \subseteq F$ and $M \in \text{MU}$.

Notation: $\text{MUS}(F)$ — the set of all MUSes of $F$.

\{ $C_1, C_2, C_3$ \} and \{ $C_1, C_2, C_4, C_6$ \} are the (only) MUSes.
Point 1: MUSes of $\mathcal{F}$ characterize the sources of inconsistency of $\mathcal{F}$.
**Point 1:** MUSes of \( \mathcal{F} \) characterize the sources of inconsistency of \( \mathcal{F} \).
**Structure of UNSAT Formulas: MUSes**

**Point 1:** MUSes of $\mathcal{F}$ characterize the sources of inconsistency of $\mathcal{F}$.

Which clauses should be removed from $\mathcal{F}$ to restore its consistency?
**Point 1:** MUSes of $\mathcal{F}$ characterize the sources of inconsistency of $\mathcal{F}$.

Which clauses should be removed from $\mathcal{F}$ to restore its consistency?
**Point 1:** MUSes of $\mathcal{F}$ characterize the sources of inconsistency of $\mathcal{F}$.

Which clauses should be removed from $\mathcal{F}$ to restore its consistency?
**Point 1:** MUSes of $F$ characterize the sources of inconsistency of $F$.

Which clauses should be removed from $F$ to restore its consistency?
Point 1: MUSes of $\mathcal{F}$ characterize the sources of inconsistency of $\mathcal{F}$.

Which clauses should be removed from $\mathcal{F}$ to restore its consistency?

Point 2: To restore consistency of $\mathcal{F}$, one must “break” all of its MUSes.
Def: $\mathcal{M}$ is a minimal correction subset (MCS) of $\mathcal{F}$ if $\mathcal{F} \setminus \mathcal{M} \in \text{SAT}$ and $\forall C \in \mathcal{F}, \mathcal{F} \setminus (\mathcal{M} \setminus \{C\}) \in \text{UNSAT}$.

Notation: $\text{MCS}(\mathcal{F})$ — the set of all MCSes of $\mathcal{F}$. 
Structure of UNSAT Formulas: MCSes

**Def:** $\mathcal{M}$ is *minimal correction subset (MCS)* of $\mathcal{F}$ if $\mathcal{F} \setminus \mathcal{M} \in \text{SAT}$ and $\forall C \in \mathcal{F}, \mathcal{F} \setminus (\mathcal{M} \setminus \{C\}) \in \text{UNSAT}$. 

**Notation:** $\text{MCS}(\mathcal{F})$ — the set of all MCSes of $\mathcal{F}$.

\[
\begin{align*}
\mathcal{C}_1 &= (p) \\
\mathcal{C}_2 &= (q) \\
\mathcal{C}_3 &= (\neg p \lor \neg q) \\
\mathcal{C}_4 &= (\neg p \lor r) \\
\mathcal{C}_5 &= (p \lor q) \\
\mathcal{C}_6 &= (\neg q \lor \neg r)
\end{align*}
\]

\{$\mathcal{C}_1\}$ $\in$ $\text{MCS}(\mathcal{F})$ 

\{$\mathcal{C}_3, \mathcal{C}_4\}$ $\in$ $\text{MCS}(\mathcal{F})$ — note: minimal $\neq$ smallest (minimum). 

\{$\mathcal{C}_2, \mathcal{C}_4\}$ $\not\in$ $\text{MCS}(\mathcal{F})$, because it is not minimal.
Def: $\mathcal{M}$ is *minimal correction subset (MCS)* of $\mathcal{F}$ if $\mathcal{F} \setminus \mathcal{M} \in \text{SAT}$ and $\forall C \in \mathcal{F}, \mathcal{F} \setminus (\mathcal{M} \setminus \{C\}) \in \text{UNSAT}$.

Notation: $\text{MCS}(\mathcal{F})$ — the set of all MCSes of $\mathcal{F}$.

$\{C_1\} \in \text{MCS}(\mathcal{F})$
**Structure of UNSAT Formulas: MCSes**

**Def:** $M$ is *minimal correction subset (MCS)* of $F$ if $F \setminus M \in SAT$ and $\forall C \in F, F \setminus (M \setminus \{C\}) \in UNSAT$.

**Notation:** $\text{MCS}(F)$ — the set of all MCSes of $F$.

\[
\{C_1\} \in \text{MCS}(F)
\]
Structure of UNSAT Formulas: MCSes

**Def:** $\mathcal{M}$ is *minimal correction subset (MCS)* of $\mathcal{F}$ if $\mathcal{F} \setminus \mathcal{M} \in \text{SAT}$ and $\forall C \in \mathcal{F}, \mathcal{F} \setminus (\mathcal{M} \setminus \{C\}) \in \text{UNSAT}.$

**Notation:** $\text{MCS}(\mathcal{F})$ — the set of all MCSes of $\mathcal{F}.$

| $C_1 = (p)$ | $C_3 = (\neg p \lor \neg q)$ | $C_5 = (p \lor q)$ |
| $C_2 = (q)$ | $C_4 = (\neg p \lor r)$ | $C_6 = (\neg q \lor \neg r)$ |

\{C_1\} ∈ MCS(\mathcal{F})
\{C_3, C_4\} ∈ MCS(\mathcal{F}) — note: minimal ≠ smallest (minimum).
Structure of UNSAT Formulas: MCSes

**Def:** \( \mathcal{M} \) is *minimal correction subset (MCS)* of \( \mathcal{F} \) if \( \mathcal{F} \setminus \mathcal{M} \in \text{SAT} \) and \( \forall C \in \mathcal{F}, \mathcal{F} \setminus (\mathcal{M} \setminus \{C\}) \in \text{UNSAT} \).

**Notation:** \( \text{MCS}(\mathcal{F}) \) — the set of all MCSes of \( \mathcal{F} \).

\[
\begin{align*}
C_1 &= (p) \\
C_2 &= (q) \\
C_3 &= (\neg p \lor \neg q) \\
C_4 &= (\neg p \lor r) \\
C_5 &= (p \lor q) \\
C_6 &= (\neg q \lor \neg r)
\end{align*}
\]

\( \{C_1\} \in \text{MCS}(\mathcal{F}) \)

\( \{C_3, C_4\} \in \text{MCS}(\mathcal{F}) — \text{note: minimal} \neq \text{smallest (minimum)}. \)
Structure of UNSAT Formulas: MCSes

**Def:** \( M \) is *minimal correction subset (MCS)* of \( F \) if \( F \setminus M \in \text{SAT} \) and \( \forall C \in F, F \setminus (M \setminus \{C\}) \in \text{UNSAT} \).

**Notation:** \( \text{MCS}(F) \) — the set of all MCSes of \( F \).

<table>
<thead>
<tr>
<th>( C_1 )</th>
<th>( C_3 )</th>
<th>( C_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>((p))</td>
<td>((\neg p \lor \neg q))</td>
<td>((p \lor q))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( C_2 )</th>
<th>( C_4 )</th>
<th>( C_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>((q))</td>
<td>((\neg p \lor r))</td>
<td>((\neg q \lor \neg r))</td>
</tr>
</tbody>
</table>

\( \{C_1\} \in \text{MCS}(F) \)
\( \{C_3, C_4\} \in \text{MCS}(F) \) — note: minimal \( \neq \) smallest (minimum).
\( \{C_2, C_4\} \notin \text{MCS}(F) \), because it is not minimal.
Structure of UNSAT Formulas: Hitting Sets Duality

Let $\mathcal{S}$ be a collection of arbitrary sets.

**Def:** A set $H$ is a hitting set of $\mathcal{S}$ if for all $S \in \mathcal{S}$, $H \cap S \neq \emptyset$.
Let $\mathcal{I}$ be a collection of arbitrary sets.

**Def:** A set $H$ is a **hitting set** of $\mathcal{I}$ if for all $S \in \mathcal{I}$, $H \cap S \neq \emptyset$.

**Def:** A hitting set $H$ is **irreducible** if no $H' \subset H$ is a hitting set of $\mathcal{I}$.

Another often-used term: hyper-graph transversal.
Let $\mathcal{S}$ be a collection of arbitrary sets.

**Def:** A set $H$ is a **hitting set** of $\mathcal{S}$ if for all $S \in \mathcal{S}$, $H \cap S \neq \emptyset$.

**Def:** A hitting set $H$ is **irreducible** if no $H' \subset H$ is a hitting set of $\mathcal{S}$.

Another often-used term: hyper-graph transversal.
Let $\mathcal{S}$ be a collection of arbitrary sets.

**Def:** A set $H$ is a **hitting set** of $\mathcal{S}$ if for all $S \in \mathcal{S}$, $H \cap S \neq \emptyset$.

**Def:** A hitting set $H$ is **irreducible** if no $H' \subset H$ is a hitting set of $\mathcal{S}$.

Another often-used term: hyper-graph transversal.

\[ \{a, c, e, f\} \] is a hitting set of \{A, B, C\}, but not irreducible.
Structure of UNSAT Formulas: Hitting Sets Duality

Let $\mathcal{S}$ be a collection of arbitrary sets.

**Def:** A set $H$ is a *hitting set* of $\mathcal{S}$ if for all $S \in \mathcal{S}$, $H \cap S \neq \emptyset$.

**Def:** A hitting set $H$ is *irreducible* if no $H' \subset H$ is a hitting set of $\mathcal{S}$.

Another often-used term: hyper-graph transversal.

\[
\begin{align*}
\{a, c, e, f\} & \text{ is a hitting set of } \{A, B, C\}, \text{ but not irreducible.} \\
\{a, d, e\} & \text{ is an irreducible hitting set, so is } \{c, e\}.
\end{align*}
\]
To restore consistency of $\mathcal{F} \in \text{UNSAT}$ one must “break” each of the MUSes of $\mathcal{F}$. 

\begin{itemize}
  \item A correction subset of $\mathcal{F}$ must be a hitting set of MUS($\mathcal{F}$).
  \item A minimal correction subset (MCS) of $\mathcal{F}$ must be an irreducible hitting set of MUS($\mathcal{F}$).
\end{itemize}

Theorem [Reiter, ’87; Birnbaum and Lozinskii, ’03]: Let $\mathcal{F}$ be in UNSAT.

\begin{itemize}
  \item $M$ is an MCS of $\mathcal{F}$ iff $M$ is an irreducible hitting set of MUS($\mathcal{F}$).
  \item $M$ is an MUS of $\mathcal{F}$ iff $M$ is an irreducible hitting set of MCS($\mathcal{F}$).
\end{itemize}

Note: In [Reiter’ 87] MCS = minimal diagnosis, MUS = minimal conflict set.
To restore consistency of $\mathcal{F} \in \text{UNSAT}$ one must “break” each of the MUSes of $\mathcal{F}$.

A correction subset of $\mathcal{F}$ must be a hitting set of $\text{MUS}(\mathcal{F})$. 

Theorem [Reiter, '87; Birnbaum and Lozinskii, '03]: Let $\mathcal{F}$ be in UNSAT.

$\triangleright$ $M$ is an MCS of $\mathcal{F}$ iff $M$ is an irreducible hitting set of $\text{MUS}(\mathcal{F})$.

$\triangleright$ $M$ is an MUS of $\mathcal{F}$ iff $M$ is an irreducible hitting set of $\text{MCS}(\mathcal{F})$.

Note: In [Reiter' 87] MCS = minimal diagnosis, MUS = minimal conflict set.
To restore consistency of $\mathcal{F} \in \text{UNSAT}$ one must “break” each of the MUSes of $\mathcal{F}$.

A correction subset of $\mathcal{F}$ must be a hitting set of $\text{MUS}(\mathcal{F})$.

A *minimal* correction subset (MCS) of $\mathcal{F}$ must be an *irreducible* hitting set of $\text{MUS}(\mathcal{F})$. 

Theorem [Reiter, '87; Birnbaum and Lozinskii, '03]: Let $\mathcal{F}$ be in UNSAT.

$
\begin{align*}
\text{M} \text{ is an MCS of } \mathcal{F} \iff & \text{M is an irreducible hitting set of MUS}(\mathcal{F}). \\
\text{M is an MUS of } \mathcal{F} \iff & \text{M is an irreducible hitting set of MCS}(\mathcal{F}).
\end{align*}
$

Note: In [Reiter' 87] MCS = minimal diagnosis, MUS = minimal conflict set.
To restore consistency of $\mathcal{F} \in \text{UNSAT}$ one must “break” each of the MUSes of $\mathcal{F}$.

A correction subset of $\mathcal{F}$ must be a hitting set of $\text{MUS}(\mathcal{F})$.

A minimal correction subset (MCS) of $\mathcal{F}$ must be an irreducible hitting set of $\text{MUS}(\mathcal{F})$.

**Theorem** [Reiter, '87; Birnbaum and Lozinskii, '03]: Let $\mathcal{F}$ be in UNSAT.

- $\mathcal{M}$ is an MCS of $\mathcal{F}$ iff $\mathcal{M}$ is an irreducible hitting set of $\text{MUS}(\mathcal{F})$.
- $\mathcal{M}$ is an MUS of $\mathcal{F}$ iff $\mathcal{M}$ is an irreducible hitting set of $\text{MCS}(\mathcal{F})$.

**Note:** In [Reiter’ 87] MCS = minimal diagnosis, MUS = minimal conflict set.
Structure of UNSAT Formulas: Hitting Sets Duality

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

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

\[ C_1 = (p) \]
\[ C_2 = (q) \]
\[ C_3 = (\neg p \lor \neg q) \]
\[ C_4 = (\neg p \lor r) \]
\[ C_5 = (p \lor q) \]
\[ C_6 = (\neg q \lor \neg r) \]
Structure of UNSAT Formulas: MaxSAT

**Def:** MaxSAT problem — given a CNF formula $\mathcal{F}$ find an assignment $\tau$ (a MaxSAT solution) that satisfies the maximum number of clauses.

Let $\tau$ be a MaxSAT solution of $\mathcal{F}$, and let $M = \text{Unsat}(\mathcal{F}, \tau)$. Note: $\mathcal{F} \setminus M \in \text{SAT}$, and $\forall C \in M$, $\mathcal{F} \setminus (M \{ C \}) \in \text{UNSAT}$. I.e. $M$ is a MCS of $\mathcal{F}$.

Point: MaxSAT is about finding the smallest MCS.

A model of $\{ C_2, \ldots, C_6 \}$ is a MaxSAT solution.

A. Belov

MU and MUSes: Theory, Algorithms and Applications

EPCL Training Camp, 2012

# 17
Structure of UNSAT Formulas: MaxSAT

**Def:** *MaxSAT problem* — given a CNF formula $\mathcal{F}$ find an assignment $\tau$ (a *MaxSAT solution*) that satisfies the *maximum* number of clauses.

Let $\tau$ be a MaxSAT solution of $\mathcal{F}$, and let $\mathcal{M} = \text{Unsat}(\mathcal{F}, \tau)$.

Point: MaxSAT is about finding the smallest MCS.
**Structure of UNSAT Formulas: MaxSAT**

**Def:** MaxSAT problem — given a CNF formula $\mathcal{F}$ find an assignment $\tau$ (a MaxSAT solution) that satisfies the *maximum* number of clauses.

Let $\tau$ be a MaxSAT solution of $\mathcal{F}$, and let $\mathcal{M} = \text{Unsat}(\mathcal{F}, \tau)$.

**Note:** $\mathcal{F} \setminus \mathcal{M} \in \text{SAT}$, and $\forall C \in \mathcal{M}$, $\mathcal{F} \setminus (\mathcal{M} \setminus \{C\}) \in \text{UNSAT}$. I.e. $\mathcal{M}$ is an MCS of $\mathcal{F}$.
**Def:** *MaxSAT problem* — given a CNF formula $\mathcal{F}$ find an assignment $\tau$ (a *MaxSAT solution*) that satisfies the *maximum* number of clauses.

Let $\tau$ be a MaxSAT solution of $\mathcal{F}$, and let $\mathcal{M} = \text{Unsat}(\mathcal{F}, \tau)$.

**Note:** $\mathcal{F} \setminus \mathcal{M} \in \text{SAT}$, and $\forall C \in \mathcal{M}, \mathcal{F} \setminus (\mathcal{M} \setminus \{C\}) \in \text{UNSAT}$. I.e. $\mathcal{M}$ is an MCS of $\mathcal{F}$.

**Point:** MaxSAT is about finding the smallest MCS.
Structure of UNSAT Formulas: MaxSAT

**Def:** MaxSAT problem — given a CNF formula $\mathcal{F}$ find an assignment $\tau$ (a MaxSAT solution) that satisfies the maximum number of clauses.

Let $\tau$ be a MaxSAT solution of $\mathcal{F}$, and let $\mathcal{M} = \text{Unsat}(\mathcal{F}, \tau)$.

**Note:** $\mathcal{F} \setminus \mathcal{M} \in \text{SAT}$, and $\forall C \in \mathcal{M}$, $\mathcal{F} \setminus (\mathcal{M} \setminus \{C\}) \in \text{UNSAT}$. I.e. $\mathcal{M}$ is an MCS of $\mathcal{F}$.

**Point:** MaxSAT is about finding the smallest MCS.

<table>
<thead>
<tr>
<th>$C_1 = (p)$</th>
<th>$C_2 = (q)$</th>
<th>$C_3 = (\neg p \lor \neg q)$</th>
<th>$C_4 = (\neg p \lor r)$</th>
<th>$C_5 = (p \lor q)$</th>
<th>$C_6 = (\neg q \lor \neg r)$</th>
</tr>
</thead>
</table>

MCSes

A model of $\{C_2, \ldots, C_6\}$ is a MaxSAT solution.
Course outline

1. Preliminaries: CNF formulas, notation, SAT solvers.
2. Minimal Unsatisfiability: example, definition.
3. Structure of UNSAT formulas: MUSes, MCSes, hitting sets duality, MaxSAT.
5. Towards the algorithms: categorization of clauses.
7. Applications.
Complexity Results

**Theorem** [Papadimitriou and Wolfe, ’88]: MU is $D^P$-complete.
Theorem [Papadimitriou and Wolfe, ‘88]: MU is $D^P$-complete.

$L \in D^P$ iff $L = L_1 \cap L_2$ with $L_1 \in \text{NP}$ and $L_2 \in \text{coNP}$.


**Theorem** [Papadimitriou and Wolfe, '88]: MU is $D^P$-complete.

$L \in D^P$ iff $L = L_1 \cap L_2$ with $L_1 \in \text{NP}$ and $L_2 \in \text{coNP}$.

- Canonical $D^P$-complete problem: SAT – UNSAT
  the set of all pairs $\langle F_1, F_2 \rangle$ with $F_1 \in \text{SAT}$ and $F_2 \in \text{UNSAT}$.

- **Note**: [Papadimitriou and Wolfe, '88] nicely reduce SAT to MU and UNSAT to MU.

- Many problems related to “critical combinatorics” are in $D^P$.
Complexity Results

**Theorem** [Papadimitriou and Wolfe, ’88]: MU is $D^P$-complete.

$L \in D^P \text{ iff } L = L_1 \cap L_2 \text{ with } L_1 \in NP \text{ and } L_2 \in coNP$.

- Canonical $D^P$-complete problem: SAT − UNSAT
  the set of all pairs $\langle F_1, F_2 \rangle$ with $F_1 \in SAT$ and $F_2 \in UNSAT$.

- **Note**: [Papadimitriou and Wolfe, ’88] nicely reduce SAT to MU and UNSAT to MU.

- Many problems related to “critical combinatorics” are in $D^P$.

For certain formulas the problem is easier.

**Def:** A **deficiency** of a CNF formula $F$, $d(F) = |F| - |Vars(F)|$.

**Theorem** [Aharoni and Linial, ’86]: $F \in MU \Rightarrow d(F) > 0$. 
Notation: MU($k$) — the set of MU formulas with deficiency $k$. 

Theorem [Fleischner, Kullmann, and Szeider, '02]: For a fixed $k$, MU($k$) $\in$ P.

▶ Specifically: $F \in$ MU($k$) can be decided in $O(L \cdot n^k + 1)$, where $n = |\text{Var}(F)|$, and $L$ is the length of $F$ (i.e. $L = O(n^2)$).

▶ $O(n^2)$ algorithm for deciding $F \in$ MU(1) in [Davidov et al, '98].

▶ $O(n^3)$ algorithm for MU(2) in [Kleine Buning, '00] (also, MU($k$) $\in$ NP).

Note: in practical applications $k$ is very large.

MUS computation: in FP $\in$ NP

Note: i.e. poly number of calls to NP oracle.

Practical algorithms for computation of MUSes are based on iterative calls to SAT solver.
Notation: $\text{MU}(k)$ — the set of MU formulas with deficiency $k$.

Theorem [Fleischner, Kullmann, and Szeider, '02]: For a fixed $k$, $\text{MU}(k) \in P$. 

Note: in practical applications $k$ is very large.

Note: i.e. poly number of calls to NP oracle.

Practical algorithms for computation of MUSes are based on iterative calls to SAT solver.
**Notation:** MU($k$) — the set of MU formulas with deficiency $k$.

**Theorem** [Fleischner, Kullmann, and Szeider, '02]: For a fixed $k$, MU($k$) $\in P$.

- Specifically: $\mathcal{F} \in$ MU($k$) can be decided in $O(L \cdot n^{k+\frac{1}{2}})$, where $n = |\text{Var}(\mathcal{F})|$, and $L$ is the length of $\mathcal{F}$ (i.e. $L = O(n^2)$).
- $O(n^2)$ algorithm for deciding $\mathcal{F} \in$ MU(1) in [Davidov et al, '98].
- $O(n^3)$ algorithm for MU(2) in [Kleine Buning, '00] (also, MU($k$) $\in$ NP).
**Complexity Results**

**Notation:** MU\((k)\) — the set of MU formulas with deficiency \(k\).

**Theorem** [Fleischner, Kullmann, and Szeider, '02]: For a fixed \(k\), MU\((k)\) \(\in\) \(P\).

- Specifically: \(\mathcal{F} \in \text{MU}(k)\) can be decided in \(O(L \cdot n^{k+\frac{1}{2}})\), where \(n = |\text{Var}(\mathcal{F})|\), and \(L\) is the length of \(\mathcal{F}\) (i.e. \(L = O(n^2)\)).
- \(O(n^2)\) algorithm for deciding \(\mathcal{F} \in \text{MU}(1)\) in [Davidov et al, '98].
- \(O(n^3)\) algorithm for MU\((2)\) in [Kleine Buning, '00] (also, MU\((k)\) \(\in\) \(NP\)).

**Note:** in practical applications \(k\) is very large.
Complexity Results

**Notation:** MU(k) — the set of MU formulas with deficiency k.

**Theorem** [Fleischner, Kullmann, and Szeider, '02]: For a fixed k, MU(k) ∈ P.

- Specifically: \( \mathcal{F} \in \text{MU}(k) \) can be decided in \( O(L \cdot n^{k+\frac{1}{2}}) \), where \( n = |\text{Var(}\mathcal{F})| \), and L is the length of \( \mathcal{F} \) (i.e. \( L = O(n^2) \)).
- \( O(n^2) \) algorithm for deciding \( \mathcal{F} \in \text{MU}(1) \) in [Davidov et al, '98].
- \( O(n^3) \) algorithm for MU(2) in [Kleine Buning, '00] (also, MU(k) ∈ NP).

**Note:** in practical applications k is very large.

MUS computation: in FP\(^{\text{NP}}\)  
*Note:* i.e. poly number of calls to NP oracle.
Complexity Results

**Notation:** MU($k$) — the set of MU formulas with deficiency $k$.

**Theorem** [Fleischner, Kullmann, and Szeider, '02]: For a fixed $k$, MU($k$) $\in P$.

- Specifically: $\mathcal{F} \in$ MU($k$) can be decided in $O(L \cdot n^{k+\frac{1}{2}})$, where $n = |\text{Var}(\mathcal{F})|$, and $L$ is the length of $\mathcal{F}$ (i.e. $L = O(n^2)$).
- $O(n^2)$ algorithm for deciding $\mathcal{F} \in$ MU(1) in [Davidov et al, '98].
- $O(n^3)$ algorithm for MU(2) in [Kleine Buning, '00] (also, MU($k$) $\in$ NP).

**Note:** in practical applications $k$ is very large.

MUS computation: in FP$^\text{NP}$ **Note:** i.e. poly number of calls to NP oracle.

Practical algorithms for computation of MUSes are based on iterative calls to SAT solver.
Related Problems

**MUS Membership Problem**: does $C \in \mathcal{F}$ belongs to some MUS of $\mathcal{F}$?

- $\Sigma^P_2$-complete [Liberatore, 05] (reduction from $\exists \forall$QBF).
- Related function problem: given $\mathcal{F}$, compute $\bigcup \text{MUS}(\mathcal{F})$. 

---

A. Belov  
MU and MUSes: Theory, Algorithms and Applications  
EPCL Training Camp, 2012  
# 21
Related Problems

*MUS Membership Problem*: does \( C \in \mathcal{F} \) belongs to some MUS of \( \mathcal{F} \)?

- \( \Sigma^P_2 \)-complete [Liberatore, 05] (reduction from \( \exists \forall \text{QBF} \)).
- Related function problem: given \( \mathcal{F} \), compute \( \bigcup \text{MUS}(\mathcal{F}) \).

*Smallest MUS Problem*: compute \( \mathcal{M} \in \text{MUS}(\mathcal{F}) \) of smallest cardinality.

- Related problem: given \( \mathcal{F} \) and \( k \in \mathbb{N} \), is there \( \mathcal{M} \in \text{MUS}(\mathcal{F}) \) such that \( |\mathcal{M}| \leq k \). \( \Sigma^P_2 \)-complete [Liberatore, 05].
Related Problems

**MUS Membership Problem**: does $C \in \mathcal{F}$ belongs to some MUS of $\mathcal{F}$?

- $\Sigma^P_2$-complete [Liberatore, 05] (reduction from $\exists \forall$QBF).
- Related function problem: given $\mathcal{F}$, compute $\bigcup MUS(\mathcal{F})$.

**Smallest MUS Problem**: compute $M \in MUS(\mathcal{F})$ of smallest cardinality.

- Related problem: given $\mathcal{F}$ and $k \in \mathbb{N}$, is there $M \in MUS(\mathcal{F})$ such that $|M| \leq k$. $\Sigma^P_2$-complete [Liberatore, 05].

**MUS Enumeration Problem**: given $\mathcal{F}$, compute the set $MUS(\mathcal{F})$. 

Note: Precise complexity results for ‘many problems are scarce.'
**Related Problems**

**MUS Membership Problem**: does $C \in \mathcal{F}$ belong to some MUS of $\mathcal{F}$?
- $\Sigma^P_2$-complete [Liberatore, 05] (reduction from $\exists \forall$ QBF).
- Related function problem: given $\mathcal{F}$, compute $\bigcup \text{MUS}(\mathcal{F})$.

**Smallest MUS Problem**: compute $\mathcal{M} \in \text{MUS}(\mathcal{F})$ of smallest cardinality.
- Related problem: given $\mathcal{F}$ and $k \in \mathbb{N}$, is there $\mathcal{M} \in \text{MUS}(\mathcal{F})$ such that $|\mathcal{M}| \leq k$. $\Sigma^P_2$-complete [Liberatore, 05].

**MUS Enumeration Problem**: given $\mathcal{F}$, compute the set $\text{MUS}(\mathcal{F})$.

**Note**: Precise complexity results for ‘many problems are scarce.'
Course outline

1. Preliminaries: CNF formulas, notation, SAT solvers.
2. Minimal Unsatisfiability: example, definition.
3. Structure of UNSAT formulas: MUSes, MCSes, hitting sets duality, MaxSAT.
5. Towards the algorithms: categorization of clauses.
7. Applications.
Categorization of Clauses in (UNSAT) CNF

\[ Kullmann \text{ et al, '06} \]

\[ C_1 = (p) \]
\[ C_2 = (q) \]
\[ C_3 = (\neg p \lor \neg q) \]
\[ C_4 = (\neg p \lor r) \]
\[ C_5 = (p \lor q) \]
\[ C_6 = (\neg q \lor \neg r) \]

**Necessary clauses** — belong to all MUSes of \( F \), i.e. in \( \bigcap \text{MUS}(F) \).

- Must be used in every resolution refutation of \( F \).
- If \( F \in \text{MU} \), then all \( C \in F \) are necessary for \( F \).
- If \( C \) is necessary for \( F \), it is necessary for every UNSAT \( F' \subset F \).

**Def:** \( C \in F \) is necessary for \( F \) if \( F \in \text{UNSAT} \) and \( F \{ C \} \in \text{SAT} \).

- To decide whether \( C \) is necessary for \( F \) is NP-complete.

\[ \text{Liberatore, 05} \]

**Note:** \( C \in F \) is necessary for \( F \) iff \( \exists \tau \), such that Unsat \( (F, \tau) = \{ C \} \).

\( \tau \) – a witness of (necessity) of \( C \in F \).
Categorization of Clauses in (UNSAT) CNF [Kullmann et al, ’06]

\[ C_1 = (p) \quad C_3 = (\neg p \lor \neg q) \quad C_5 = (p \lor q) \]

\[ C_2 = (q) \quad C_4 = (\neg p \lor r) \quad C_6 = (\neg q \lor \neg r) \]

UNSAT

**Necessary clauses** — belong to all MUSes of \( \mathcal{F} \), i.e. in \( \bigcap \text{MUS}(\mathcal{F}) \).
Categorization of Clauses in (UNSAT) CNF [Kullmann et al, ’06]

C\(_1\) = (p)

C\(_2\) = (q)

C\(_3\) = (¬p ∨ ¬q)

C\(_4\) = (¬p ∨ r)

C\(_5\) = (p ∨ q)

C\(_6\) = (¬q ∨ ¬r)

**Necessary clauses** — belong to all MUSes of \(\mathcal{F}\), i.e. in \(\bigcap \text{MUS}(\mathcal{F})\).

- Must be used in every resolution refutation of \(\mathcal{F}\).
Categorization of Clauses in (UNSAT) CNF

\[ C_1 = (p) \]
\[ C_2 = (q) \]
\[ C_3 = (\neg p \lor \neg q) \]
\[ C_4 = (\neg p \lor r) \]
\[ C_5 = (p \lor q) \]
\[ C_6 = (\neg q \lor \neg r) \]

\textbf{Necessary clauses} — belong to all MUSes of \( F \), i.e. in \( \bigcap \text{MUS}(F) \).

- Must be used in every resolution refutation of \( F \).
- If \( F \in \text{MU} \), then all \( C \in F \) are necessary for \( F \).
**Categorization of Clauses in (UNSAT) CNF** [Kullmann et al, ’06]

\[ C_1 = (p) \]
\[ C_2 = (q) \]
\[ C_3 = (\neg p \lor \neg q) \]
\[ C_4 = (\neg p \lor r) \]
\[ C_5 = (p \lor q) \]
\[ C_6 = (\neg q \lor \neg r) \]

**Necessary clauses** — belong to *all* MUSes of \( \mathcal{F} \), i.e. in \( \bigcap \text{MUS}(\mathcal{F}) \).

- Must be used in every resolution refutation of \( \mathcal{F} \).
- If \( \mathcal{F} \in \text{MU} \), then all \( C \in \mathcal{F} \) are necessary for \( \mathcal{F} \).
- If \( C \) is necessary for \( \mathcal{F} \), it is necessary for every UNSAT \( \mathcal{F}' \subset \mathcal{F} \).
Categorization of Clauses in (UNSAT) CNF [Kullmann et al, '06]

Necessary clauses — belong to all MUSes of $\mathcal{F}$, i.e. in $\bigcap \text{MUS}(\mathcal{F})$.

- Must be used in every resolution refutation of $\mathcal{F}$.
- If $\mathcal{F} \in \text{MU}$, then all $C \in \mathcal{F}$ are necessary for $\mathcal{F}$.
- If $C$ is necessary for $\mathcal{F}$, it is necessary for every UNSAT $\mathcal{F}' \subset \mathcal{F}$.

**Def:** $C \in \mathcal{F}$ is necessary for $\mathcal{F}$ if $\mathcal{F} \in \text{UNSAT}$ and $\mathcal{F} \setminus \{C\} \in \text{SAT}$.

- To decide whether $C$ is necessary for $\mathcal{F}$ is NP-complete. [Liberatore, 05]
Categorization of Clauses in (UNSAT) CNF [Kullmann et al, ’06]

\[
\begin{align*}
C_1 &= (p) & C_3 &= (\neg p \lor \neg q) & C_5 &= (p \lor q) \\
C_2 &= (q) & C_4 &= (\neg p \lor r) & C_6 &= (\neg q \lor \neg r)
\end{align*}
\]

SAT

Necessary clauses — belong to all MUSes of \( \mathcal{F} \), i.e. in \( \bigcap \text{MUS}(\mathcal{F}) \).

- Must be used in every resolution refutation of \( \mathcal{F} \).
- If \( \mathcal{F} \in \text{MU} \), then all \( C \in \mathcal{F} \) are necessary for \( \mathcal{F} \).
- If \( C \) is necessary for \( \mathcal{F} \), it is necessary for every UNSAT \( \mathcal{F}' \subset \mathcal{F} \).

**Def:** \( C \in \mathcal{F} \) is necessary for \( \mathcal{F} \) if \( \mathcal{F} \in \text{UNSAT} \) and \( \mathcal{F} \setminus \{C\} \in \text{SAT} \).

- To decide whether \( C \) is necessary for \( \mathcal{F} \) is NP-complete. [Liberatore, 05]
### Categorization of Clauses in (UNSAT) CNF

[Kullmann et al, '06]

<table>
<thead>
<tr>
<th>Clause $C_i$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1 = (p)$</td>
<td>Necessary</td>
</tr>
<tr>
<td>$C_2 = (q)$</td>
<td>Necessary</td>
</tr>
<tr>
<td>$C_3 = (¬p \lor ¬q)$</td>
<td>Necessary</td>
</tr>
<tr>
<td>$C_4 = (¬p \lor r)$</td>
<td>Necessary</td>
</tr>
<tr>
<td>$C_5 = (p \lor q)$</td>
<td>Necessary</td>
</tr>
<tr>
<td>$C_6 = (¬q \lor ¬r)$</td>
<td>Necessary</td>
</tr>
</tbody>
</table>

**SAT**

_**Necessary clauses**_ — belong to all MUSes of $\mathcal{F}$, i.e. in $\bigcap \text{MUS}(\mathcal{F})$.

- Must be used in every resolution refutation of $\mathcal{F}$.
- If $\mathcal{F} \in \text{MU}$, then all $C \in \mathcal{F}$ are necessary for $\mathcal{F}$.
- If $C$ is necessary for $\mathcal{F}$, it is necessary for every UNSAT $\mathcal{F}' \subset \mathcal{F}$.

**Def:** $C \in \mathcal{F}$ is necessary for $\mathcal{F}$ if $\mathcal{F} \in \text{UNSAT}$ and $\mathcal{F} \setminus \{C\} \in \text{SAT}$.

- To decide whether $C$ is necessary for $\mathcal{F}$ is NP-complete.[Liberatore, 05]

**Note:** $C \in \mathcal{F}$ is necessary for $\mathcal{F}$ iff $\exists \tau$, such that $\text{Unsat}(\mathcal{F}, \tau) = \{C\}$.

- $\tau$ — a witness of (necessity) of $C \in \mathcal{F}$. 
Categorization of Clauses in (UNSAT) CNF [Kullmann et al, ’06]

$C_1 = (p)$

$C_2 = (q)$

$C_3 = (\neg p \lor \neg q)$

$C_4 = (\neg p \lor r)$

$C_5 = (p \lor q)$

$C_6 = (\neg q \lor \neg r)$

**Potentially necessary clauses** — in some but not all MUSes of $\mathcal{F}$. 

Never necessary clauses — the rest.

▶ May become necessary when some clauses are removed from $\mathcal{F}$.

▶ May be forced into every refutation by removing some clauses from $\mathcal{F}$.

▶ To decide whether $C$ belongs to some MUS is $\Sigma_2^p$-complete. [Liberatore, 05]
Categorization of Clauses in (UNSAT) CNF [Kullmann et al, ’06]

\[
\begin{align*}
C_1 &= (p) \\
C_2 &= (q) \\
C_3 &= (\neg p \lor \neg q) \\
C_4 &= (\neg p \lor r) \\
C_5 &= (p \lor q) \\
C_6 &= (\neg q \lor \neg r)
\end{align*}
\]

**Potentially necessary clauses** — in some but not all MUSes of \(\mathcal{F}\).

- May become necessary when some clauses are removed from \(\mathcal{F}\).
Categorization of Clauses in (UNSAT) CNF [Kullmann et al, ‘06]

\[C_1 = (p) \quad C_3 = (\neg p \lor \neg q) \quad C_5 = (p \lor q)\]

\[C_2 = (q) \quad C_4 = (\neg p \lor r) \quad C_6 = (\neg q \lor \neg r)\]

**UNSAT**

*Potentially necessary clauses* — in some but not all MUSes of \(\mathcal{F}\).

- May become necessary when some clauses are removed from \(\mathcal{F}\).
### Categorization of Clauses in (UNSAT) CNF

[Kullmann et al, '06]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1 = (p)$</td>
<td>$C_3 = (\neg p \lor \neg q)$</td>
<td>$C_5 = (p \lor q)$</td>
</tr>
<tr>
<td>$C_2 = (q)$</td>
<td>$C_4 = (\neg p \lor r)$</td>
<td>$C_6 = (\neg q \lor \neg r)$</td>
</tr>
</tbody>
</table>

SAT

---

**Potentially necessary clauses** — in some but not all MUSes of $\mathcal{F}$.

- May become necessary when some clauses are removed from $\mathcal{F}$. 

Categorization of Clauses in (UNSAT) CNF [Kullmann et al, ’06]

\[
\begin{align*}
C_1 &= (p) & C_3 &= (\neg p \lor \neg q) & C_5 &= (p \lor q) \\
C_2 &= (q) & C_4 &= (\neg p \lor r) & C_6 &= (\neg q \lor \neg r)
\end{align*}
\]

SAT

**Potentially necessary clauses** — in some but not all MUSes of $\mathcal{F}$.

- May become necessary when some clauses are removed from $\mathcal{F}$.
- May be forced into every refutation by removing some clauses from $\mathcal{F}$.
Categorization of Clauses in (UNSAT) CNF [Kullmann et al, '06]

\[
\begin{align*}
C_1 &= (p) & C_3 &= (\neg p \lor \neg q) & C_5 &= (p \lor q) \\
C_2 &= (q) & C_4 &= (\neg p \lor r) & C_6 &= (\neg q \lor \neg r)
\end{align*}
\]

\[
\text{SAT}
\]

**Potentially necessary clauses** — in some but not all MUSes of $\mathcal{F}$.

- May become necessary when some clauses are removed from $\mathcal{F}$.
- May be forced into every refutation by removing some clauses from $\mathcal{F}$.
- To decide whether $C$ belongs to some MUS is $\Sigma^P_2$-complete. [Liberatore, 05]
### Categorization of Clauses in (UNSAT) CNF

[Kullmann et al, '06]

<table>
<thead>
<tr>
<th>Clause</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>$(p)$</td>
</tr>
<tr>
<td>$C_2$</td>
<td>$(q)$</td>
</tr>
<tr>
<td>$C_3$</td>
<td>$\neg p \lor \neg q$</td>
</tr>
<tr>
<td>$C_4$</td>
<td>$\neg p \lor r$</td>
</tr>
<tr>
<td>$C_5$</td>
<td>$p \lor q$</td>
</tr>
<tr>
<td>$C_6$</td>
<td>$\neg q \lor \neg r$</td>
</tr>
</tbody>
</table>

**UNSAT**

---

**Potentially necessary clauses** — in some but not all MUSes of $\mathcal{F}$.
- May become necessary when some clauses are removed from $\mathcal{F}$.
- May be forced into every refutation by removing some clauses from $\mathcal{F}$.
- To decide whether $C$ belongs to some MUS is $\Sigma^P_2$-complete.[Liberatore, 05]

**Never necessary clauses** — the rest.
- Some might be used in resolution refutation; some will never be used.
Course outline

1. Preliminaries: CNF formulas, notation, SAT solvers.
2. Minimal Unsatisfiability: example, definition.
3. Structure of UNSAT formulas: MUSes, MCSes, hitting sets duality, MaxSAT.
5. Towards the algorithms: categorization of clauses.
7. Applications.
Algorithms for Computing MUSes

Practical MUS extraction algorithms are based on detection of necessary clauses (aka *transition clauses*).

▶ C is necessary for $F \in \text{UNSAT}$ if $F \{C\} \in \text{SAT}$.

Notation:
- $\text{SAT}(F)$ — denotes SAT solver call. Returns $true$ if $F \in \text{SAT}$, $false$ otherwise.

Typically classified according to the way the search is performed:
- **Deletion-based** (or **destructive**): necessary clauses are detected on transition from UNSAT to SAT. MUS is built by removing unnecessary clauses.
- **Insertion-based** (or **constructive**): necessary clauses are detected on transition from SAT to UNSAT. MUS is built by adding necessary clauses.
- **Dichotomic** (binary search).

Complexity: in terms of number of SAT solver calls.
Algorithms for Computing MUSes

Practical MUS extraction algorithms are based on detection of necessary clauses (aka *transition clauses*).

Most algorithms use iterative calls to SAT solver for this purpose.

- ▶ $C$ is necessary for $F \in \text{UNSAT}$ if $F \setminus \{C\} \in \text{SAT}$.

Notation:

- SAT($F$) — denotes SAT solver call. Returns true if $F \in \text{SAT}$, false otherwise.

Typically classified according to the way the search is performed:

- ▶ Deletion-based (or destructive): necessary clauses are detected on transition from UNSAT to SAT. MUS is built by removing unnecessary clauses.
- ▶ Insertion-based (or constructive): necessary clauses are detected on transition from SAT to UNSAT. MUS is built by adding necessary clauses.
- ▶ Dichotomic (binary search).

Complexity: in terms of number of SAT solver calls.
Algorithms for Computing MUSes

Practical MUS extraction algorithms are based on detection of necessary clauses (aka *transition clauses*).

Most algorithms use iterative calls to SAT solver for this purpose.

- **C** is necessary for $\mathcal{F} \in \text{UNSAT}$ if $\mathcal{F} \setminus \{C\} \in \text{SAT}$.
- **Notation:** $\text{SAT}(\mathcal{F})$ — denotes SAT solver call. Returns **true** if $\mathcal{F} \in \text{SAT}$, **false** otherwise.
Algorithms for Computing MUSes

Practical MUS extraction algorithms are based on detection of necessary clauses (aka \textit{transition clauses}).

Most algorithms use iterative calls to SAT solver for this purpose.

- $C$ is necessary for $\mathcal{F} \in \text{UNSAT}$ if $\mathcal{F} \setminus \{C\} \in \text{SAT}$.

- \textbf{Notation:} \text{SAT}(\mathcal{F}) — denotes SAT solver call. Returns \textbf{true} if $\mathcal{F} \in \text{SAT}$, \textbf{false} otherwise.

Typically classified according to the way the search is performed:
Algorithms for Computing MUSes

Practical MUS extraction algorithms are based on detection of necessary clauses (aka *transition clauses*).

Most algorithms use iterative calls to SAT solver for this purpose.

- $C$ is necessary for $\mathcal{F} \in \text{UNSAT}$ if $\mathcal{F} \setminus \{C\} \in \text{SAT}$.

- **Notation:** $\text{SAT}(\mathcal{F})$ — denotes SAT solver call. Returns **true** if $\mathcal{F} \in \text{SAT}$, **false** otherwise.

Typically classified according to the way the search is performed:

- **Deletion-based** (or *destructive*): necessary clauses are detected on transition from UNSAT to SAT. MUS is built by removing unnecessary clauses.
Algorithms for Computing MUSes

Practical MUS extraction algorithms are based on detection of necessary clauses (aka transition clauses).

Most algorithms use iterative calls to SAT solver for this purpose.

- $C$ is necessary for $F \in \text{UNSAT}$ if $F \setminus \{C\} \in \text{SAT}$.

Notation: SAT($F$) — denotes SAT solver call. Returns true if $F \in \text{SAT}$, false otherwise.

Typically classified according to the way the search is performed:

- Deletion-based (or destructive): necessary clauses are detected on transition from UNSAT to SAT. MUS is built by removing unnecessary clauses.
- Insertion-based (or constructive): necessary clauses are detected on transition from SAT to UNSAT. MUS is built by adding necessary clauses.
Algorithms for Computing MUSes

Practical MUS extraction algorithms are based on detection of necessary clauses (aka \textit{transition clauses}).

Most algorithms use iterative calls to SAT solver for this purpose.

- $C$ is necessary for $\mathcal{F} \in \text{UNSAT}$ if $\mathcal{F} \setminus \{C\} \in \text{SAT}$.
- \textbf{Notation:} SAT($\mathcal{F}$) --- denotes SAT solver call. Returns \texttt{true} if $\mathcal{F} \in \text{SAT}$, \texttt{false} otherwise.

Typically classified according to the way the search is performed:

- \textit{Deletion-based} (or \textit{destructive}): necessary clauses are detected on transition from UNSAT to SAT. MUS is built by removing unnecessary clauses.
- \textit{Insertion-based} (or \textit{constructive}): necessary clauses are detected on transition from SAT to UNSAT. MUS is built by adding necessary clauses.
- \textit{Dichotomic} (binary search).
Algorithms for Computing MUSes

Practical MUS extraction algorithms are based on detection of necessary clauses (aka \textit{transition clauses}).

Most algorithms use iterative calls to SAT solver for this purpose.

- \( C \) is necessary for \( \mathcal{F} \in \text{UNSAT} \) if \( \mathcal{F} \setminus \{ C \} \in \text{SAT} \).
- \textbf{Notation:} \( \text{SAT}(\mathcal{F}) \) — denotes SAT solver call. Returns \textbf{true} if \( \mathcal{F} \in \text{SAT} \), \textbf{false} otherwise.

Typically classified according to the way the search is performed:

- \textbf{Deletion-based} (or \textit{destructive}): necessary clauses are detected on transition from UNSAT to SAT. MUS is built by removing unnecessary clauses.
- \textbf{Insertion-based} (or \textit{constructive}): necessary clauses are detected on transition from SAT to UNSAT. MUS is built by adding necessary clauses.
- \textbf{Dichotomic} (binary search).

Complexity: in terms of number of SAT solver calls.
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation
Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.
2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation
Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.

2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

\[
\mathcal{M} = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}\}
\]
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation

Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.

2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

$\mathcal{M} = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}\}$
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation

Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ – remove it from $\mathcal{M}$.

2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

$$\mathcal{M} = \{C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}\}$$
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation

Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.

2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

\[
\mathcal{M} = \{C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}\}
\]
Deletion-Based MUS Computation

Init: \( \mathcal{M} = \mathcal{F} \) — MUS overapproximation
Loop: for each clause \( C \in \mathcal{M} \)

1. if \( \mathcal{M} \setminus \{C\} \in \text{UNSAT} \), then \( C \) is not necessary for \( \mathcal{M} \) — remove it from \( \mathcal{M} \).
2. otherwise, \( C \) is necessary, keep it in \( \mathcal{M} \).

\[ \mathcal{M} = \{C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}\} \]
Deletion-Based MUS Computation

Init: \( M = F \) — MUS overapproximation
Loop: for each clause \( C \in M \)

1. if \( M \setminus \{ C \} \in \text{UNSAT} \), then \( C \) is not necessary for \( M \) — remove it from \( M \).

2. otherwise, \( C \) is necessary, keep it in \( M \).

\[ M = \{ C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12} \} \]
Deletion-Based MUS Computation

Init: $M = F$ — MUS overapproximation
Loop: for each clause $C \in M$

1. if $M \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $M$ — remove it from $M$.
2. otherwise, $C$ is necessary, keep it in $M$.

$M = \{C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}\}$
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation

Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.

2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

$\mathcal{M} = \{C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}\}$
Deletion-Based MUS Computation

Init: \( M = \emptyset \) — MUS overapproximation

Loop: for each clause \( C \in M \)

1. if \( M \setminus \{C\} \in \text{UNSAT} \), then \( C \) is not necessary for \( M \) – remove it from \( M \).
2. otherwise, \( C \) is necessary, keep it in \( M \).

\[
M = \{C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}\}
\]
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation
Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.

2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

$\mathcal{M} = \{C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}\}$
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation

Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.

2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

$\mathcal{M} = \{C_6, C_8, C_9, C_{10}, C_{11}, C_{12}\}$
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation
Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.

2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

$\mathcal{M} = \{C_6, C_9, C_{10}, C_{11}, C_{12}\}$
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation

Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.

2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

$$\mathcal{M} = \{C_6, C_{10}, C_{11}, C_{12}\}$$
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation
Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.

2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

$\mathcal{M} = \{C_6, C_{10}, C_{11}, C_{12}\}$
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation

Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.

2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

\[ \mathcal{M} = \{C_6, C_{10}, C_{11}, C_{12}\} \]
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation

Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.

2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

$\mathcal{M} = \{C_6, C_{10}, C_{11}, C_{12}\}$
Deletion-Based MUS Computation

Init: $M = \mathcal{F} —$ MUS overapproximation
Loop: for each clause $C \in M$

1. if $M \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $M$ – remove it from $M$.
2. otherwise, $C$ is necessary, keep it in $M$.

\[ M = \{C_6, C_{10}, C_{11}, C_{12}\} \]
Deletion-Based MUS Computation

Init: $\mathcal{M} = \mathcal{F}$ — MUS overapproximation
Loop: for each clause $C \in \mathcal{M}$

1. if $\mathcal{M} \setminus \{C\} \in \text{UNSAT}$, then $C$ is not necessary for $\mathcal{M}$ — remove it from $\mathcal{M}$.
2. otherwise, $C$ is necessary, keep it in $\mathcal{M}$.

$\mathcal{M} = \{C_6, C_{10}, C_{11}, C_{12}\}$ is an MUS of $\mathcal{F}$
Deletion-Based MUS Computation

**Input**: Unsatisfiable CNF formula $\mathcal{F}$

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

\[
\mathcal{M} \leftarrow \mathcal{F} \quad \text{// MUS over-approximation}
\]

\[
\text{foreach } C \in \mathcal{M} \text{ do} \quad \text{// Inv: tested clauses in } \mathcal{M} \text{ are nec. for } \mathcal{M}
\]

\[
\begin{cases}
\text{if not SAT}(\mathcal{M} \setminus \{C\}) \text{ then} \quad \text{// If UNSAT, } C \text{ is not necessary for } \mathcal{M} \\
\quad \mathcal{M} \leftarrow \mathcal{M} \setminus \{C\}
\end{cases}
\]

\[\text{return } \mathcal{M} \quad \text{// } \mathcal{M} \text{ is an MUS of } \mathcal{F}\]
Deletion-Based MUS Computation

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

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

$M \leftarrow F$  

// MUS over-approximation

foreach $C \in M$ do  

// Inv: tested clauses in $M$ are nec. for $M$

if not SAT($M \setminus \{C\}$) then  

// If UNSAT, $C$ is not necessary for $M$

$M \leftarrow M \setminus \{C\}$

return $M$  

// $M$ is an MUS of $F$

Number of calls to SAT oracle: $\Theta(|F|)$
Deletion-Based MUS Computation

**Input**: Unsatisfiable CNF formula $\mathcal{F}$

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

$\mathcal{M} \leftarrow \mathcal{F}$  

// MUS over-approximation

**foreach** $C \in \mathcal{M}$ **do**  

// Inv: tested clauses in $\mathcal{M}$ are nec. for $\mathcal{M}$

**if** not SAT($\mathcal{M} \setminus \{C\}$) **then**  

// If UNSAT, $C$ is not necessary for $\mathcal{M}$

$\mathcal{M} \leftarrow \mathcal{M} \setminus \{C\}$

**return** $\mathcal{M}$  

// $\mathcal{M}$ is an MUS of $\mathcal{F}$

Number of calls to SAT oracle: $\Theta(|\mathcal{F}|)$

But, certain optimizations make it scale on practical applications **better** than any other algorithm.
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in \text{UNSAT}$ the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.
Insertion-Based MUS Computation

Init: $M = \emptyset$ — MUS under-approximation; $S = \emptyset$ — working formula.

1. while $M \cup S \in \text{SAT}$, pick $C \in F$ and add it to $S$.
2. when $M \cup S \in \text{UNSAT}$ the last clause $C$ added to $S$ is necessary for $M \cup S$.
3. add $C$ to $M$, let $F = S \setminus \{C\}$, and let $S = \emptyset$.
4. if $F = \emptyset$ done, otherwise goto 1.

$M = \emptyset$ \quad $S = \emptyset$ \quad $F = \{C_1, \ldots, C_{12}\}$
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $S = \emptyset$ — working formula.

1. while $\mathcal{M} \cup S \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $S$.

2. when $\mathcal{M} \cup S \in \text{UNSAT}$ the last clause $C$ added to $S$ is necessary for $\mathcal{M} \cup S$.

3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = S \setminus \{C\}$, and let $S = \emptyset$.

4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

\[ \begin{align*}
\mathcal{M} &= \{\} \\
S &= \{\} \\
\mathcal{F} &= \{C_1, \ldots, C_{12}\}
\end{align*} \]
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in \text{UNSAT}$ the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

$\mathcal{M} = \{\} \quad \mathcal{S} = \{C_1\} \quad \mathcal{F} = \{C_1, \ldots, C_{12}\}$
Insertion-Based MUS Computation

Init: $M = \emptyset$ — MUS under-approximation; $S = \emptyset$ — working formula.

1. while $M \cup S \in \text{SAT}$, pick $C \in F$ and add it to $S$.
2. when $M \cup S \in \text{UNSAT}$ the last clause $C$ added to $S$ is necessary for $M \cup S$.
3. add $C$ to $M$, let $F = S \setminus \{C\}$, and let $S = \emptyset$.
4. if $F = \emptyset$ done, otherwise goto 1.

\[
\begin{align*}
&M = \{\} & S = \{C_1, C_2\} & F = \{C_1, \ldots, C_{12}\} \\
&C_1 & C_2 & C_3 & C_4 \\
&C_5 & C_6 & C_7 & C_8 \\
&C_9 & C_{10} & C_{11} & C_{12} \\
\end{align*}
\]
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in \text{UNSAT}$ the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

\[
\begin{align*}
\mathcal{M} &= \{\} & \mathcal{S} &= \{C_1, C_2, C_3\} & \mathcal{F} &= \{C_1, \ldots, C_{12}\}
\end{align*}
\]
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in \text{UNSAT}$ the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

\[
\begin{align*}
\mathcal{M} &= \{\} & \mathcal{S} &= \{C_1, C_2, C_3, C_4\} & \mathcal{F} &= \{C_1, \ldots, C_{12}\}
\end{align*}
\]
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in \text{UNSAT}$ the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

\[
\begin{align*}
\mathcal{M} &= \{\} \quad \mathcal{S} = \{C_1, C_2, C_3, C_4, C_5\} \quad \mathcal{F} = \{C_1, \ldots, C_{12}\}
\end{align*}
\]
Insertion-Based MUS Computation

Init: \( M = \emptyset \) — MUS under-approximation; \( S = \emptyset \) — working formula.

1. while \( M \cup S \in \text{SAT} \), pick \( C \in F \) and add it to \( S \).
2. when \( M \cup S \in \text{UNSAT} \) the last clause \( C \) added to \( S \) is necessary for \( M \cup S \).
3. add \( C \) to \( M \), let \( F = S \setminus \{C\} \), and let \( S = \emptyset \).
4. if \( F = \emptyset \) done, otherwise goto 1.

\[ M = \{\} \quad S = \{C_1, C_2, C_3, C_4, C_5, C_6\} \quad F = \{C_1, \ldots, C_{12}\} \]
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in \text{UNSAT}$ the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

$\mathcal{M} = \{C_6\}$ \quad $\mathcal{S} = \{C_1, C_2, C_3, C_4, C_5, C_6\}$ \quad $\mathcal{F} = \{C_1, \ldots, C_{12}\}$
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in \text{UNSAT}$ the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

$\mathcal{M} = \{C_6\}$  $\mathcal{S} = \{C_1, C_2, C_3, C_4, C_5, C_6\}$  $\mathcal{F} = \{C_1, C_2, C_3, C_4, C_5\}$
Insertion-Based MUS Computation

Init: $M = \emptyset$ — MUS under-approximation; $S = \emptyset$ — working formula.

1. while $M \cup S \in \text{SAT}$, pick $C \in F$ and add it to $S$.
2. when $M \cup S \in \text{UNSAT}$ the last clause $C$ added to $S$ is necessary for $M \cup S$.
3. add $C$ to $M$, let $F = S \setminus \{C\}$, and let $S = \emptyset$.
4. if $F = \emptyset$ done, otherwise goto 1.

\[ M = \{C_6\} \quad S = \{\} \quad F = \{C_1, C_2, C_3, C_4, C_5\} \]
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in$ SAT, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in$ UNSAT the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

\[\mathcal{M} = \{C_6\} \quad \mathcal{S} = \{C_1\} \quad \mathcal{F} = \{C_1, C_2, C_3, C_4, C_5\}\]
Insertion-Based MUS Computation

Init: \( \mathcal{M} = \emptyset \) — MUS under-approximation; \( \mathcal{S} = \emptyset \) — working formula.

1. while \( \mathcal{M} \cup \mathcal{S} \in \text{SAT} \), pick \( C \in \mathcal{F} \) and add it to \( \mathcal{S} \).
2. when \( \mathcal{M} \cup \mathcal{S} \in \text{UNSAT} \) the last clause \( C \) added to \( \mathcal{S} \) is necessary for \( \mathcal{M} \cup \mathcal{S} \).
3. add \( C \) to \( \mathcal{M} \), let \( \mathcal{F} = \mathcal{S} \setminus \{C\} \), and let \( \mathcal{S} = \emptyset \).
4. if \( \mathcal{F} = \emptyset \) done, otherwise goto 1.

\[
\begin{align*}
\mathcal{M} &= \{C_6\} & \mathcal{S} &= \{C_1, C_2\} & \mathcal{F} &= \mathcal{F} = \{C_1, C_2, C_3, C_4, C_5\}
\end{align*}
\]
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in \text{UNSAT}$ the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

$\mathcal{M} = \{C_6\} \quad \mathcal{S} = \{C_1, C_2, C_3\} \quad \mathcal{F} = \{C_1, C_2, C_3, C_4, C_5\}$
Insertion-Based MUS Computation

Init: \( \mathcal{M} = \emptyset \) — MUS under-approximation; \( \mathcal{S} = \emptyset \) — working formula.

1. while \( \mathcal{M} \cup \mathcal{S} \in \text{SAT} \), pick \( C \in \mathcal{F} \) and add it to \( \mathcal{S} \).
2. when \( \mathcal{M} \cup \mathcal{S} \in \text{UNSAT} \) the last clause \( C \) added to \( \mathcal{S} \) is necessary for \( \mathcal{M} \cup \mathcal{S} \).
3. add \( C \) to \( \mathcal{M} \), let \( \mathcal{F} = \mathcal{S} \setminus \{C\} \), and let \( \mathcal{S} = \emptyset \).
4. if \( \mathcal{F} = \emptyset \) done, otherwise goto 1.

\[ \mathcal{M} = \{C_6\} \quad \mathcal{S} = \{C_1, C_2, C_3, C_4\} \quad \mathcal{F} = \{C_1, C_2, C_3, C_4, C_5\} \]
Insertion-Based MUS Computation

Init: \( \mathcal{M} = \emptyset \) — MUS under-approximation; \( \mathcal{S} = \emptyset \) — working formula.

1. while \( \mathcal{M} \cup \mathcal{S} \in \text{SAT} \), pick \( C \in \mathcal{F} \) and add it to \( \mathcal{S} \).
2. when \( \mathcal{M} \cup \mathcal{S} \in \text{UNSAT} \) the last clause \( C \) added to \( \mathcal{S} \) is necessary for \( \mathcal{M} \cup \mathcal{S} \).
3. add \( C \) to \( \mathcal{M} \), let \( \mathcal{F} = \mathcal{S} \setminus \{C\} \), and let \( \mathcal{S} = \emptyset \).
4. if \( \mathcal{F} = \emptyset \) done, otherwise goto 1.

\[
\begin{align*}
\mathcal{M} &= \{C_6\} & \mathcal{S} &= \{C_1, C_2, C_3, C_4, C_5\} & \mathcal{F} &= \{C_1, C_2, C_3, C_4, C_5\}
\end{align*}
\]
Insertion-Based MUS Computation

Init: \( \mathcal{M} = \emptyset \) — MUS under-approximation; \( \mathcal{S} = \emptyset \) — working formula.

1. while \( \mathcal{M} \cup \mathcal{S} \in \text{SAT} \), pick \( C \in \mathcal{F} \) and add it to \( \mathcal{S} \).
2. when \( \mathcal{M} \cup \mathcal{S} \in \text{UNSAT} \) the last clause \( C \) added to \( \mathcal{S} \) is necessary for \( \mathcal{M} \cup \mathcal{S} \).
3. add \( C \) to \( \mathcal{M} \), let \( \mathcal{F} = \mathcal{S} \setminus \{C\} \), and let \( \mathcal{S} = \emptyset \).
4. if \( \mathcal{F} = \emptyset \) done, otherwise goto 1.

\[ \mathcal{M} = \{C_6, C_5\}, \quad \mathcal{S} = \{C_1, C_2, C_3, C_4, C_5\}, \quad \mathcal{F} = \{C_1, C_2, C_3, C_4, C_5\} \]
Insertion-Based MUS Computation

**Init:** \( \mathcal{M} = \emptyset \) — MUS under-approximation; \( \mathcal{S} = \emptyset \) — working formula.

1. while \( \mathcal{M} \cup \mathcal{S} \in \text{SAT} \), pick \( C \in \mathcal{F} \) and add it to \( \mathcal{S} \).
2. when \( \mathcal{M} \cup \mathcal{S} \in \text{UNSAT} \) the last clause \( C \) added to \( \mathcal{S} \) is necessary for \( \mathcal{M} \cup \mathcal{S} \).
3. add \( C \) to \( \mathcal{M} \), let \( \mathcal{F} = \mathcal{S} \setminus \{C\} \), and let \( \mathcal{S} = \emptyset \).
4. if \( \mathcal{F} = \emptyset \) done, otherwise goto 1.

\[
\begin{align*}
\mathcal{M} &= \{C_6, C_5\} & \mathcal{S} &= \{C_1, C_2, C_3, C_4, C_5\} & \mathcal{F} &= \{C_1, C_2, C_3, C_4\}
\end{align*}
\]
Insertion-Based MUS Computation

Init: \( \mathcal{M} = \emptyset \) — MUS under-approximation; \( \mathcal{S} = \emptyset \) — working formula.

1. while \( \mathcal{M} \cup \mathcal{S} \in \text{SAT} \), pick \( C \in \mathcal{F} \) and add it to \( \mathcal{S} \).
2. when \( \mathcal{M} \cup \mathcal{S} \in \text{UNSAT} \) the last clause \( C \) added to \( \mathcal{S} \) is necessary for \( \mathcal{M} \cup \mathcal{S} \).
3. add \( C \) to \( \mathcal{M} \), let \( \mathcal{F} = \mathcal{S} \setminus \{ C \} \), and let \( \mathcal{S} = \emptyset \).
4. if \( \mathcal{F} = \emptyset \) done, otherwise goto 1.

\[
\mathcal{M} = \{ C_6, C_5 \} \quad \mathcal{S} = \{ \} \quad \mathcal{F} = \{ C_1, C_2, C_3, C_4 \}
\]
Insertion-Based MUS Computation

Init: $M = \emptyset$ — MUS under-approximation; $S = \emptyset$ — working formula.

1. while $M \cup S \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $S$.
2. when $M \cup S \in \text{UNSAT}$ the last clause $C$ added to $S$ is necessary for $M \cup S$.
3. add $C$ to $M$, let $\mathcal{F} = S \setminus \{C\}$, and let $S = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

\[ M = \{C_6, C_5\} \quad S = \{C_1\} \quad \mathcal{F} = \{C_1, C_2, C_3, C_4\} \]
Insertion-Based MUS Computation

Init: $M = \emptyset$ — MUS under-approximation; $S = \emptyset$ — working formula.

1. while $M \cup S \in \text{SAT}$, pick $C \in F$ and add it to $S$.
2. when $M \cup S \in \text{UNSAT}$ the last clause $C$ added to $S$ is necessary for $M \cup S$.
3. add $C$ to $M$, let $F = S \setminus \{C\}$, and let $S = \emptyset$.
4. if $F = \emptyset$ done, otherwise goto 1.

$$M = \{C_6, C_5\} \quad S = \{C_1, C_2\} \quad F = \{C_1, C_2, C_3, C_4\}$$
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $S = \emptyset$ — working formula.

1. while $\mathcal{M} \cup S \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $S$.
2. when $\mathcal{M} \cup S \in \text{UNSAT}$ the last clause $C$ added to $S$ is necessary for $\mathcal{M} \cup S$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = S \setminus \{C\}$, and let $S = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

$\mathcal{M} = \{C_6, C_5, C_2\}$  $S = \{C_1, C_2\}$  $\mathcal{F} = \{C_1, C_2, C_3, C_4\}$
Insertion-Based MUS Computation

Init: $M = \emptyset$ — MUS under-approximation; $S = \emptyset$ — working formula.

1. while $M \cup S \in SAT$, pick $C \in F$ and add it to $S$.
2. when $M \cup S \in UNSAT$ the last clause $C$ added to $S$ is necessary for $M \cup S$.
3. add $C$ to $M$, let $F = S \setminus \{C\}$, and let $S = \emptyset$.
4. if $F = \emptyset$ done, otherwise goto 1.

\[ M = \{ C_6, C_5, C_2 \} \quad S = \{ C_1, C_2 \} \quad F = \{ C_1 \} \]
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in \text{UNSAT}$ the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

\[ \mathcal{M} = \{C_6, C_5, C_2\} \quad \mathcal{S} = \{\} \quad \mathcal{F} = \{C_1\} \]
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in \text{UNSAT}$ the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

$\mathcal{M} = \{C_6, C_5, C_2\}$  $\mathcal{S} = \{C_1\}$  $\mathcal{F} = \{C_1\}$
Insertion-Based MUS Computation

Init: \( \mathcal{M} = \emptyset \) — MUS under-approximation; \( \mathcal{S} = \emptyset \) — working formula.

1. while \( \mathcal{M} \cup \mathcal{S} \in \text{SAT} \), pick \( C \in \mathcal{F} \) and add it to \( \mathcal{S} \).
2. when \( \mathcal{M} \cup \mathcal{S} \in \text{UNSAT} \) the last clause \( C \) added to \( \mathcal{S} \) is necessary for \( \mathcal{M} \cup \mathcal{S} \).
3. add \( C \) to \( \mathcal{M} \), let \( \mathcal{F} = \mathcal{S} \setminus \{C\} \), and let \( \mathcal{S} = \emptyset \).
4. if \( \mathcal{F} = \emptyset \) done, otherwise goto 1.

\[
\mathcal{M} = \{C_6, C_5, C_2, C_1\} \quad \mathcal{S} = \{C_1\} \quad \mathcal{F} = \{C_1\}
\]
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in \text{UNSAT}$ the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

\[ \mathcal{M} = \{C_6, C_5, C_2, C_1\} \quad \mathcal{S} = \{C_1\} \quad \mathcal{F} = \{\} \]
Insertion-Based MUS Computation

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\mathcal{S} = \emptyset$ — working formula.

1. while $\mathcal{M} \cup \mathcal{S} \in \text{SAT}$, pick $C \in \mathcal{F}$ and add it to $\mathcal{S}$.
2. when $\mathcal{M} \cup \mathcal{S} \in \text{UNSAT}$ the last clause $C$ added to $\mathcal{S}$ is necessary for $\mathcal{M} \cup \mathcal{S}$.
3. add $C$ to $\mathcal{M}$, let $\mathcal{F} = \mathcal{S} \setminus \{C\}$, and let $\mathcal{S} = \emptyset$.
4. if $\mathcal{F} = \emptyset$ done, otherwise goto 1.

$\mathcal{M} = \{C_6, C_5, C_2, C_1\}$ is an MUS of the input formula $\mathcal{F}$.
Insertion-Based MUS Computation

Input: Unsatisfiable CNF Formula $\mathcal{F}$
Output: $\mathcal{M} \in \text{MUS}(\mathcal{F})$

$\mathcal{M} \leftarrow \emptyset$ \hspace{1cm} // MUS under-approximation

\begin{algorithm}
\textbf{while} $\mathcal{F} \neq \emptyset$ \textbf{do} \\
\hspace{1cm} $S \leftarrow \emptyset$
\hspace{1cm} \textbf{while} SAT($\mathcal{M} \cup S$) \textbf{do} \\
\hspace{2cm} $C \leftarrow \text{PickClause}($$\mathcal{F}$$)$
\hspace{2cm} $S \leftarrow S \cup \{C\}$
\hspace{2cm} \text{M} \leftarrow \text{M} \cup \{C\}$
\hspace{2cm} $\mathcal{F} \leftarrow S \setminus \{C\}$
\hspace{1cm} \text{return} $\mathcal{M}$ \hspace{1cm} // $\mathcal{M}$ is an MUS of $\mathcal{F}$
\end{algorithm}

Inv: $\forall C \in \mathcal{M}$ is nec. for $\mathcal{F} \cup \mathcal{M} \in \text{UNSAT}$

Working formula

Number of calls to SAT oracle: $O(|\mathcal{F}| \times |\mathcal{M}|)$
Worst case ($\mathcal{F} \in \text{MU}$): $\Theta(|\mathcal{F}|^2)$
Insertion-Based MUS Computation

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

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

1. $\mathcal{M} \leftarrow \emptyset$  
   // MUS under-approximation
2. **while** $\mathcal{F} \neq \emptyset$ **do**
   
   2.1. $S \leftarrow \emptyset$
   
   2.2. **while** $\text{SAT}(\mathcal{M} \cup S)$ **do**
   
   2.2.1. $C \leftarrow \text{PickClause}(\mathcal{F})$
   
   2.2.2. $S \leftarrow S \cup \{C\}$
   
   2.2.3. $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$
   
   2.2.4. $\mathcal{F} \leftarrow S \setminus \{C\}$
   
   // Assert: $C$ is nec. for $\mathcal{M} \cup S$

3. **return** $\mathcal{M}$
   // $\mathcal{M}$ is an MUS of $\mathcal{F}$

Number of calls to SAT oracle: $O(\mathcal{|F|} \times |\mathcal{M}|)$
Insertion-Based MUS Computation

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

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

$\mathcal{M} \leftarrow \emptyset$ \hspace{1cm} // MUS under-approximation

while $\mathcal{F} \neq \emptyset$ do

\hspace{1cm} $S \leftarrow \emptyset$ \hspace{2cm} // Inv: $\forall C \in \mathcal{M}$ is nec. for $\mathcal{F} \cup \mathcal{M} \in \text{UNSAT}$

\hspace{1cm} while SAT$(\mathcal{M} \cup S)$ do

\hspace{2cm} $C \leftarrow \text{PickClause}(\mathcal{F})$

\hspace{2cm} $S \leftarrow S \cup \{C\}$

\hspace{2cm} $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$ \hspace{2cm} // Assert: $C$ is nec. for $\mathcal{M} \cup S$

\hspace{2cm} $\mathcal{F} \leftarrow S \setminus \{C\}$

end while

end while

return $\mathcal{M}$ \hspace{1cm} // $\mathcal{M}$ is an MUS of $\mathcal{F}$

Number of calls to SAT oracle: $O(|\mathcal{F}| \times |\mathcal{M}|)$

Worst case ($\mathcal{F} \in \text{MU}$): $\Theta(|\mathcal{F}|^2)$
Insertion-Based MUS Computation

Why bother?

1. In principle, can much faster than deletion-based when there is a lot of small MUSes.
2. Easier SAT solver calls — SAT formulas are easier to solve than UNSAT.

Note: Empirically, with the current optimizations the deletion-based algorithms are orders of magnitude faster than insertion-based.

Some interesting variations of the insertion-based algorithms:
- Addition of redundancy checks [Van Maaren and Wieringa, '08]
- Relaxation-variable based approach [Marques-Silva and Lynce, '11]

A. Belov
Why bother?

1. In principle, can much faster than deletion-based when there is a lot of small MUSes.

Note: Empirically, with the current optimizations the deletion-based algorithms are orders of magnitude faster than insertion-based.

Some interesting variations of the insertion-based algorithms:

- Addition of redundancy checks [Van Maaren and Wieringa, '08]
- Relaxation-variable based approach [Marques-Silva and Lynce, '11]
Why bother?

1. In principle, can much faster than deletion-based when there is a lot of small MUSes.

2. Easier SAT solver calls — SAT formulas are easier to solve than UNSAT.
Insertion-Based MUS Computation

Why bother?

1. In principle, can much faster than deletion-based when there is a lot small MUSes.

2. Easier SAT solver calls — SAT formulas are easier to solve than UNSAT.

Note: Empirically, with the current optimizations the deletion-based algorithms are orders of magnitude faster than insertion-based.
Insertion-Based MUS Computation

Why bother?

1. In principle, can much faster than deletion-based when there is a lot small MUSes.
2. Easier SAT solver calls — SAT formulas are easier to solve than UNSAT.

Note: Empirically, with the current optimizations the deletion-based algorithms are orders of magnitude faster than insertion-based.

Some interesting variations of the insertion-based algorithms:

- Addition of redundancy checks [Van Maaren and Wieringa, '08]
- Relaxation-variable based approach [Marques-Silva and Lynce, '11]
**Def:** A clause $C \in F$ (not necessarily UNSAT) is *redundant* in $F$ if $F \setminus \{C\} \models \{C\}$. $C$ is *irredundant* otherwise.

Alternatively, $C$ is redundant if $F \equiv F \setminus \{C\}$.
Def: A clause $C \in \mathcal{F}$ (not necessarily UNSAT) is **redundant** in $\mathcal{F}$ if $\mathcal{F} \setminus \{C\} \models \{C\}$. $C$ is **irredundant** otherwise.

Alternatively, $C$ is redundant if $\mathcal{F} \equiv \mathcal{F} \setminus \{C\}$.

**Note:** Necessary clauses are a special case of irredundant clauses when $\mathcal{F} \in$ UNSAT. Thus, every clause in an MUS $\mathcal{M}$ of $\mathcal{F}$ is irredundant.
**Def:** A clause $C \in \mathcal{F}$ (not necessarily UNSAT) is **redundant** in $\mathcal{F}$ if $\mathcal{F} \setminus \{C\} \vDash \{C\}$. $C$ is **irredundant** otherwise.

Alternatively, $C$ is redundant if $\mathcal{F} \equiv \mathcal{F} \setminus \{C\}$.

**Note:** Necessary clauses are a special case of irredundant clauses when $\mathcal{F} \in$ UNSAT. Thus, every clause in an MUS $\mathcal{M}$ of $\mathcal{F}$ is irredundant.

**Idea:** [Van Maaren and Wieringa, '08] In insertion-based algorithm, do not add $C \in \mathcal{F}$ to the working formula $\mathcal{S}$ if $\mathcal{M} \cup \mathcal{S} \vDash \{C\}$, as otherwise $C$ would be redundant.
Checking redundancy in the inner loop of the algorithm:

Let $\neg C$ denote $\bigcup_{l \in C} \neg l$.

If $M \cup S \cup \{\neg C\} \in \text{UNSAT}$, then $C$ is redundant $\Rightarrow$ cannot be in the computed MUS.

So, replace $\text{SAT}(M \cup S)$ with $\text{SAT}(M \cup S \cup \{\neg C\})$.

If outcome is true, add $C$ to $S$, otherwise remove it from $F$.

But, then how do we know when to terminate the inner loop?

We don't, but when $M \cup S \in \text{UNSAT}$, every new clause is redundant.

So, we just check every clause of $F$.

Cheap in incremental SAT solvers, they re-use learned clauses.
Insertion-Based MUS with Redundancy Checks

Checking redundancy in the inner loop of the algorithm:

- Let \( \{\neg C\} \) denote \( \bigcup_{l \in C} \{\neg l\} \).

- If \( M \cup S \cup \{\neg C\} \in \text{UNSAT} \), then \( C \) is redundant \( \Rightarrow \) cannot be in the computed MUS.

- So, replace \( \text{SAT} (M \cup S) \) with \( \text{SAT} (M \cup S \cup \{\neg C\}) \).

- If outcome is true, add \( C \) to \( S \), otherwise remove it from \( F \).

- But, then how do we know when to terminate the inner loop?

- We don't, but when \( M \cup S \in \text{UNSAT} \), every new clause is redundant.

- So, we just check every clause of \( F \).

- Cheap in incremental SAT solvers, they re-use learned clauses.
Checking redundancy in the inner loop of the algorithm:

- Let $\{\neg C\}$ denote $\bigcup_{l \in C} \{\neg l\}$.
- If $M \cup S \cup \{\neg C\} \in \text{UNSAT}$, then $C$ is redundant $\Rightarrow$ cannot be in the computed MUS.

But, then how do we know when to terminate the inner loop?

- We don’t, but when $M \cup S \in \text{UNSAT}$, every new clause is redundant.
- So, we just check every clause of $F$.
- Cheap in incremental SAT solvers, they re-use learned clauses.
Checking redundancy in the inner loop of the algorithm:

- Let $\neg C$ denote $\bigcup_{l \in C} \{\neg l\}$.
- If $M \cup S \cup \{\neg C\} \in \text{UNSAT}$, then $C$ is redundant $\Rightarrow$ cannot be in the computed MUS.
- So, replace $\text{SAT}(M \cup S)$ with $\text{SAT}(M \cup S \cup \{\neg C\})$.
Checking redundancy in the inner loop of the algorithm:

- Let \( \{\neg C\} \) denote \( \bigcup_{l \in C} \{\neg l\} \).
- If \( M \cup S \cup \{\neg C\} \in \text{UNSAT} \), then \( C \) is redundant \( \Rightarrow \) cannot be in the computed MUS.
- So, replace \( \text{SAT}(M \cup S) \) with \( \text{SAT}(M \cup S \cup \{\neg C\}) \).
- If outcome is \textbf{true}, add \( C \) to \( S \), otherwise remove it from \( F \).
Checking redundancy in the inner loop of the algorithm:

- Let \( \{\neg C\} \) denote \( \bigcup_{l \in C}\{\neg l\} \).
- If \( M \cup S \cup \{\neg C\} \in \text{UNSAT} \), then \( C \) is redundant \( \Rightarrow \) cannot be in the computed MUS.
- So, replace \( \text{SAT}(M \cup S) \) with \( \text{SAT}(M \cup S \cup \{\neg C\}) \).
- If outcome is \textbf{true}, add \( C \) to \( S \), otherwise remove it from \( F \).

But, then how do we know when to terminate the inner loop?
Insertion-Based MUS with Redundancy Checks

Checking redundancy in the inner loop of the algorithm:

- Let \( \{\neg C\} \) denote \( \bigcup_{l \in C} \{\neg l\} \).
- If \( \mathcal{M} \cup S \cup \{\neg C\} \in UNSAT \), then \( C \) is redundant \( \Rightarrow \) cannot be in the computed MUS.
- So, replace \( \text{SAT}(\mathcal{M} \cup S) \) with \( \text{SAT}(\mathcal{M} \cup S \cup \{\neg C\}) \).
- If outcome is true, add \( C \) to \( S \), otherwise remove it from \( F \).

But, then how do we know when to terminate the inner loop?

- We don’t, but when \( \mathcal{M} \cup S \in UNSAT \), every new clause is redundant.
Checking redundancy in the inner loop of the algorithm:

- Let \( \{\neg C\} \) denote \( \bigcup_{l \in C} \{\neg l\} \).
- If \( M \cup S \cup \{\neg C\} \not\in \text{UNSAT} \), then \( C \) is redundant \( \Rightarrow \) cannot be in the computed MUS.
- So, replace \( \text{SAT}(M \cup S) \) with \( \text{SAT}(M \cup S \cup \{\neg C\}) \).
- If outcome is \textbf{true}, add \( C \) to \( S \), otherwise remove it from \( F \).

But, then how do we know when to terminate the inner loop?

- We don’t, but when \( M \cup S \not\in \text{UNSAT} \), every new clause is redundant.
- So, we just check every clause of \( F \).
Insertion-Based MUS with Redundancy Checks

Checking redundancy in the inner loop of the algorithm:

▶ Let \( \{\neg C\} \) denote \( \bigcup_{l \in C} \{\neg l\} \).
▶ If \( M \cup S \cup \{\neg C\} \in UNSAT \), then \( C \) is redundant \( \Rightarrow \) cannot be in the computed MUS.
▶ So, replace \( SAT(M \cup S) \) with \( SAT(M \cup S \cup \{\neg C\}) \).
▶ If outcome is true, add \( C \) to \( S \), otherwise remove it from \( F \).

But, then how do we know when to terminate the inner loop?

▶ We don’t, but when \( M \cup S \in UNSAT \), every new clause is redundant.
▶ So, we just check every clause of \( F \).
▶ Cheap in incremental SAT solvers, they re-use learned clauses.
Insertion-Based MUS with Redundancy Checks

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

**Output**: MUS $\mathcal{M}$

\[
\mathcal{M} \leftarrow \emptyset \\
\text{while } \mathcal{F} \neq \emptyset \\
\quad S \leftarrow \emptyset \\
\quad \text{foreach } C \in \mathcal{F} \text{ do} \\
\quad \\
\quad \\
\quad \text{if SAT}(\mathcal{M} \cup S \cup \{\neg C\}) \text{ then} \\
\quad \\
\quad \quad S \leftarrow S \cup \{C\} \\
\quad \quad C_n \leftarrow C \\
\quad \mathcal{M} \leftarrow \mathcal{M} \cup \{C_n\} \\
\quad \mathcal{F} \leftarrow S \setminus \{C_n\}
\]

\[
\text{return } \mathcal{M} \\
\]

The same complexity as before, but in practice $\mathcal{M} \cup \mathcal{F}$ shrinks much faster. The extra SAT calls (when $\mathcal{M} \cup S$ is already UNSAT) are cheap.

A. Belov

MU and MUSes: Theory, Algorithms and Applications

EPCL Training Camp, 2012
Insertion-Based MUS with Redundancy Checks

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

**Output**: MUS $\mathcal{M}$

$\mathcal{M} \leftarrow \emptyset$ // MUS under-approximation

**while** $\mathcal{F} \neq \emptyset$ **do** 

$S \leftarrow \emptyset$ // Inv: $\forall C \in \mathcal{M}$ is nec. for $\mathcal{F} \cup \mathcal{M} \in \text{UNSAT}$

**foreach** $C \in \mathcal{F}$ **do**

- **if** SAT$(\mathcal{M} \cup S \cup \{\neg C\})$ **then**
  - $S \leftarrow S \cup \{C\}$ // C is not redundant
  - $C_n \leftarrow C$ // Remember the last irredundant clause

- $\mathcal{M} \leftarrow \mathcal{M} \cup \{C_n\}$ // Assert: $C_n$ is nec. for $\mathcal{M} \cup S$

- $\mathcal{F} \leftarrow S \setminus \{C_n\}$

{return $\mathcal{M}$} // $\mathcal{M}$ is an MUS of $\mathcal{F}$

Same complexity as before, but in practice $\mathcal{M} \cup \mathcal{F}$ shrinks much faster.
Insertion-Based MUS with Redundancy Checks

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

**Output**: MUS $\mathcal{M}$

\[
\mathcal{M} \leftarrow \emptyset \quad \text{// MUS under-approximation}
\]

while $\mathcal{F} \neq \emptyset$ do

\[
S \leftarrow \emptyset \quad \text{// Working formula}
\]

foreach $C \in \mathcal{F}$ do

\[
\text{if SAT}(\mathcal{M} \cup S \cup \{\neg C\}) \text{ then}
\]

\[
S \leftarrow S \cup \{C\}
\]

\[
C_n \leftarrow C \quad \text{// Remember the last irredundant clause}
\]

\[
\mathcal{M} \leftarrow \mathcal{M} \cup \{C_n\} \quad \text{// Assert: } C_n \text{ is nec. for } \mathcal{M} \cup S
\]

\[
\mathcal{F} \leftarrow S \setminus \{C_n\} \quad \text{// Inv: } \forall C \in \mathcal{M} \text{ is nec. for } \mathcal{F} \cup \mathcal{M} \in \text{UNSAT}
\]

return $\mathcal{M}$

Same complexity as before, but in practice $\mathcal{M} \cup \mathcal{F}$ shrinks much faster.

The extra SAT calls (when $\mathcal{M} \cup S$ is already UNSAT) are cheap.
Insertion-Based MUS with Relaxation Variables

**Idea:** [Marques-Silva and Lynce, '11] instead of looking for necessary clauses explicitly, ask the SAT solver to find them.

1. Relax all clauses in $F$:
   - Replace each clause $C \in F$ with a clause $(r \lor C)$ — $r$ is a fresh relaxation variable.
   - Let $F_R \leftarrow \{ (r_i \lor C_i) | C_i \in F \}$

2. Add to $F_R$ a CNF of AtMost1 constraint on relaxation variables:
   \[ \sum r_i \leq 1. \]

Point 1: Setting all $r_i$ to 0 in $F_R$ makes $F_R$ to be equivalent to $F$.

Point 2: Setting some $r_i$ to 1 in $F_R$ makes $F_R$ to be equivalent to $F\{C_i\}$. 

A. Belov  MU and MUSes: Theory, Algorithms and Applications  EPCL Training Camp, 2012  # 35
Insertion-Based MUS with Relaxation Variables


How?
Insertion-Based MUS with Relaxation Variables

**Idea:** [Marques-Silva and Lynce, '11] instead of looking for necessary clauses explicitly, ask the SAT solver to find them.

How?

1. **Relax all clauses in** $\mathcal{F}$:
   - Replace each clause $C \in \mathcal{F}$ with a clause $(r \lor C)$ — $r$ is a fresh relaxation variable.
   - Let $\mathcal{F}^R \leftarrow \{(r_i \lor C_i) \mid C_i \in \mathcal{F}\}$
Insertion-Based MUS with Relaxation Variables


How?

1. Relax all clauses in $\mathcal{F}$:
   - Replace each clause $C \in \mathcal{F}$ with a clause $(r \lor C)$ — $r$ is a fresh relaxation variable.
   - Let $\mathcal{F}^R \leftarrow \{(r_i \lor C_i) \mid C_i \in \mathcal{F}\}$

2. Add to $\mathcal{F}^R$ a CNF of AtMost1 constraint on relaxation variables: $\sum r_i r_i \leq 1$. 
Insertion-Based MUS with Relaxation Variables

**Idea:** [Marques-Silva and Lynce, ’11] instead of looking for necessary clauses explicitly, ask the SAT solver to find them.

How?

1. Relax all clauses in $\mathcal{F}$:
   - Replace each clause $C \in \mathcal{F}$ with a clause $(r \lor C)$ — $r$ is a fresh relaxation variable.
   - Let $\mathcal{F}^R \leftarrow \{(r_i \lor C_i) \mid C_i \in \mathcal{F}\}$

2. Add to $\mathcal{F}^R$ a CNF of \textit{AtMost1} constraint on relaxation variables: $\sum r_i \leq 1$.

**Point 1:** setting all $r_i$ to 0 in $\mathcal{F}^R$ makes $\mathcal{F}^R$ to be equivalent to $\mathcal{F}$. 
Insertion-Based MUS with Relaxation Variables

**Idea:** [Marques-Silva and Lynce, '11] instead of looking for necessary clauses explicitly, ask the SAT solver to find them.

How?

1. Relax all clauses in $\mathcal{F}$:
   - Replace each clause $C \in \mathcal{F}$ with a clause $(r \lor C)$ — $r$ is a fresh relaxation variable.
   - Let $\mathcal{F}^R \leftarrow \{(r_i \lor C_i) \mid C_i \in \mathcal{F}\}$

2. Add to $\mathcal{F}^R$ a CNF of AtMost1 constraint on relaxation variables: $\sum r_i \leq 1$.

**Point 1:** setting all $r_i$ to 0 in $\mathcal{F}^R$ makes $\mathcal{F}^R$ to be equivalent to $\mathcal{F}$.

**Point 2:** setting some $r_i$ to 1 in $\mathcal{F}^R$ makes $\mathcal{F}^R$ to be equivalent to $\mathcal{F} \setminus \{C_i\}$. 
Then, the query $\text{SAT}(F^R)$ means:

is there one (relaxed) clause $(r_i \lor C_i)$ such that $F \setminus \{C_i\} \in \text{SAT}$?
Then, the query $\text{SAT}(F^R)$ means:

is there one (relaxed) clause $(r_i \lor C_i)$ such that $F \setminus \{C_i\} \in \text{SAT}$?

If the outcome is true (SAT), then look at the model $\tau$: the clause $C_i$ for which $\tau(r_i) = 1$ is necessary for $F$, since $F \in \text{UNSAT}$.

- Add $C_i$ to $M$ (the under-approximation of an MUS);
- Un-relax the clause.

If the outcome is false (UNSAT), then either need to remove more than one clause, or already have MUS.

- If there are still relaxed clauses in $F^R$, then remove one.
- Otherwise, the set of un-relaxed clauses are unsatisfiable, i.e. represents an MUS.
Insertion-Based MUS with Relaxation Variables

Then, the query $\text{SAT}(\mathcal{F}^R)$ means:

$$\text{is there one (relaxed) clause } (r_i \lor C_i) \text{ such that } \mathcal{F} \setminus \{C_i\} \in \text{SAT} ?$$

If the outcome is **true** (SAT), then look at the model $\tau$: the clause $C_i$ for which $\tau(r_i) = 1$ is necessary for $\mathcal{F}$, since $\mathcal{F} \not\in \text{UNSAT}$.

- Add $C_i$ to $\mathcal{M}$ (the under-approximation of an MUS);
- Un-relax the clause.

If the outcome is **false** (UNSAT), then either need to remove more than one clause, or already have MUS.

- If there are still relaxed clauses in $\mathcal{F}^R$, then remove one.
- Otherwise, the set of un-relaxed clauses are unsatisfiable, i.e. represents an MUS.
Insertion-Based MUS with Relaxation Variables

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

**Output**: MUS $\mathcal{M}$

\[
\mathcal{M} \leftarrow \emptyset \quad \text{// MUS under-approximation}
\]
\[
\mathcal{F}^R \leftarrow \{(r_i \lor C_i) \mid C_i \in \mathcal{F}\} \quad \text{// $\mathcal{F}^R$ working formula (relaxed clauses)}
\]
\[
\mathcal{T} \leftarrow \text{CNF}(\sum r_i \leq 1) \quad \text{// $\leq 1$ constraint}
\]

while $\mathcal{F}^R \neq \emptyset$ do

\[
(\text{st}, \nu) \leftarrow \text{SAT}(\mathcal{F}^R \cup \mathcal{T} \cup \mathcal{M})
\]

if $\text{st} = \text{true}$ then

\[
r_i \leftarrow \text{TrueRelaxationVariable}(\nu)
\]

else

\[
\mathcal{F}^R \leftarrow \mathcal{F}^R \setminus \text{PickClause}(\mathcal{F}^R) \quad \text{// Remove some un-nec. clause}
\]

return $\mathcal{M}$ \quad \text{// Final $\mathcal{M}$ is an MUS}

Number of calls to SAT oracle: $O(|\mathcal{F}|)$

But SAT calls are more difficult.

A. Belov

MU and MUSes: Theory, Algorithms and Applications

EPCL Training Camp, 2012
Insertion-Based MUS with Relaxation Variables

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

**Output**: MUS $\mathcal{M}$

$\mathcal{M} \leftarrow \emptyset$  
// MUS under-approximation

$\mathcal{F}^R \leftarrow \{(r_i \lor C_i) \mid C_i \in \mathcal{F}\}$  
// $\mathcal{F}^R$ working formula (relaxed clauses)

$\mathcal{T} \leftarrow \text{CNF}(\sum r_i \leq 1)$  
// $\leq 1$ constraint

while $\mathcal{F}^R \neq \emptyset$ do  
// Repeat while relaxed clauses exist

(st, $\nu$) $\leftarrow \text{SAT}(\mathcal{F}^R \cup \mathcal{T} \cup \mathcal{M})$

if st = true then  
// There is a necessary clause

$r_i \leftarrow \text{TrueRelaxationVariable}(\nu)$

else  
// No necessary clauses: remove one

$\mathcal{F}^R \leftarrow \mathcal{F}^R \setminus \text{PickClause}(\mathcal{F}^R)$  
// Remove some un-nec. clause

return $\mathcal{M}$  
// Final $\mathcal{M}$ is an MUS

Number of calls to SAT oracle: $\mathcal{O}(|\mathcal{F}|)$
Insertion-Based MUS with Relaxation Variables

Input: Unsatisfiable CNF Formula $F$
Output: MUS $M$

\[ M \leftarrow \emptyset \]  // MUS under-approximation

\[ F^R \leftarrow \{(r_i \lor C_i) \mid C_i \in F\} \]  // $F^R$ working formula (relaxed clauses)

\[ T \leftarrow \text{CNF}(\sum r_i \leq 1) \]  // $\leq 1$ constraint

while $F^R \neq \emptyset$ do
    (st, $\nu$) $\leftarrow$ SAT($F^R \cup T \cup M$)
    if st = true then
        \[ r_i \leftarrow \text{TrueRelaxationVariable}(\nu) \]  // There is a necessary clause
    else
        \[ F^R \leftarrow F^R \setminus \text{PickClause}(F^R) \]  // No necessary clauses: remove one
            // Remove some un-nec. clause

return $M$  // Final $M$ is an MUS

Number of calls to SAT oracle: $O(|F|)$

But SAT calls are more difficult.
Dichotomic MUS extraction

Init: $M = \emptyset$ — MUS under-approximation; $min = 1$, $max = |F|$.

1. $mid = \lfloor (min + max)/2 \rfloor$
2. if $\{C_1, \ldots, C_{mid}\} \in SAT$, $min = mid + 1$, otherwise $max = mid$
3. when $min = max$, $C_{min}$ is necessary for $M \cup \{C_1, \ldots, C_{min}\} \Rightarrow$ add it to $M$. Stop when $M \in UNSAT$. 
Dichotomic MUS extraction

Init: \( \mathcal{M} = \emptyset \) — MUS under-approximation; \( min = 1, \ max = |\mathcal{F}|. \)

1. \( mid = \lfloor (min + max) / 2 \rfloor \)
2. if \( \{ C_1, \ldots, C_{mid} \} \in SAT \), \( min = mid + 1 \), otherwise \( max = mid \)
3. when \( min = max \), \( C_{min} \) is necessary for \( \mathcal{M} \cup \{ C_1, \ldots, C_{min} \} \Rightarrow \) add it to \( \mathcal{M} \). Stop when \( \mathcal{M} \in UNSAT. \)

\[
\begin{array}{cccc}
C_1 & C_2 & C_3 & C_4 \\
C_5 & C_6 & C_7 & C_8 \\
C_9 & C_{10} & C_{11} & C_{12} \\
\end{array}
\]

\( \mathcal{M} = \{ \} \quad min = 1 \quad max = 12 \)
Dichotomic MUS extraction

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $min = 1$, $max = |\mathcal{F}|$.

1. $mid = \lfloor (\text{min} + \text{max})/2 \rfloor$
2. if $\{C_1, \ldots, C_{mid}\} \in \text{SAT}$, $min = mid + 1$, otherwise $max = mid$
3. when $\text{min} = \text{max}$, $C_{\text{min}}$ is necessary for $\mathcal{M} \cup \{C_1, \ldots, C_{\text{min}}\} \Rightarrow$ add it to $\mathcal{M}$. Stop when $\mathcal{M} \in \text{UNSAT}$.

\[ \mathcal{M} = \{\} \quad mid = 6 \]
Dichotomic MUS extraction

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $min = 1$, $max = |\mathcal{F}|$.

1. $mid = \lfloor (min + max)/2 \rfloor$
2. if $\{C_1, \ldots, C_{mid}\} \in \text{SAT}$, $min = mid + 1$, otherwise $max = mid$
3. when $min = max$, $C_{min}$ is necessary for $\mathcal{M} \cup \{C_1, \ldots, C_{min}\} \Rightarrow$ add it to $\mathcal{M}$. Stop when $\mathcal{M} \in \text{UNSAT}$.

\[\mathcal{M} = \{\} \quad min = 1 \quad max = 6\]
Dichotomic MUS extraction

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $min = 1$, $max = |\mathcal{F}|$.

1. $mid = \lfloor (min + max)/2 \rfloor$
2. if $\{C_1, \ldots, C_{mid}\} \in SAT$, $min = mid + 1$, otherwise $max = mid$
3. when $min = max$, $C_{min}$ is necessary for $\mathcal{M} \cup \{C_1, \ldots, C_{min}\} \Rightarrow$ add it to $\mathcal{M}$. Stop when $\mathcal{M} \in UNSAT$.

$\mathcal{M} = \{\} \quad mid = 3$
Dichotomic MUS extraction

Init: $M = \emptyset$ — MUS under-approximation; $min = 1$, $max = |\mathcal{F}|$.

1. $mid = \lfloor (min + max)/2 \rfloor$

2. if $\{ C_1, \ldots, C_{mid} \} \in SAT$, $min = mid + 1$, otherwise $max = mid$

3. when $min = max$, $C_{min}$ is necessary for $M \cup \{ C_1, \ldots, C_{min} \} \Rightarrow$ add it to $M$. Stop when $M \in UNSAT$.

\[ M = \{ \} \quad \text{min} = 4 \quad \text{max} = 6 \]
Dichotomic MUS extraction

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $min = 1$, $max = |\mathcal{F}|$.

1. $mid = \lfloor (min + max)/2 \rfloor$

2. if $\{C_1, \ldots, C_{mid}\} \in SAT$, $min = mid + 1$, otherwise $max = mid$

3. when $min = max$, $C_{min}$ is necessary for $\mathcal{M} \cup \{C_1, \ldots, C_{min}\} \Rightarrow$ add it to $\mathcal{M}$. Stop when $\mathcal{M} \in UNSAT$.

\[
\begin{array}{cccc}
C_1 & C_2 & C_3 & C_4 \\
C_5 & C_6 & & \\
\end{array}
\]

$\mathcal{M} = \{\} \quad mid = 5$
Dichotomic MUS extraction

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $min = 1$, $max = |\mathcal{F}|$.

1. $mid = \lfloor (min + max)/2 \rfloor$
2. if $\{C_1, \ldots, C_{mid}\} \in SAT$, $min = mid + 1$, otherwise $max = mid$
3. when $min = max$, $C_{min}$ is necessary for $\mathcal{M} \cup \{C_1, \ldots, C_{min}\} \Rightarrow$ add it to $\mathcal{M}$. Stop when $\mathcal{M} \in UNSAT$. 

\[\mathcal{M} = \{\} \quad min = max = 6\]
Dichotomic MUS extraction

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $min = 1$, $max = |\mathcal{F}|$.

1. $mid = \lfloor (min + max)/2 \rfloor$
2. if $\{C_1, \ldots, C_{mid}\} \in SAT$, $min = mid + 1$, otherwise $max = mid$
3. when $min = max$, $C_{min}$ is necessary for $\mathcal{M} \cup \{C_1, \ldots, C_{min}\} \Rightarrow$ add it to $\mathcal{M}$. Stop when $\mathcal{M} \in UNSAT$.

\[ \mathcal{M} = \{C_6\} \quad min = max = 6 \]
Dichotomic MUS extraction

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $\min = 1$, $\max = |\mathcal{F}|$.

1. $\text{mid} = \lfloor (\min + \max) / 2 \rfloor$
2. if $\{C_1, \ldots, C_{\text{mid}}\} \in \text{SAT}$, $\min = \text{mid} + 1$, otherwise $\max = \text{mid}$
3. when $\min = \max$, $C_{\min}$ is necessary for $\mathcal{M} \cup \{C_1, \ldots, C_{\min}\} \Rightarrow$ add it to $\mathcal{M}$. Stop when $\mathcal{M} \in \text{UNSAT}$.

$\mathcal{M} = \{C_6\}$, $\min = 1$, $\max = 5$
Dichotomic MUS extraction

Init: \( \mathcal{M} = \emptyset \) — MUS under-approximation; \( min = 1, \ max = |\mathcal{F}| \).

1. \( \text{mid} = \lfloor (min + max)/2 \rfloor \)
2. if \( \{C_1, \ldots, C_{\text{mid}}\} \in SAT \), \( min = \text{mid} + 1 \), otherwise \( max = \text{mid} \)
3. when \( min = max \), \( C_{\text{min}} \) is necessary for \( \mathcal{M} \cup \{C_1, \ldots, C_{\text{min}}\} \Rightarrow \) add it to \( \mathcal{M} \). Stop when \( \mathcal{M} \in UNSAT \).

\[ \mathcal{M} = \{C_6\} \quad \text{mid} = 3 \]
Dichotomic MUS extraction

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $min = 1$, $max = |\mathcal{F}|$.

1. $mid = \lfloor (min + max)/2 \rfloor$

2. if $\{C_1, \ldots, C_{mid}\} \in SAT$, $min = mid + 1$, otherwise $max = mid$

3. when $min = max$, $C_{min}$ is necessary for $\mathcal{M} \cup \{C_1, \ldots, C_{min}\} \Rightarrow$ add it to $\mathcal{M}$. Stop when $\mathcal{M} \in UNSAT$.

$\mathcal{M} = \{C_6\}$ $min = 4$ $max = 5$
Dichotomic MUS extraction

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $min = 1$, $max = |\mathcal{F}|$.

1. $mid = \lfloor (min + max)/2 \rfloor$
2. if $\{C_1, \ldots, C_{mid}\} \in SAT$, $min = mid + 1$, otherwise $max = mid$
3. when $min = max$, $C_{min}$ is necessary for $\mathcal{M} \cup \{C_1, \ldots, C_{min}\} \Rightarrow$ add it to $\mathcal{M}$. Stop when $\mathcal{M} \in UNSAT.$

\[
\begin{align*}
C_1 & \quad C_2 & \quad C_3 & \quad C_4 \\
C_5 & \quad C_6 & & \\
\text{SAT} & & & \\
\mathcal{M} = \{C_6\} & \quad mid = 4
\end{align*}
\]
Dichotomic MUS extraction

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $min = 1$, $max = |\mathcal{F}|$.

1. $mid = \lfloor (min + max)/2 \rfloor$

2. if $\{C_1, \ldots, C_{mid}\} \in SAT$, $min = mid + 1$, otherwise $max = mid$

3. when $min = max$, $C_{min}$ is necessary for $\mathcal{M} \cup \{C_1, \ldots, C_{min}\} \Rightarrow$ add it to $\mathcal{M}$. Stop when $\mathcal{M} \in UNSAT$.

\[\begin{array}{cccc}
C_1 & C_2 & C_3 & C_4 \\
C_5 & C_6 & \text{UNSAT} & \\
\end{array}\]

$\mathcal{M} = \{C_6\}$, $min = max = 5$
Dichotomic MUS extraction

Init: $\mathcal{M} = \emptyset$ — MUS under-approximation; $min = 1$, $max = |\mathcal{F}|$.

1. $mid = \lfloor (min + max)/2 \rfloor$
2. if $\{C_1, \ldots, C_{mid}\} \in SAT$, $min = mid + 1$, otherwise $max = mid$
3. when $min = max$, $C_{min}$ is necessary for $\mathcal{M} \cup \{C_1, \ldots, C_{min}\} \Rightarrow$ add it to $\mathcal{M}$. Stop when $\mathcal{M} \in UNSAT$.

\[
\begin{align*}
\mathcal{M} &= \{C_6, C_5\} & min = max = 5
\end{align*}
\]
Dichotomic MUS extraction

**Input**: Unsatisfiable CNF Formula $\mathcal{F} = \{C_1, \ldots, C_m\}$

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

$\mathcal{M} \leftarrow \emptyset$  \hspace{1cm} // MUS under-approximation

while SAT($\mathcal{M}$) do

\[ \langle \text{min}, \text{max} \rangle \leftarrow \langle 1, |\mathcal{F}| \rangle \]

while min $\neq$ max do

\[ \text{mid} = \lceil (\text{min} + \text{max}) / 2 \rceil \]

$\mathcal{S} \leftarrow \{C_1, \ldots, C_{\text{mid}}\}$

if SAT($\mathcal{M} \cup \mathcal{S}$) then

\[ \text{min} \leftarrow \text{mid} + 1 \]

else

\[ \text{max} \leftarrow \text{mid} \]

$\mathcal{M} \leftarrow \mathcal{M} \cup \{C_{\text{min}}\}$

$\mathcal{F} \leftarrow \{C_1 \ldots, C_{\text{min} - 1}\}$

\[ \text{Number of calls to SAT oracle: } O(|\mathcal{M}| \times \log(|\mathcal{F}|)) \]

return $\mathcal{M}$  \hspace{1cm} // Final $\mathcal{M}$ is MUS
Dichotomic MUS extraction

**Input**: Unsatisfiable CNF Formula $\mathcal{F} = \{C_1, \ldots, C_m\}$

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

$\begin{align*}
\mathcal{M} \leftarrow \emptyset & \quad \text{// MUS under-approximation} \\
\text{while } \text{SAT}(\mathcal{M}) \text{ do} \\
\quad \langle \text{min, max} \rangle \leftarrow \langle 1, |\mathcal{F}| \rangle \\
\quad \text{while } \text{min} \neq \text{max} \text{ do} \\
\quad \quad \text{mid} = \lfloor (\text{min} + \text{max})/2 \rfloor \\
\quad \quad \mathcal{S} \leftarrow \{C_1, \ldots, C_{\text{mid}}\} \\
\quad \quad \text{if } \text{SAT}(\mathcal{M} \cup \mathcal{S}) \text{ then} \\
\quad \quad \quad \text{min} \leftarrow \text{mid} + 1 \\
\quad \quad \text{else} \\
\quad \quad \quad \text{max} \leftarrow \text{mid} \\
\quad \mathcal{M} \leftarrow \mathcal{M} \cup \{C_{\text{min}}\} \\
\quad \mathcal{F} \leftarrow \{C_1, \ldots, C_{\text{min} - 1}\} \\
\text{return } \mathcal{M} & \quad \text{// Final } \mathcal{M} \text{ is MUS}
\end{align*}$

Number of calls to SAT oracle: $\mathcal{O}(|\mathcal{M}| \times \log(|\mathcal{F}|))$
### Complexity and Performance of MUS Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Calls to SAT Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deletion</td>
<td>$\Theta(</td>
</tr>
<tr>
<td>Insertion</td>
<td>$O(</td>
</tr>
<tr>
<td>Insertion with redundancy checks</td>
<td>$O(</td>
</tr>
<tr>
<td>Insertion with relaxation variables</td>
<td>$O(</td>
</tr>
<tr>
<td>Dichotomic</td>
<td>$O(</td>
</tr>
</tbody>
</table>

In practice, deletion-based solutions currently dominate others. This is due to a number of optimizations.
# Complexity and Performance of MUS Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Calls to SAT Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deletion</td>
<td>$\Theta(</td>
</tr>
<tr>
<td>Insertion</td>
<td>$O(</td>
</tr>
<tr>
<td>Insertion with redundancy checks</td>
<td>$O(</td>
</tr>
<tr>
<td>Insertion with relaxation variables</td>
<td>$O(</td>
</tr>
<tr>
<td>Dichotomic</td>
<td>$O(</td>
</tr>
</tbody>
</table>

In practice, deletion-based solutions currently dominate others.

This is due to a number of optimizations.

- Several similarities between deletion-based and insertion-based algorithms
  - Both identify necessary clauses.
  - In both, $F \cup M$ includes MUS

- How to make SAT solvers calls easier?
  - Redundancy removal — simplifies SAT solver calls.

- Hybrid MUS extraction algorithm is organized to implement these techniques.

- Optimizations take advantage of modern SAT solvers capabilities: unsatisfiable core generation, SAT solving with assumptions.

- Several similarities between deletion-based and insertion-based algorithms
  - Both identify necessary clauses.
  - In both, $\mathcal{F} \cup \mathcal{M}$ includes MUS

- Key issue is how to reduce the number of SAT solver calls

- Clause-set refinement — reduces the number of UNSAT outcomes.
- Model rotation — reduces the number of SAT outcomes.
- How to make SAT solvers calls easier?
- Redundancy removal — simplifies SAT solver calls.

Hybrid MUS extraction algorithm is organized to implement these techniques.

Optimations take advantage of modern SAT solvers capabilities: unsatisfiable core generation, SAT solving with assumptions.
Several similarities between deletion-based and insertion-based algorithms

▶ Both identify necessary clauses.
▶ In both, $\mathcal{F} \cup \mathcal{M}$ includes MUS

Key issue is how to reduce the number of SAT solver calls

▶ *Clause-set refinement* — reduces the number UNSAT outcomes.
Several similarities between deletion-based and insertion-based algorithms

- Both identify necessary clauses.
- In both, $\mathcal{F} \cup \mathcal{M}$ includes MUS

Key issue is how to reduce the number of SAT solver calls

- **Clause-set refinement** — reduces the number UNSAT outcomes.
- **Model rotation** — reduces the number SAT outcomes.

- Several similarities between deletion-based and insertion-based algorithms
  - Both identify necessary clauses.
  - In both, \( \mathcal{F} \cup \mathcal{M} \) includes MUS

- Key issue is how to reduce the number of SAT solver calls
  - *Clause-set refinement* — reduces the number UNSAT outcomes.
  - *Model rotation* — reduces the number SAT outcomes.

- How to make SAT solvers calls easier?
  - *Redundancy removal* — simplifies SAT solver calls.

- Several similarities between deletion-based and insertion-based algorithms
  - Both identify necessary clauses.
  - In both, $\mathcal{F} \cup \mathcal{M}$ includes MUS

- Key issue is how to reduce the number of SAT solver calls
  - *Clause-set refinement* — reduces the number UNSAT outcomes.
  - *Model rotation* — reduces the number SAT outcomes.

- How to make SAT solvers calls easier?
  - *Redundancy removal* — simplifies SAT solver calls.

- Hybrid MUS extraction algorithm is organized to implement these techniques.

- Several similarities between deletion-based and insertion-based algorithms
  - Both identify necessary clauses.
  - In both, $\mathcal{F} \cup \mathcal{M}$ includes MUS

- Key issue is how to reduce the number of SAT solver calls
  - *Clause-set refinement* — reduces the number UNSAT outcomes.
  - *Model rotation* — reduces the number SAT outcomes.

- How to make SAT solvers calls easier?
  - *Redundancy removal* — simplifies SAT solver calls.

- Hybrid MUS extraction algorithm is organized to implement these techniques.

- Optimizations take advantage of modern SAT solvers capabilities: unsatisfiable core generation, SAT solving with assumptions.
Hybrid MUS Extraction w/o optimizations

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

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

$\mathcal{F}' \leftarrow \mathcal{F}$  \hspace{1cm} // Working CNF formula
$\mathcal{M} \leftarrow \emptyset$ \hspace{1cm} // MUS under-approximation

while $\mathcal{F}' \neq \emptyset$ do  \hspace{1cm} // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$

\[ C \leftarrow \text{PickClause}(\mathcal{F}') \]

\[ \text{st} = \text{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\})) \] \hspace{1cm} // Redundancy removal

if $\text{st} = \text{true}$ then

\[ \mathcal{M} \leftarrow \mathcal{M} \cup \{C\} \] \hspace{1cm} // If SAT, C is necessary for $\mathcal{M} \cup \mathcal{F}'$

\[ \text{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau) \] \hspace{1cm} // Recursive model rotation

else

$\mathcal{F}' \leftarrow \mathcal{F}' \setminus \{C\}$ \hspace{1cm} // Clause-set refinement

end if

end while

return $\mathcal{M}$  \hspace{1cm} // $\mathcal{M} \in \text{MUS}(\mathcal{F})$
Hybrid MUS Extraction w/o optimizations

**Input**: Unsatisfiable CNF Formula $F$

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

$$F' \leftarrow F$$  // Working CNF formula

$$M \leftarrow \emptyset$$  // MUS under-approximation

**while** $F' \neq \emptyset$ **do**  // Inv: $M \subseteq F$, and $\forall C \in M$ is nec. for $M \cup F'$

$$C \leftarrow \text{PickClause}(F')$$

$$\text{st} = \text{SAT}(M \cup (F' \setminus \{C\}))$$  // Redundancy removal

**if** $\text{st} = \text{true}$ **then**  // If SAT, $C$ is necessary for $M \cup F'$

$$M \leftarrow M \cup \{C\}$$

$$\text{RMR}(F' \cup M, M, \tau)$$  // Recursive model rotation

**else**

$$F' \leftarrow F' \setminus \{C\}$$  // Clause-set refinement

**return** $M$  // $M \in \text{MUS}(F)$

Essentially deletion-based algorithm, but with insertion-like datastructures.
Optimizations: clause-set refinement/trimming

- **Fact**: Every unsatisfiable formula contains at least one MUS.
- Hence, if $U$ is an unsatisfiable core of $F$, all clauses outside of $U$ can be removed from $F$.
- Relies on the capability of SAT solvers to return unsatisfiable core.
- Effect: remove multiple unnecessary clauses at once.
- Applied to the working formula inside the main loop (e.g. $M \cup F'$ in the Hybrid algorithm) — *clause-set refinement*.
- Applied to the input formula prior to MUS extraction — *clause-set trimming*.
  - Until fix point
  - A fixed number of times
  - Until size change is bounded
Hybrid MUS Extraction: clause-set refinement

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

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

$\mathcal{F}' \leftarrow \mathcal{F}$ \hspace{1cm} // Working CNF formula

$\mathcal{M} \leftarrow \emptyset$ \hspace{1cm} // MUS under-approximation

while $\mathcal{F}' \neq \emptyset$ do \hspace{1cm} // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$

\hspace{1cm} $C \leftarrow \text{PickClause}(\mathcal{F}')$

\hspace{1cm} $\text{st} = \text{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\}))$ \hspace{1cm} // Redundancy removal

\hspace{1cm} if $\text{st} = \text{true}$ then \hspace{1cm} // If SAT, $C$ is necessary for $\mathcal{M} \cup \mathcal{F}'$

\hspace{2cm} $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$ \hspace{1cm} // Recursive model rotation

\hspace{2cm} $\text{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau)$

\hspace{1cm} else\hspace{1cm} // Clause-set refinement

\hspace{2cm} $\mathcal{F}' \leftarrow \mathcal{F}' \setminus \{C\}$

return $\mathcal{M}$ \hspace{1cm} // $\mathcal{M} \in \text{MUS}(\mathcal{F})$
Hybrid MUS Extraction: clause-set refinement

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

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

\[
\begin{align*}
\mathcal{F}' & \leftarrow \mathcal{F} \quad \text{// Working CNF formula} \\
\mathcal{M} & \leftarrow \emptyset \quad \text{// MUS under-approximation} \\
\text{while } \mathcal{F}' \neq \emptyset & \text{ do} \quad \text{// Inv: } \mathcal{M} \subseteq \mathcal{F}, \text{ and } \forall C \in \mathcal{M} \text{ is nec. for } \mathcal{M} \cup \mathcal{F}' \\
& \quad \begin{cases}
C & \leftarrow \text{PickClause}(\mathcal{F}') \\
(st, U) & = \text{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\})) \\
\text{if } st = \text{true} & \text{ then} \quad \text{// If SAT, \( C \) is necessary for } \mathcal{M} \cup \mathcal{F}' \\
& \quad \begin{cases}
\mathcal{M} & \leftarrow \mathcal{M} \cup \{C\} \\
\text{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau) & \text{// Recursive model rotation} \\
\end{cases} \\
\text{else} & \quad \text{// Clause-set refinement} \\
& \quad \mathcal{F}' \leftarrow U \setminus \mathcal{M} \\
\end{cases}
\end{align*}
\]

return $\mathcal{M}$ \quad // $\mathcal{M} \in \text{MUS}(\mathcal{F})$
Impact of clause-set refinement

- 295 benchmarks from track of SAT Competition 2011.
- Time limit 1800 sec, memory limit 4 GB.

- HYB, no optimizations (#sol=132) vs refinement only (#sol=221)
  - Left: number of SAT solver calls. Right: CPU time (sec).
  - Color: MUS size (% of input size).
Optimizations: recursive model rotation (RMR)

▶ **Fact:** $C$ is necessary for $F$ iff $F \in \text{UNSAT}$ and $\exists \tau$ such that $\text{Unsat}(F, \tau) = \{C\}$. $\tau$ is a *witness* (of necessity) for $C$.

▶ During (hybrid) MUS extraction: when $M \cup (F' \setminus \{C\}) \in \text{SAT}$, the assignment $\tau$ found by the SAT solver is a witness for $C$.

▶ Witnesses are also available in other algorithms for MUS extraction.
Optimizations: recursive model rotation (RMR)

Fact: $C$ is necessary for $\mathcal{F}$ iff $\mathcal{F} \in \text{UNSAT}$ and $\exists \tau$ such that $\text{Unsat}(\mathcal{F}, \tau) = \{C\}$. $\tau$ is a witness (of necessity) for $C$.

- During (hybrid) MUS extraction: when $M \cup (\mathcal{F}' \setminus \{C\}) \in \text{SAT}$, the assignment $\tau$ found by the SAT solver is a witness for $C$.
- Witnesses are also available in other algorithms for MUS extraction.

Model rotation [Marques-Silva&Lyence'11]: given a witness $\tau$ for $C$, try to modify it into a witness $\tau'$ for another clause $C'$. How?
Example

\( \mathcal{F} = \{ C_1, \ldots, C_6 \} \)

\( \mathcal{M} \) — an over-approximation of some MUS of \( \mathcal{F} \)

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

\( \mathcal{M} \setminus \{ C_3 \} \in SAT \), hence \( C_3 \) is necessary.
Example

\( \mathcal{F} = \{ C_1, \ldots, C_6 \} \)

\( \mathcal{M} \) — an over-approximation of some MUS of \( \mathcal{F} \)

\[
C_3 = x \lor \neg y \\
C_5 = y \lor z
\]

\[
C_2 = \neg x \lor y \\
C_4 = \neg x \lor \neg y \\
C_6 = y \lor \neg z
\]

\( \mathcal{M} \setminus \{ C_3 \} \in \text{SAT}, \text{ hence } C_3 \text{ is necessary.} \)

SAT solver returns \( \tau = \{ \neg x, y, z \} \)
Example

\[ \mathcal{F} = \{ C_1, \ldots, C_6 \} \]

\[ \mathcal{M} \] is an over-approximation of some MUS of \( \mathcal{F} \)

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

\[ \mathcal{M} \setminus \{ C_3 \} \in SAT, \text{ hence } C_3 \text{ is necessary.} \]

SAT solver returns \( \tau = \{ \neg x, y, z \} \), \( \text{Unsat}(M, \tau) = \{ C_3 \} \).
Example

\[ \mathcal{F} = \{C_1, \ldots, C_6\} \]

\( \mathcal{M} \) — an over-approximation of some MUS of \( \mathcal{F} \)

\[ C_3 = x \lor \neg y \quad C_5 = y \lor z \]

\[ C_2 = \neg x \lor y \quad C_4 = \neg x \lor \neg y \quad C_6 = y \lor \neg z \]

\( \mathcal{M} \setminus \{C_3\} \in \text{SAT} \), hence \( C_3 \) is necessary.

SAT solver returns \( \tau = \{\neg x, y, z\} \), \( \text{Unsat}(\mathcal{M}, \tau) = \{C_3\} \).

Flip \( x \) in \( \tau \): \( \tau' = \{x, y, z\} \)
Example

\[ \mathcal{F} = \{ C_1, \ldots, C_6 \} \]

\( \mathcal{M} \) — an over-approximation of some MUS of \( \mathcal{F} \)

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

\( \mathcal{M} \setminus \{ C_3 \} \in \text{SAT} \), hence \( C_3 \) is necessary.

SAT solver returns \( \tau = \{ \neg x, y, z \} \), \( \text{Unsat}(\mathcal{M}, \tau) = \{ C_3 \} \).

Flip \( x \) in \( \tau \): \( \tau' = \{ x, y, z \} \), \( \text{Unsat}(\mathcal{M}, \tau') = \{ C_4 \} \)
Example

\( \mathcal{F} = \{ C_1, \ldots, C_6 \} \)

\( \mathcal{M} \) — an over-approximation of some MUS of \( \mathcal{F} \)

\[
\begin{align*}
C_3 &= x \lor \neg y \\
C_4 &= \neg x \lor \neg y \\
C_5 &= y \lor z \\
C_6 &= y \lor \neg z
\end{align*}
\]

\( \mathcal{M} \setminus \{ C_3 \} \in \text{SAT}, \text{ hence } C_3 \text{ is necessary.} \)

SAT solver returns \( \tau = \{ \neg x, y, z \} \), \( \text{Unsat}(M, \tau) = \{ C_3 \} \).

Flip \( x \) in \( \tau \): \( \tau' = \{ x, y, z \} \), \( \text{Unsat}(M, \tau') = \{ C_4 \} \rightarrow C_4 \text{ is necessary.} \)
Example

$\mathcal{F} = \{C_1, \ldots, C_6\}$

$\mathcal{M}$ — an over-approximation of some MUS of $\mathcal{F}$

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

$\mathcal{M} \setminus \{C_3\} \in \text{SAT}$, hence $C_3$ is necessary.

SAT solver returns $\tau = \{\neg x, y, z\}$, $\text{Unsat}(\mathcal{M}, \tau) = \{C_3\}$.

Flip $x$ in $\tau$: $\tau' = \{x, y, z\}$, $\text{Unsat}(\mathcal{M}, \tau') = \{C_4\}$ $\rightarrow$ $C_4$ is necessary.

Flip $x$ in $\tau'$: back to $\tau$. $C_3$ is already known to be necessary.
Example

\( \mathcal{F} = \{ C_1, \ldots, C_6 \} \)

\( \mathcal{M} \) — an over-approximation of some MUS of \( \mathcal{F} \)

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

\( \mathcal{M} \setminus \{ C_3 \} \in \text{SAT}, \text{ hence } C_3 \text{ is necessary.} \)

SAT solver returns \( \tau = \{ \neg x, y, z \} \), \( \text{Unsat}(M, \tau) = \{ C_3 \} \).

Flip \( x \) in \( \tau \): \( \tau' = \{ x, y, z \} \), \( \text{Unsat}(M, \tau') = \{ C_4 \} \rightarrow C_4 \text{ is necessary.} \)

Flip \( x \) in \( \tau' \): back to \( \tau \). \( C_3 \) is already known to be necessary.

Flip \( y \) in \( \tau' \): \( \tau'' = \{ x, \neg y, z \} \)
Example

\( \mathcal{F} = \{ C_1, \ldots, C_6 \} \)

\( \mathcal{M} \) — an over-approximation of some MUS of \( \mathcal{F} \)

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

\( \mathcal{M} \setminus \{ C_3 \} \in \text{SAT} \), hence \( C_3 \) is necessary.

SAT solver returns \( \tau = \{ \neg x, y, z \} \), \( \text{Unsat}(\mathcal{M}, \tau) = \{ C_3 \} \).

Flip \( x \) in \( \tau \): \( \tau' = \{ x, y, z \} \), \( \text{Unsat}(\mathcal{M}, \tau') = \{ C_4 \} \) \( \rightarrow \) \( C_4 \) is necessary.

Flip \( x \) in \( \tau' \): back to \( \tau \). \( C_3 \) is already known to be necessary.

Flip \( y \) in \( \tau' \): \( \tau'' = \{ x, \neg y, z \} \), \( \text{Unsat}(\mathcal{M}, \tau'') = \{ C_2, C_6 \} \).
Example

\( \mathcal{F} = \{C_1, \ldots, C_6\} \)

\( \mathcal{M} \) — an over-approximation of some MUS of \( \mathcal{F} \)

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

\( \mathcal{M} \setminus \{C_3\} \in \text{SAT}, \text{ hence } C_3 \text{ is necessary.} \)

SAT solver returns \( \tau = \{\neg x, y, z\} \), \( \text{Unsat}(M, \tau) = \{C_3\} \).

Flip \( x \) in \( \tau \): \( \tau' = \{x, y, z\} \), \( \text{Unsat}(M, \tau') = \{C_4\} \rightarrow C_4 \text{ is necessary.} \)

Flip \( x \) in \( \tau' \): back to \( \tau \). \( C_3 \) is already known to be necessary.

Flip \( y \) in \( \tau' \): \( \tau'' = \{x, \neg y, z\} \), \( \text{Unsat}(M, \tau'') = \{C_2, C_6\} \).

Tried all variables in \( C_4 \) — stop.
Example

\[ \mathcal{F} = \{ C_1, \ldots, C_6 \} \]

\( \mathcal{M} \) — an over-approximation of some MUS of \( \mathcal{F} \)

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

\( \mathcal{M} \setminus \{ C_3 \} \in \text{SAT} \), hence \( C_3 \) is necessary.

SAT solver returns \( \tau = \{ \neg x, y, z \} \), \( \text{Unsat}(\mathcal{M}, \tau) = \{ C_3 \} \).

Flip \( x \) in \( \tau \): \( \tau' = \{ x, y, z \} \), \( \text{Unsat}(\mathcal{M}, \tau') = \{ C_4 \} \rightarrow C_4 \) is necessary.

Flip \( x \) in \( \tau' \): back to \( \tau \). \( C_3 \) is already known to be necessary.

Flip \( y \) in \( \tau' \): \( \tau'' = \{ x, \neg y, z \} \), \( \text{Unsat}(\mathcal{M}, \tau'') = \{ C_2, C_6 \} \).

\( C_4 \) is necessary, without SAT solver call.
Idea: when model rotation stops, backtrack to a necessary clause detected earlier and flip another variable.

Motivation:

Fact: let $\tau$ be a witness for $C$ in $\mathcal{F}$, that is $\text{Unsat}(\mathcal{F}, \tau) = \{C\}$. Then, the sets $\text{Unsat}(\mathcal{F}, \tau|_{\neg x})$ for $x \in \text{Var}(C)$ are pairwise disjoint.

- By flipping different variables we are likely to detect new necessary clauses.
Example

\( F = \{ C_1, \ldots, C_6 \} \)

\( \mathcal{M} \) (the over-approximation of some MUS of \( F \)):

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

\( \mathcal{M} \setminus \{ C_3 \} \in \text{SAT} \), hence \( C_3 \) is necessary.

SAT solver returns \( \tau = \{ \neg x, y, z \} \), \( \text{Unsat}(M, \tau) = \{ C_3 \} \).

Flip \( x \) in \( \tau \): \( \tau' = \{ x, y, z \} \), \( \text{Unsat}(M, \tau') = \{ C_4 \} \) → \( C_4 \) is necessary.

Flip \( x \) in \( \tau' \): back to \( \tau \). \( C_3 \) is already known to be necessary.

Flip \( y \) in \( \tau' \): \( \tau'' = \{ x, \neg y, z \} \), \( \text{Unsat}(M, \tau'') = \{ C_2, C_6 \} \).

Tried all variables in \( C_4 \) — stop, go back to \( C_3 \) and \( \tau \).
Example

\[ F = \{ C_1, \ldots, C_6 \} \]

\( \mathcal{M} \) (the over-approximation of some MUS of \( F \)):

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

\( \mathcal{M} \setminus \{ C_3 \} \in \text{SAT} \), hence \( C_3 \) is necessary.

SAT solver returns \( \tau = \{ \neg x, y, z \} \), \( \text{Unsat}(M, \tau) = \{ C_3 \} \).

Flip \( x \) in \( \tau \): \( \tau' = \{ x, y, z \} \), \( \text{Unsat}(M, \tau') = \{ C_4 \} \rightarrow C_4 \) is necessary.
Example

\( F = \{C_1, \ldots, C_6\} \)

\( \mathcal{M} \) (the over-approximation of some MUS of \( F \)):

\[ C_3 = x \lor \neg y \quad C_5 = y \lor z \]

\[ C_2 = \neg x \lor y \quad C_4 = \neg x \lor \neg y \quad C_6 = y \lor \neg z \]

\( \mathcal{M} \setminus \{C_3\} \in \text{SAT} \), hence \( C_3 \) is necessary.

SAT solver returns \( \tau = \{\neg x, y, z\}, \ Unsat(M, \tau) = \{C_3\} \).

Flip \( x \) in \( \tau \): \( \tau' = \{x, y, z\} \), \( Unsat(M, \tau') = \{C_4\} \rightarrow C_4 \) is necessary.

Flip \( y \) in \( \tau \): \( \tau' = \{\neg x, \neg y, z\} \)
Example

\[ F = \{ C_1, \ldots, C_6 \} \]

\( \mathcal{M} \) (the over-approximation of some MUS of \( F \)):

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

\( \mathcal{M} \setminus \{ C_3 \} \in \text{SAT} \), hence \( C_3 \) is necessary.

SAT solver returns \( \tau = \{ \neg x, y, z \} \), \( \text{Unsat}(\mathcal{M}, \tau) = \{ C_3 \} \).

Flip \( x \) in \( \tau \): \( \tau' = \{ x, y, z \} \), \( \text{Unsat}(\mathcal{M}, \tau') = \{ C_4 \} \rightarrow C_4 \) is necessary.

Flip \( y \) in \( \tau \): \( \tau' = \{ \neg x, \neg y, z \} \), \( \text{Unsat}(\mathcal{M}, \tau') = \{ C_6 \} \rightarrow C_6 \) is necessary.
Example

\[ F = \{ C_1, \ldots, C_6 \} \]

\( M \) (the over-approximation of some MUS of \( F \)):

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

\( M \setminus \{ C_3 \} \in \text{SAT} \), hence \( C_3 \) is necessary.

SAT solver returns \( \tau = \{ \neg x, y, z \} \), \( \text{Unsat}(M, \tau) = \{ C_3 \} \).

Flip \( x \) in \( \tau \): \( \tau' = \{ x, y, z \} \), \( \text{Unsat}(M, \tau') = \{ C_4 \} \rightarrow C_4 \) is necessary.

Flip \( y \) in \( \tau \): \( \tau' = \{ \neg x, \neg y, z \} \), \( \text{Unsat}(M, \tau') = \{ C_6 \} \rightarrow C_6 \) is necessary.

Flip \( z \) in \( \tau' \): \( \tau'' = \{ \neg x, \neg y, \neg z \} \).
Example

\[ F = \{ C_1, \ldots, C_6 \} \]

\( \mathcal{M} \) (the over-approximation of some MUS of \( F \)):

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

\( \mathcal{M} \setminus \{ C_3 \} \in \text{SAT} \), hence \( C_3 \) is necessary.

SAT solver returns \( \tau = \{ \neg x, y, z \} \), \( \text{Unsat}(M, \tau) = \{ C_3 \} \).

Flip \( x \) in \( \tau \): \( \tau' = \{ x, y, z \} \), \( \text{Unsat}(M, \tau') = \{ C_4 \} \rightarrow C_4 \) is necessary.

Flip \( y \) in \( \tau \): \( \tau' = \{ \neg x, \neg y, z \} \), \( \text{Unsat}(M, \tau') = \{ C_6 \} \rightarrow C_6 \) is necessary.

Flip \( z \) in \( \tau' \): \( \tau'' = \{ \neg x, \neg y, \neg z \} \), \( \text{Unsat}(M, \tau'') = \{ C_5 \} \rightarrow C_5 \) is necessary.
Example

\[ F = \{ C_1, \ldots, C_6 \} \]

\( \mathcal{M} \) (the over-approximation of some MUS of \( F \)):

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

\( \mathcal{M} \setminus \{ C_3 \} \in \text{SAT} \), hence \( C_3 \) is necessary.

SAT solver returns \( \tau = \{ \neg x, y, z \} \), \( \text{Unsat}(M, \tau) = \{ C_3 \} \).

Flip \( x \) in \( \tau \): \( \tau' = \{ x, y, z \} \), \( \text{Unsat}(M, \tau') = \{ C_4 \} \rightarrow C_4 \) is necessary.

Flip \( y \) in \( \tau \): \( \tau' = \{ \neg x, \neg y, z \} \), \( \text{Unsat}(M, \tau') = \{ C_6 \} \rightarrow C_6 \) is necessary.

Flip \( z \) in \( \tau' \): \( \tau'' = \{ \neg x, \neg y, \neg z \} \), \( \text{Unsat}(M, \tau'') = \{ C_5 \} \rightarrow C_5 \) is necessary.

\( C_4, C_5, C_6 \) are necessary, without SAT solver call.
Recursive Model Rotation (RMR)

**Input**: $\mathcal{F}$ — an unsatisfiable CNF formula

$\mathcal{M} \subseteq \mathcal{F}$ — a set of transition clauses of $\mathcal{F}$

$\tau$ — a model of $\mathcal{F} \setminus \{C\}$ for some $C \in \mathcal{M}$

**Effect**: $\mathcal{M}$ may contain additional transition clauses of $\mathcal{F}$

$C \leftarrow$ the single clause in $\text{Unsat}(\mathcal{F}, \tau)$

**foreach** $x \in \text{Var}(C)$ **do**

$\tau' \leftarrow \tau|_{\neg x}$

**if** $\text{Unsat}(\mathcal{F}, \tau') = \{C'\}$ **and** $C' \notin \mathcal{M}$ **then**

$\mathcal{M} \leftarrow \mathcal{M} \cup \{C'\}$

RMR($\mathcal{F}, \mathcal{M}, \tau'$)
Recursive Model Rotation (RMR)

**Input**: \( \mathcal{F} \) — an unsatisfiable CNF formula

: \( \mathcal{M} \subseteq \mathcal{F} \) — a set of transition clauses of \( \mathcal{F} \)

: \( \tau \) — a model of \( \mathcal{F} \setminus \{C\} \) for some \( C \in \mathcal{M} \)

**Effect**: \( \mathcal{M} \) may contain additional transition clauses of \( \mathcal{F} \)

\[ C \leftarrow \text{the single clause in } \text{Unsat}(\mathcal{F}, \tau) \]

**foreach** \( x \in \text{Var}(C) \) **do**

\[ \tau' \leftarrow \tau \big|_x \]

**if** \( \text{Unsat}(\mathcal{F}, \tau') = \{C'\} \) **and** \( C' \notin \mathcal{M} \) **then**

\[ \mathcal{M} \leftarrow \mathcal{M} \cup \{C'\} \]

RMR \((\mathcal{F}, \mathcal{M}, \tau')\)

The second condition of **if** keeps the number of the recursive calls linear in the size of computed MUS.
Hybrid MUS Extraction: RMR

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

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

1. $\mathcal{F}' \leftarrow \mathcal{F}$ \hspace{1cm} // Working CNF formula
2. $\mathcal{M} \leftarrow \emptyset$ \hspace{1cm} // MUS under-approximation

while $\mathcal{F}' \neq \emptyset$ do \hspace{1cm} // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$

   1. $C \leftarrow \text{PickClause}(\mathcal{F}')$
   2. $(\text{st}, \mathcal{U}) = \text{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\}))$ \hspace{1cm} // Redundancy removal
   3. if $\text{st} = \text{true}$ then \hspace{1cm} // If SAT, $C$ is necessary for $\mathcal{M} \cup \mathcal{F}'$
      1. $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$
      2. $\text{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau)$ \hspace{1cm} // Recursive model rotation
   4. else
      1. $\mathcal{F}' \leftarrow \mathcal{U} \setminus \mathcal{M}$ \hspace{1cm} // Clause-set refinement

return $\mathcal{M}$ \hspace{1cm} // $\mathcal{M} \in \text{MUS}(\mathcal{F})$
Hybrid MUS Extraction: RMR

Input: Unsatisfiable CNF Formula $\mathcal{F}$

Output: $\mathcal{M} \in \text{MUS}(\mathcal{F})$

$\mathcal{F}' \leftarrow \mathcal{F}$  
$\mathcal{M} \leftarrow \emptyset$

while $\mathcal{F}' \neq \emptyset$ do
  // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$
  $C \leftarrow \text{PickClause}(\mathcal{F}')$
  $(\text{st}, \mathcal{U}, \tau) = \text{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\}))$  
  if st = true then
    // If SAT, C is necessary for $\mathcal{M} \cup \mathcal{F}'$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$
    $\text{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau)$  
  else
    // Clause-set refinement
    $\mathcal{F}' \leftarrow \mathcal{U} \setminus \mathcal{M}$

return $\mathcal{M}$  

// $\mathcal{M} \in \text{MUS}(\mathcal{F})$
Impact of recursive model rotation

- 295 benchmarks from track of SAT Competition 2011.
- Time limit 1800 sec, memory limit 4 GB.

- HYB, refinement only (#sol=221) vs refinement+RMR (#sol=254)
  - Left: number of SAT solver calls. Right: CPU time (sec).
  - Color: MUS size (% of input size).

A. Belov
MU and MUSes: Theory, Algorithms and Applications
EPCL Training Camp, 2012
# 52
Optimizations: redundancy removal

▶ **Fact:** If $\mathcal{F} \in \text{UNSAT}$, then $\mathcal{F} \setminus \{C\} \equiv \mathcal{F} \setminus \{C\} \cup \{\neg C\}$.

▶ $\{\neg C\}$ stands for $\bigcup_{l \in C} \neg l$.

▶ During (hybrid) MUS extraction: add $\{\neg C\}$ to the formula before SAT solver call.

▶ Similar to Insertion with Redundancy Checks, however the purpose is not to detect redundant clauses.

▶ Effect: make SAT calls easier.

▶ But: if $\mathcal{F} \setminus \{C\} \cup \{\neg C\} \in \text{UNSAT}$ and any of the literals from $\{\neg C\}$ are in the unsatisfiable core $\mathcal{U}$, the core cannot be safely used for refinement ($\mathcal{F} \cap \mathcal{U}$ may be SAT).

▶ Adaptive approach: if a core is “tainted”, disable redundancy removal until the next SAT outcome.
Hybrid MUS Extraction: redundancy removal

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

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

$\mathcal{F}' \leftarrow \mathcal{F}$  \quad // Working CNF formula

$\mathcal{M} \leftarrow \emptyset$  \quad // MUS under-approximation

while $\mathcal{F}' \neq \emptyset$ do  \quad // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$

  $C \leftarrow \text{PickClause}(\mathcal{F}')$

  $(\text{st}, \mathcal{U}, \tau) = \text{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\}))$  \quad // Redundancy removal

  if $\text{st} = \text{true}$ then  \quad // If SAT, $C$ is necessary for $\mathcal{M} \cup \mathcal{F}'$

    $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$

    $\text{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau)$  \quad // Recursive model rotation

  else

    $\mathcal{F}' \leftarrow \mathcal{U} \setminus \mathcal{M}$  \quad // Clause-set refinement

return $\mathcal{M}$  \quad // $\mathcal{M} \in \text{MUS}(\mathcal{F})$
Hybrid MUS Extraction: redundancy removal

**Input**: Unsatisfiable CNF Formula $\mathcal{F}$

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

$\mathcal{F}' \leftarrow \mathcal{F}$  \hspace{1cm} // Working CNF formula

$\mathcal{M} \leftarrow \emptyset$  \hspace{1cm} // MUS under-approximation

while $\mathcal{F}' \neq \emptyset$ do  \hspace{1cm} // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$

\[ C \leftarrow \text{PickClause}(\mathcal{F}') \]

\[ (st, \tau, \mathcal{U}) = \text{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\}) \cup \{\neg C\}) \]  \hspace{1cm} // Redundancy removal

\[
\begin{array}{l}
\text{if } st = \text{true} \text{ then} \\
\quad \mathcal{M} \leftarrow \mathcal{M} \cup \{C\} \\
\quad \text{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau) \quad \text{// Recursive model rotation}
\end{array}
\]

\[
\begin{array}{l}
\text{else if } \mathcal{U} \cap \{\neg C\} = \emptyset \text{ then} \\
\quad \mathcal{F}' \leftarrow \mathcal{U} \setminus \mathcal{M} \quad \text{// If the core is ‘‘clean’’} \\
\quad \text{// Clause-set refinement}
\end{array}
\]

return $\mathcal{M}$  \hspace{1cm} // $\mathcal{M} \in \text{MUS}(\mathcal{F})$
Impact of (adaptive) redundancy removal

- 295 benchmarks from track of SAT Competition 2011.
- Time limit 1800 sec, memory limit 4 GB.

- HYB, refinement+RMR (\#sol=254) vs ref+RMR+rra (\#sol=260)
  - Left: avg. time per SAT call (msec). Right: CPU time (sec).
  - Color: MUS size (% of input size).
Performance of different algorithms: run-time

- 295 benchmarks used in the MUS track of SAT Competition 2011.
- Time limit 1800 sec, memory limit 4 GB.
Heuristics

Local-search based: clauses that are falsified often are likely to belong to some MUS [Mazure et al, '98].

- Sure: for any $\tau$, $Unsat(\mathcal{F}, \tau)$ is an over-approximation of one or more MCSes. And hence (by hitting sets duality) must be a hitting set of MUS($\mathcal{F}$).
Heuristics

Local-search based: clauses that are falsified often are likely to belong to some MUS [Mazure et al, '98].

- Sure: for any $\tau$, $Unsat(\mathcal{F}, \tau)$ is an over-approximation of one or more MCSes. And hence (by hitting sets duality) must be a hitting set of $MUS(\mathcal{F})$.

Refinement of this idea: pay attention to the critical clauses [Gregoire et al, '07].

- $C \in \mathcal{F}$ is critical under $\tau$ if $\tau(C) = 0$, and for any $x \in Var(C)$, $\tau|_{\neg x}$ will falsify a clause satisfied by $\tau$.
- Every clause in MU formula is critical for some $\tau$ (it is for the witness assignment).
- Every clause falsified in a local minima of LS algorithm is critical.
Heuristics

Local-search based: clauses that are falsified often are likely to belong to some MUS [Mazure et al, '98].

▶ Sure: for any $\tau$, $\text{Unsat}(\mathcal{F}, \tau)$ is an over-approximation of one or more MCSes. And hence (by hitting sets duality) must be a hitting set of MUS($\mathcal{F}$).

Refinement of this idea: pay attention to the critical clauses [Gregoire et al, '07].

▶ $C \in \mathcal{F}$ is critical under $\tau$ if $\tau(C) = 0$, and for any $x \in \text{Var}(C)$, $\tau|_{\neg x}$ will falsify a clause satisfied by $\tau$.

▶ Every clause in MU formula is critical for some $\tau$ (it is for the witness assignment).

▶ Every clause falsified in a local minima of LS algorithm is critical.

Appears to work on some instances.
Heuristics

Local-search based: clauses that are falsified often are likely to belong to some MUS [Mazure et al, ’98].

- Sure: for any $\tau$, $\text{Unsat}(\mathcal{F}, \tau)$ is an over-approximation of one or more MCSes. And hence (by hitting sets duality) must be a hitting set of MUS($\mathcal{F}$).

Refinement of this idea: pay attention to the critical clauses [Gregoire et al, ’07].

- $C \in \mathcal{F}$ is critical under $\tau$ if $\tau(C) = 0$, and for any $x \in \text{Var}(C)$, $\tau|_{\neg x}$ will falsify a clause satisfied by $\tau$.
- Every clause in MU formula is critical for some $\tau$ (it is for the witness assignment).
- Every clause falsified in a local minima of LS algorithm is critical.

Appears to work on some instances.

Problem 1: SLS is painfully ineffective on practical instances.
Heuristics

Local-search based: clauses that are falsified often are likely to belong to some MUS [Mazure et al, '98].

- Sure: for any $\tau$, $Unsat(\mathcal{F}, \tau)$ is an over-approximation of one or more MCSes. And hence (by hitting sets duality) must be a hitting set of MUS($\mathcal{F}$).

Refinement of this idea: pay attention to the critical clauses [Gregoire et al, '07].

- $C \in \mathcal{F}$ is critical under $\tau$ if $\tau(C) = 0$, and for any $x \in Var(C)$, $\tau|_{\neg x}$ will falsify a clause satisfied by $\tau$.
- Every clause in MU formula is critical for some $\tau$ (it is for the witness assignment).
- Every clause falsified in a local minima of LS algorithm is critical.

Appears to work on some instances.

Problem 1: SLS is painfully ineffective on practical instances.

Problem 2: its not quite clear whether going to necessary clauses is a good idea in the first place.
Some Recent MUS Extractors

**MUSer2** [A. Belov and J. Marques-Silva] — many different algorithms (deletion, insertion, dichotomic, and many variations), all optimizations are integrated, can use different SAT solvers.

**Haifa-MUC** [V. Ryvchin] — deletion-based algorithm that manipulates resolution proofs, built on top of minisat-2.2.

**MoUsSaka** [S. Kottler] — deletion-based, uses activity-based heuristic.

**picomus** [A. Biere] — deletion-based, on top of picosat.

**SAT4J** [D. Le Berre] — a number of different algorithms, Java-based.
Applications: Formal Equivalence Checking (FEC)

- FEC is a technique for formally proving the equivalence of two design models (e.g. RTL “golden” model vs the implementation).

FEC has to be performed compositionally: separate the models into small parts (slices), and the equivalence between the slices with BDD or SAT-based FEC engine.

Note: any slice in isolation can have more behaviours than when it is part of the complete model (e.g. some combinations of inputs are not possible).

FEC is performed under the environmental assumptions which mimic the essential behaviour of the complete model with respect to the slice.
Applications: Formal Equivalence Checking (FEC)

- FEC is a technique for formally proving the equivalence of two design models (e.g. RTL “golden” model vs the implementation).
- FEC has to be performed compositionally: separate the models into small parts (slices), and the equivalence between the slices with BDD or SAT-based FEC engine.
Applications: Formal Equivalence Checking (FEC)

- FEC is a technique for formally proving the equivalence of two design models (e.g. RTL “golden” model vs the implementation).
- FEC has to be performed compositionally: separate the models into small parts (slices), and the equivalence between the slices with BDD or SAT-based FEC engine.
- **Note**: any slice in isolation can have more behaviours than when it is part of the complete model (e.g. some combinations of inputs are not possible).
Applications: Formal Equivalence Checking (FEC)

- FEC is a technique for formally proving the equivalence of two design models (e.g. RTL “golden” model vs the implementation).

- FEC has to be performed compositionally: separate the models into small parts (slices), and the equivalence between the slices with BDD or SAT-based FEC engine.

- **Note**: any slice in isolation can have more behaviours than when it is part of the complete model (e.g. some combinations of inputs are not possible).

- FEC is performed under the *environmental assumptions* which mimic the essential behaviour of the complete model with respect to the slice.
Applications: Formal Equivalence Checking (FEC)

SAT-based FEC

- Build a CNF formula $\mathcal{F}$ that captures the logic of the two slices and the environmental assumptions.
- Property: $\mathcal{F} \in \text{UNSAT}$ if and only if the slices, under the given assumptions, are functionally equivalent.

But now, if $\mathcal{F} \in \text{UNSAT}$, the assumptions need to be confirmed — the designer must prove that the assumptions are guaranteed by the model (assume-guarantee reasoning).

Thus, it is critical to reduce the number of assumptions. MUSes provide an effective and practically feasible way to reduce the number of assumptions; critical impact on the efficiency of the design flow.
Applications: Formal Equivalence Checking (FEC)

SAT-based FEC

- Build a CNF formula $\mathcal{F}$ that captures the logic of the two slices and the environmental assumptions.
- Property: $\mathcal{F} \in \text{UNSAT}$ if and only if the slices, under the given assumptions, are functionally equivalent.

But now, if $\mathcal{F} \in \text{UNSAT}$, the assumptions need to be confirmed — the designer must prove that the assumptions are guaranteed by the model (assume-guarantee reasoning).
Applications: Formal Equivalence Checking (FEC)

SAT-based FEC

- Build a CNF formula $F$ that captures the logic of the two slices and the environmental assumptions.

- Property: $F \in \text{UNSAT}$ if and only if the slices, under the given assumptions, are functionally equivalent.

But now, if $F \in \text{UNSAT}$, the assumptions need to be confirmed — the designer must prove that the assumptions are guaranteed by the model (assume-guarantee reasoning).

Thus, it is critical to reduce the number of assumptions.
Applications: Formal Equivalence Checking (FEC)

SAT-based FEC

- Build a CNF formula $\mathcal{F}$ that captures the logic of the two slices and the environmental assumptions.

- Property: $\mathcal{F} \in \text{UNSAT}$ if and only if the slices, under the given assumptions, are functionally equivalent.

But now, if $\mathcal{F} \in \text{UNSAT}$, the assumptions need to be confirmed — the designer must prove that the assumptions are guaranteed by the model (assume-guarantee reasoning).

Thus, it is critical to reduce the number of assumptions.

MUSes provide an effective and practically feasible way to reduce the number of assumptions; critical impact on the efficiency of the design flow.
Applications: Proof-based Abstraction Refinement (PBA)

An approach to model checking of large industrial hardware designs.

▶ Run a Bounded Model Checking (BMC) run for some small depth $k$.
▶ i.e. construct a propositional formula $\text{BMC}(k)$ such that it is UNSAT if and only if no execution of the FSM (representing the design) with $\leq k$ steps violates the correctness property.
▶ If $\text{BMC}(k) \in \text{SAT}$ — the property is violated, and we are done.
▶ If $\text{BMC}(k) \in \text{UNSAT}$, use an unsatisfiable core $U \subseteq \text{BMC}(k)$ to construct a localization abstraction of the design:
▶ let $\text{LC}(L, k)$ be the set of clauses in the $\text{BMC}(k)$ that represent the input-output relationship of some latch $L$.
▶ if $U \cap \text{LC}(L, k) = \emptyset$, drop $L$ from the design (i.e. replace it by primary input).

Note: the abstraction has more behaviours, but no bad runs of length $\leq k$. 
Applications: Proof-based Abstraction Refinement (PBA)

An approach to model checking of large industrial hardware designs.

- Run a Bounded Model Checking (BMC) run for some small depth $k$.
  - i.e. construct a propositional formula $BMC(k)$ such that it is UNSAT if and only if no execution of the FSM (representing the design) with $\leq k$ steps violates the correctness property.

Note: the abstraction has more behaviours, but no bad runs of length $\leq k$. 

A. Belov
Applications: Proof-based Abstraction Refinement (PBA)

An approach to model checking of large industrial hardware designs.

- Run a Bounded Model Checking (BMC) run for some small depth \( k \).
  - i.e. construct a propositional formula \( BMC(k) \) such that it is UNSAT if and only if no execution of the FSM (representing the design) with \( \leq k \) steps violates the correctness property.

- If \( BMC(k) \in SAT \) — the property is violated, and we are done.

Note: the abstraction has more behaviours, but no bad runs of length \( \leq k \).
Applications: Proof-based Abstraction Refinement (PBA)

An approach to model checking of large industrial hardware designs.

- Run a Bounded Model Checking (BMC) run for some small depth \( k \).
  - i.e. construct a propositional formula \( BMC(k) \) such that it is UNSAT if and only if no execution of the FSM (representing the design) with \( \leq k \) steps violates the correctness property.

- If \( BMC(k) \in \text{SAT} \) — the property is violated, and we are done.

- If \( BMC(k) \in \text{UNSAT} \), use an unsatisfiable core \( \mathcal{U} \subseteq BMC(k) \) to construct a localization abstraction of the design:
  - let \( LC(L, k) \) be the set of clauses in the \( BMC(k) \) that represent the input-output relationship of some latch \( L \).
  - if \( \mathcal{U} \cap LC(L, k) = \emptyset \), drop \( L \) from the design (i.e. replace it by primary input)
Applications: Proof-based Abstraction Refinement (PBA)

An approach to model checking of large industrial hardware designs.

- Run a Bounded Model Checking (BMC) run for some small depth $k$.
  - i.e. construct a propositional formula \( BMC(k) \) such that it is UNSAT if and only if no execution of the FSM (representing the design) with \( \leq k \) steps violates the correctness property.

- If \( BMC(k) \in \text{SAT} \) — the property is violated, and we are done.

- If \( BMC(k) \in \text{UNSAT} \), use an unsatisfiable core \( \mathcal{U} \subseteq BMC(k) \) to construct a localization abstraction of the design:
  - let \( LC(L, k) \) be the set of clauses in the \( BMC(k) \) that represent the input-output relationship of some latch \( L \).
  - if \( \mathcal{U} \cap LC(L, k) = \emptyset \), drop \( L \) from the design (i.e. replace it by primary input)

**Note:** the abstraction has more behaviours, but no bad runs of length \( \leq k \).
Applications: Proof-based Abstraction Refinement (PBA)

**Note:** Each removed latch reduces the size of the state space by half

**Note:** Abstractions typically contain a small fraction of the latches in the concrete designs ⇒ easier to model-check.
Applications: Proof-based Abstraction Refinement (PBA)

**Note:** Each removed latch reduces the size of the state space by half

**Note:** Abstractions typically contain a small fraction of the latches in the concrete designs ⇒ easier to model-check.

Now, run a complete model checker (e.g. BDD-based, IC3, etc.):

- If the property holds on the abstraction, then it holds in the concrete design. Done.

- If the property is violated, the length of the counterexample can be used for the next BMC run.
Applications: Proof-based Abstraction Refinement (PBA)

**Note:** Each removed latch reduces the size of the state space by half

**Note:** Abstractions typically contain a small fraction of the latches in the concrete designs ⇒ easier to model-check.

Now, run a complete model checker (e.g. BDD-based, IC3, etc.):

- If the property holds on the abstraction, then it holds in the concrete design. Done.
- If the property is violated, the length of the counterexample can be used for the next BMC run.

Extremely beneficial to abstract away as many latches as possible ⇒ smaller cores are better ⇒ MUSes.

**Note:** Recent evaluation of the effectiveness of MUSes in this setting is to appear at DATE-13.
Open Problems

Heuristics

- Current work is very scarce.
- In light of recently developed optimizations, it's not even clear what kind of heuristics is needed (e.g. maybe it's better to go for unnecessary clauses, to take advantage of clause-set refinement).
Open Problems

Heuristics

- Current work is very scarce.
- In light of recently developed optimizations, it's not even clear what kind of heuristics is needed (e.g. maybe it's better to go for unnecessary clauses, to take advantage of clause-set refinement).

Scalability

- New algorithms, new optimization techniques.
Open Problems

Heuristics

- Current work is very scarce.
- In light of recently developed optimizations, it's not even clear what kind of heuristics is needed (e.g. maybe it's better to go for unnecessary clauses, to take advantage of clause-set refinement).

Scalability

- New algorithms, new optimization techniques.

Applications

- New applications; typically pose new types of problems (extensions of the “standard” MUS computation — more on this tomorrow).
Further Reading

Recent surveys on algorithms


Entry point for theory


Advanced techniques

References


