Satisfiability, Boolean Modeling and Computation
Spring 2016

Matti Järvisalo

Lecture 6: Iterative applications of SAT Solvers.
Minimal unsatisfiability, part I.
Feb 4, 2016
Beyond Satisfiability
SAT Solvers: Practical “NP-Oracles”

- “From NP-intractability to NP-tractability”
- Attack problems via *iterative calls* to a SAT solver

### Beyond mere Satisfiability

- **Optimization:** *Maximum satisfiability* (MaxSAT)

- Boolean combinations of linear inequalities, bit-vector arithmetics, ...: *Satisfiability modulo theories* (SMT)

- Higher levels of the polynomial hierarchy: *Counter-example guided abstraction refinement*

- Model counting and enumeration

...
On This Lecture

- Solving problem by iterative calls to a SAT solver
- Incremental SAT
- Minimal unsatisfiability, part I
Applying SAT Solvers Iteratively
**Graph $k$-Coloring**

**$k$–Coloring**

**Input:** Graph $G = (V, E)$, integer $k$ (colors $1, \ldots, k$).

**Solution:** a $k$-coloring of $G$ (if one exists), otherwise “no”

$k$-coloring:
- Every node $v \in V$ assigned exactly one color.
- For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.

SAT solvers provide one of the most efficient approaches to optimally coloring graphs.
**Graph $k$-Coloring**

**$k$–Coloring**

**Input:** Graph $G = (V, E)$, integer $k$ (colors 1, \ldots, $k$).

**Solution:** a $k$-coloring of $G$ (if one exists), otherwise “no”

$k$-coloring:

- Every node $v \in V$ assigned exactly one color.
- For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.

![Graph with 3-coloring](image)

a 3-coloring $(1, 2, 3)$

SAT solvers provide one of the most efficient approaches to optimally coloring graphs.
Graph $k$-Coloring

$k$–Coloring

**Input:** Graph $G = (V, E)$, integer $k$ (colors 1, \ldots, $k$).

**Solution:** a $k$-coloring of $G$ (if one exists), otherwise “no”

$k$-coloring:

- Every node $v \in V$ assigned exactly one color.
- For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.

![Graphs with colorings]

- a 3-coloring $(1, 2, 3)$
- no 2-colorings

SAT solvers provide one of the most efficient approaches to optimally coloring graphs.
Graph $k$-Coloring

$k$-Coloring

**Input:** Graph $G = (V, E)$, integer $k$ (colors $1, \ldots, k$).

**Solution:** a $k$-coloring of $G$ (if one exists), otherwise “no”

$k$-coloring:
- Every node $v \in V$ assigned exactly one color.
- For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.

SAT solvers provide one of the most efficient approaches to optimally coloring graphs.
3-Coloring as SAT

3-Coloring

**Input**: Graph $G = (V, E)$.

**Solution**: a 3-coloring of $G$ (if one exists), otherwise “no”

- **Variables**: $v_i = “\text{node } v \text{ has color } i \in \{r, g, b\}”$

- **Constraints**:
  - Every node $v \in V$ assigned exactly one color. In other words:
    - $v$ assigned at least one color.
    - $v$ assigned at most one color.
  - For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.
3-Coloring as SAT

3-Coloring

**Input:** Graph $G = (V, E)$.

**Solution:** a 3-coloring of $G$ (if one exists), otherwise “no”

A SAT encoding of 3-Coloring for a given graph $G = (V, E)$:

- **Variables:** $v_i =$ “node $v$ has color $i \in \{r, g, b\}$”
- **Constraints:**
  - Every node $v \in V$ assigned exactly one color.
    - In other words:
      - $v$ assigned at least one color.
      - $v$ assigned at most one color.
  - For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.
3-Coloring as SAT

3-Coloring

**Input:** Graph $G = (V, E)$.

**Solution:** a 3-coloring of $G$ (if one exists), otherwise “no”

A SAT encoding of 3-Coloring for a given graph $G = (V, E)$:

- **Variables:** $v_i =$ “node $v$ has color $i \in \{r, g, b\}$”
- **Constraints:**
  - Every node $v \in V$ assigned exactly one color.
    In other words:
    - $v$ assigned at least one color. $v_r \lor v_g \lor v_b$
    - $v$ assigned at most one color.
  - For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.
3-Coloring as SAT

3-Coloring

**Input:** Graph $G = (V, E)$.  
**Solution:** a 3-coloring of $G$ (if one exists), otherwise “no”

A SAT encoding of 3-Coloring for a given graph $G = (V, E)$:

- **Variables:** $v_i = “$node $v$ has color $i \in \{r, g, b\}$”
- **Constraints:**
  - Every node $v \in V$ assigned exactly one color. In other words:
    - $v$ assigned at least one color.
    - $v$ assigned at most one color.
      - (pairwise encoding)
    - For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.
3-Coloring as SAT

3-Coloring

**Input:** Graph $G = (V, E)$.

**Solution:** a 3-coloring of $G$ (if one exists), otherwise “no”

A SAT encoding of 3-Coloring for a given graph $G = (V, E)$:

- **Variables:** $v_i = \text{“node } v \text{ has color } i \in \{r, g, b\}”$
- **Constraints:**
  - Every node $v \in V$ assigned exactly one color.
    - In other words:
      - $v$ assigned at least one color.
      - $v$ assigned at most one color.
        (pairwise encoding)
  - For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.

\[ v_r \lor v_g \lor v_b \]
\[ \neg v_r \lor \neg v_g \]
\[ \neg v_r \lor \neg v_b \]
\[ \neg v_g \lor \neg v_b \]
\[ \neg v_r \lor \neg u_r \]
\[ \neg v_g \lor \neg u_g \]
\[ \neg v_b \lor \neg u_b \]
$k$–Coloring as SAT

$k$–Coloring

**Input:** Graph $G = (V, E)$, integer $k$.

**Solution:** a $k$-coloring of $G$ (if one exist), otherwise “no”

A SAT encoding of $k$-Coloring for a given graph $G = (V, E)$:

- **Variables:** $v_i = \text{“node } v \text{ has color } i \in \{1, \ldots, k\}\text{”}$
- **Constraints:**
  - Every node $v \in V$ assigned exactly one color. In other words:
    - $v$ assigned at least one color.
    - $v$ assigned at most one color.
  - For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.
**$k$–Coloring as SAT**

### $k$–Coloring

**Input:** Graph $G = (V, E)$, integer $k$.

**Solution:** a $k$-coloring of $G$ (if one exist), otherwise “no”

A SAT encoding of $k$-Coloring for a given graph $G = (V, E)$:

- **Variables:** $v_i = \text{“node } v \text{ has color } i \in \{1, \ldots, k\}”$
- **Constraints:**
  - Every node $v \in V$ assigned exactly one color.
    - In other words:
      - $v$ assigned at least one color.
      - $v$ assigned at most one color.
  - For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.
**k–Coloring as SAT**

**k–Coloring**

**Input:** Graph $G = (V, E)$, integer $k$.

**Solution:** a $k$-coloring of $G$ (if one exist), otherwise “no”

A SAT encoding of $k$-Coloring for a given graph $G = (V, E)$:

- **Variables:** $v_i = “\text{node } v \text{ has color } i \in \{1, \ldots, k\}”$
- **Constraints:**
  - Every node $v \in V$ assigned exactly one color. 
    In other words:
    - $v$ assigned at least one color. 
    - $v$ assigned at most one color. 
    - $v_1 \lor \cdots \lor v_k$
    - $\neg v_i \lor \neg v_j \ \forall i \neq j \in \{1, \ldots, k\}$
  - For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.
**k–Coloring as SAT**

**k–Coloring**

**Input:** Graph $G = (V, E)$, integer $k$.

**Solution:** a $k$-coloring of $G$ (if one exist), otherwise “no”

A SAT encoding of $k$-Coloring for a given graph $G = (V, E)$:

- **Variables:** $v_i = “node v has color \ i \in \{1, \ldots, k\}”$
- **Constraints:**
  - Every node $v \in V$ assigned exactly one color. In other words:
    - $v$ assigned at least one color.
      
    $v_1 \lor \cdots \lor v_k$
    - $v$ assigned at most one color.
      
    $\neg v_i \lor \neg v_j \ \forall i \neq j \in \{1, \ldots, k\}$
      
    *(pairwise encoding)*
  - For each edge $(v, u) \in E$, nodes $v$ and $u$ must have different colors.
    
    $\neg v_i \lor \neg u_i \ \forall i \in \{1, \ldots, k\}$
Finding an *Optimal* Graph Coloring with SAT

**Input:** Graph $G$

**Solution:** The least number $k$ such that $G$ has a $k$–coloring.

Initialize the number of colors $k := 1$

$k := k + 1$

A simple, generic scheme for solving various optimization problems

Examples: Bounded model checking / Planning:
“Does there exist an execution / plan of length $k$ which leads to a bad/goal state?”
Finding an *Optimal* Graph Coloring with SAT

- **Input**: Graph $G$
- **Solution**: The least number $k$ such that $G$ has a $k$–coloring.
- Initialize the number of colors $k := 1$

\[
\begin{align*}
   k &:= k + 1 \\
\end{align*}
\]

A simple, generic scheme for solving various optimization problems

- Examples: Bounded model checking / Planning:
  “Does there exist an execution / plan of length $k$ which leads to a bad/goal state?”
Iterating over $k$

- Different ways of iterating over values of $k$.
- Two “standard” approaches:

  **Linear search**
  - Start from $k = 1$.
  - Increment $k$ by 1 until a solution is found.

  **Binary search**
  - Requirement:
    - *knowledge of the maximum possible value of $k$!*
    - For **Graph Coloring**:
      - upper bound: maximum node degree + 1
    - Requirement: *if there is no solution for some $k$, then there is no solution for any $k' < k*
      - In general, this may also not be the case
      - Depends on
        - (i) the problem
        - (ii) the problem encoding
Incremental SAT
Incremental SAT

In many application scenarios, it is beneficial to be able to make several SAT checks on the same input CNF formula under different forced partial assignments.

- Such forced partial assignments are called assumptions
- “Is the formula $F$ satisfiable under the assumption $x = 1$?”

Various modern CDCL SAT solvers implement an API for solving under assumption

- The input formula is read in only once
- The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions
- The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls
- The solver can keep its internal state from the previous solver call to the next
  - Learned clauses
  - Heuristic scores
Incremental SAT

- In many application scenarios, it is beneficial to be able to make several SAT checks on the same input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called assumptions
  - “Is the formula $F$ satisfiable under the assumption $x = 1$?”
- Various modern CDCL SAT solvers implement an API for solving under assumption
  - The input formula is read in only once
  - The user implements an iterative loop that calls the same solver instantiation under different sets of assumptions
  - The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls
  - The solver can keep its internal state from the previous solver call to the next
    - Learned clauses
    - Heuristic scores
Incremental SAT

- In many application scenarios, it is beneficial to be able to make several SAT checks on the same input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called assumptions
  - “Is the formula $F$ satisfiable under the assumption $x = 1$?”
- Various modern CDCL SAT solvers implement an API for solving under assumption
  - The input formula is read in only once
  - The user implements an iterative loop that calls the same solver instantiation under different sets of assumptions
  - The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls
  - The solver can keep its internal state from the previous solver call to the next
    - Learned clauses
    - Heuristic scores
In many application scenarios, it is beneficial to be able to make several SAT checks on the same input CNF formula under different forced partial assignments.

Such forced partial assignments are called assumptions.

"Is the formula $F$ satisfiable under the assumption $x = 1$?"

Various modern CDCL SAT solvers implement an API for solving under assumption

The input formula is read in only once.

The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions.

The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls.

The solver can keep its internal state from the previous solver call to the next.

- Learned clauses
- Heuristic scores
Incremental SAT

- In many application scenarios, it is beneficial to be able to make several SAT checks on the same input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called assumptions
  - “Is the formula $F$ satisfiable under the assumption $x = 1$?”
- Various modern CDCL SAT solvers implement an API for solving under assumption
  - The input formula is read in only once
  - The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions
  - The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls
  - The solver can keep its internal state from the previous solver call to the next
    - Learned clauses
    - Heuristic scores
In many application scenarios, it is beneficial to be able to make several SAT checks on the same input CNF formula under different forced partial assignments.

Such forced partial assignments are called assumptions.

"Is the formula $F$ satisfiable under the assumption $x = 1$?"

Various modern CDCL SAT solvers implement an API for solving under assumption

- The input formula is read in only once
- The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions
- The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls
- The solver can keep its internal state from the previous solver call to the next
  - Learned clauses
  - Heuristic scores
Incremental SAT: Minisat

Minisat

- Perhaps the most used SAT solver
- Very clean and easy-to-understand-and-modify source code
- Offers an incremental interface

- `solve(partial assignment: list of assumptions)`: for making a SAT solver call under a set of assumptions

- `analyzeFinal`: returns an explanation for unsatisfiability under the assumptions as a clause over a subset of the assumptions

- `addClauses`: for adding more clauses between solver calls
Incremental SAT: Explaining Unsatisfiability

Recall:
- CDCL determines unsatisfiability when learning the empty clause
  - By propagating a conflict at decision level 0

Explaining unsatisfiability under assumptions:
- The reason for unsatisfiability can be traced back to assumptions that were necessary for propagating the conflict at level 0.

Essentially:
- Force the assumptions as the first “decisions”
- When one of these decisions results in a conflict: trace the reason of the conflict back to the forced assumptions

This is very useful for many applications
- Heavily used in SAT-based approaches to Boolean optimization
Explaining Unsatisfiability: UNSAT cores

**Unsatisfiable core**
An unsatisfiable subset of the clauses in an unsatisfiable CNF formula

**Simple way of finding an UNSAT core of** $F = \bigwedge_{i=1}^{n} C_i$
- Introduce new *assumption / relaxation* variables $a_i$
- Solve $\bigwedge_{i=1}^{n} (C_i \lor a_i)$ *under the assumptions* $a_i = 0$ for each $i = 1..n$

**Minimal unsatisfiable core (MUS)**
UNSAT core such that each of its subsets is *satisfiable*
- A minimal explanation (in terms of clauses) of unsatisfiability
Explaining Unsatisfiability: UNSAT cores

Unsatisfiable core
An unsatisfiable subset of the clauses in an unsatisfiable CNF formula

Simple way of finding an UNSAT core of $F = \bigwedge_{i=1}^{n} C_i$
- Introduce new *assumption / relaxation* variables $a_i$
- Solve $\bigwedge_{i=1}^{n} (C_i \lor a_i)$ *under the assumptions* $a_i = 0$ for each $i = 1..n$

Minimal unsatisfiable core (MUS)
UNSAT core such that each of its subsets is *satisfiable*
- A minimal explanation (in terms of clauses) of unsatisfiability
Minimal Unsatisfiability
Motivation

Finding (small) explanations for sources of inconsistency is important to repair inconsistency one needs to identify its sources.

Various real-world applications

- error localization/diagnosis and debugging
- model checking
- logic synthesis
- ...

MUS Extraction

Finding minimal subsets of clauses that are unsatisfiable

- Also: basis for core-guided maximum satisfiability (MaxSAT) solvers

Theory

An important criticality problem — $D^P$-completeness
Minimally Unsatisfiable CNF Formulas (MUSes): Example

CNF formula $F = \bigwedge_{i=1}^{6} C_i$, viewed as the set of clauses $\{C_1, \ldots, C_6\}$.

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

Narrowing down sources of unsatisfiability (inconsistency) of $F$

- $\{C_1, C_3, C_4\} \in \text{UNSAT}$
- $\{C_1, C_3, C_4\}$ minimally wrt UNSAT: every subset of $\{C_1, C_3, C_4\}$ is satisfiable.
- $\leadsto \{C_1, C_3, C_4\}$ is a minimally unsatisfiable subformula (MUS) of $F$.

Minimality: not necessarily unique.

$\{C_1, C_2, C_3, C_6\}$ is also an MUS of $F$. 
Minimally Unsatisfiable CNF Formulas (MUSes): Example

CNF formula $F = \bigwedge_{i=1}^{6} C_i$, viewed as the set of clauses $\{C_1, \ldots, C_6\}$.

C1 : (x)
C2 : (¬x ⊕ z)
C3 : (y)
C4 : (¬x ⊕ ¬y)
C5 : (x ⊕ y)
C6 : (¬y ⊕ ¬z)

Narrowing down sources of unsatisfiability (inconsistency) of $F$

- $\{C_1, C_3, C_4\} \in$ UNSAT
- $\{C_1, C_3, C_4\}$ minimally wrt UNSAT:
  every subset of $\{C_1, C_3, C_4\}$ is satisfiable.
  $\sim \Rightarrow \{C_1, C_3, C_4\}$ is a minimally unsatisfiable subformula (MUS) of $F$.
- Minimality: not necessarily unique.
  $\{C_1, C_2, C_3, C_6\}$ is also an MUS of $F$. 
Minimally Unsatisfiable CNF Formulas (MUSes): Example

CNF formula $F = \bigwedge_{i=1}^{6} C_i$, viewed as the set of clauses $\{C_1, \ldots, C_6\}$.

Narrowing down sources of unsatisfiability (inconsistency) of $F$

$\{C_1, C_3, C_4\} \in \text{UNSAT}$

$\{C_1, C_3, C_4\}$ minimally wrt UNSAT: every subset of $\{C_1, C_3, C_4\}$ is satisfiable.

$\Rightarrow \{C_1, C_3, C_4\}$ is a minimally unsatisfiable subformula (MUS) of $F$.

Minimality: not necessarily unique.

$\{C_1, C_2, C_3, C_6\}$ is also an MUS of $F$.

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

CNF formula $F = \bigwedge_{i=1}^{6} C_i$, viewed as the set of clauses $\{C_1, \ldots, C_6\}$.

Cylindrical:
- $C_1 : (x)$
- $C_2 : (\neg x \lor z)$
- $C_3 : (y)$
- $C_4 : (\neg x \lor \neg y)$
- $C_5 : (x \lor y)$
- $C_6 : (\neg y \lor \neg z)$

Narrowing down sources of unsatisfiability (inconsistency) of $F$

- $\{C_1, C_3, C_4\} \in \text{UNSAT}$
- $\{C_1, C_3, C_4\}$ minimally wrt UNSAT:
  - every subset of $\{C_1, C_3, C_4\}$ is satisfiable.
  - $\leadsto \{C_1, C_3, C_4\}$ is a minimally unsatisfiable subformula (MUS) of $F$.

Minimality: not necessarily unique.
- $\{C_1, C_2, C_3, C_6\}$ is also an MUS of $F$. 
Minimally Unsatisfiable CNF Formulas (MUSes): Example

CNF formula $F = \bigwedge_{i=1}^{6} C_i$, viewed as the set of clauses $\{C_1, \ldots, C_6\}$.

Narrowing down sources of unsatisfiability (inconsistency) of $F$

- $\{C_1, C_3, C_4\} \in \text{UNSAT}$
- $\{C_1, C_3, C_4\}$ minimally wrt UNSAT: every subset of $\{C_1, C_3, C_4\}$ is satisfiable.
- $\leadsto \{C_1, C_3, C_4\}$ is a minimally unsatisfiable subformula (MUS) of $F$.

Minimality: not necessarily unique.

$\{C_1, C_2, C_3, C_6\}$ is also an MUS of $F$. 

$C_1 : (x)$
$C_2 : (\neg x \lor z)$
$C_3 : (y)$
$C_4 : (\neg x \lor \neg y)$
$C_5 : (x \lor y)$
$C_6 : (\neg y \lor \neg z)$
Minimal Unsatisfiability

**Minimal unsatisfiability**

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

**Minimal unsatisfiable subset (MUS)**

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

\( \text{MUS}(F) \): the set of MUSes of a CNF formula \( F \).
The Complexity Class $D^P$: Characterizes \textit{criticality} problems

A language $L$ is in $D^P$ if there are two languages $L_1$ and $L_2$ such that

- $L_1 \in \text{NP}$,
- $L_2 \in \text{coNP}$, and
- $L = L_1 \cap L_2$.

Canonical $D^P$–complete problem: SAT-UNSAT

\[
\text{SAT-UNSAT} = \{(L_1, L_2) \mid L_1 \in \text{SAT and } L_2 \in \text{UNSAT}\}.
\]

In theory:

- two SAT solver calls are needed to decide whether a given pair $(L_1, L_2) \in \text{SAT-UNSAT}$.
MUS Extraction: Complexity

MUS extraction: finding a critically unsatisfiable subset of clauses

Decision problem: “Is a given CNF formula MU?” is $D^P$-complete

MUS extraction as a functional (search) problem: in $FP^{NP}$

- Polynomial number of SAT solver calls suffices
- Practical MUS extraction algorithms use SAT solvers iteratively
Understanding UNSAT: Hitting Set Duality
Minimal Correction Subsets (MCSes)

\( M \subseteq F \) is a \textit{minimal correction subset} of \( F \) if

- \( F \setminus M \in \text{SAT} \), and
- \( F \setminus (M \setminus \{C\}) \in \text{UNSAT} \) for all \( C \in M \).

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

Example

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

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

\( F \setminus \{C_1\} \text{ SAT} \leadsto \{C_1\} \in \text{MCS}(F) \).

\( F \setminus \{C_2, C_4\} \text{ SAT}, \)
\( F \setminus \{C_2\}, F \setminus \{C_4\} \text{ UNSAT} \)
\( \leadsto \{C_2, C_4\} \in \text{MCS}(F) \).

\( \{C_2, C_3\} \notin \text{MCS}(F) \)
although \( F \setminus \{C_2, C_3\} \text{ SAT} \),
because \( F \setminus \{C_3\} \text{ SAT} \!\)
Minimal Correction Subsets (MCSes)

$M \subseteq F$ is a minimal correction subset of $F$ if
- $F \setminus M \in \text{SAT}$, and
- $F \setminus (M \setminus \{C\}) \in \text{UNSAT}$ for all $C \in M$.

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

Example

$M \subseteq F$ is a minimal correction subset of $F$ if
- $F \setminus M \in \text{SAT}$, and
- $F \setminus (M \setminus \{C\}) \in \text{UNSAT}$ for all $C \in M$.

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

Example

$C_1: (x)$
$C_2: (\neg x \vee z)$
$C_3: (y)$
$C_4: (\neg x \vee \neg y)$
$C_5: (x \vee y)$
$C_6: (\neg y \vee \neg z)$

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

- $F \setminus \{C_1\} \in \text{SAT} \Rightarrow \{C_1\} \in \text{MCS}(F)$.
- $F \setminus \{C_2, C_4\} \in \text{SAT}$,
  $F \setminus \{C_2\}, F \setminus \{C_4\} \in \text{UNSAT}$
  $\Rightarrow \{C_2, C_4\} \in \text{MCS}(F)$.
- $\{C_2, C_3\} \not\in \text{MCS}(F)$
  although $F \setminus \{C_2, C_3\} \in \text{SAT}$,
  because $F \setminus \{C_3\} \in \text{SAT}!$
Minimal Correction Subsets (MCSes)

$M \subseteq F$ is a minimal correction subset of $F$ if

- $F \setminus M \in \text{SAT}$, and
- $F \setminus (M \setminus \{C\}) \in \text{UNSAT}$ for all $C \in M$.

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

Example

- $\text{MUS}(F) = \{\{C_1, C_3, C_4\}, \{C_1, C_2, C_3, C_6\}\}$
- $F \setminus \{C_1\} \in \text{SAT} \implies \{C_1\} \in \text{MCS}(F)$.
- $F \setminus \{C_2, C_4\} \in \text{SAT}$, $F \setminus \{C_2\}, F \setminus \{C_4\} \in \text{UNSAT}
  \implies \{C_2, C_4\} \in \text{MCS}(F)$.
- $\{C_2, C_3\} \notin \text{MCS}(F)$
  although $F \setminus \{C_2, C_3\} \in \text{SAT}$,
  because $F \setminus \{C_3\} \in \text{SAT}$!
**Minimal Correction Subsets (MCSes)**

$M \subseteq F$ is a *minimal correction subset* of $F$ if

- $F \setminus M \in \text{SAT}$, and
- $F \setminus (M \setminus \{C\}) \in \text{UNSAT}$ for all $C \in M$.

**MCS($F$):** the set of all MCSes of $F$.

**Example**

- $M \subseteq F$ is a minimal correction subset of $F$ if
  - $F \setminus M \in \text{SAT}$, and
  - $F \setminus (M \setminus \{C\}) \in \text{UNSAT}$ for all $C \in M$.

- $MCS(F)$: the set of all MCSes of $F$.

- **Example**
  - $C_1: (x)$
  - $C_2: (\neg x \lor z)$
  - $C_3: (y)$
  - $C_4: (\neg x \lor \neg y)$
  - $C_5: (x \lor y)$
  - $C_6: (\neg y \lor \neg z)$

- **MUS($F$) =** $\{\{C_1, C_3, C_4\}, \{C_1, C_2, C_3, C_6\}\}$

- $F \setminus \{C_1\} \text{ SAT} \not\implies \{C_1\} \in \text{MCS}(F)$.

- $F \setminus \{C_2, C_4\} \text{ SAT}$,
  - $F \setminus \{C_2\}, F \setminus \{C_4\} \text{ UNSAT}$
  - $\not\implies \{C_2, C_4\} \in \text{MCS}(F)$.

- $\{C_2, C_3\} \not\in \text{MCS}(F)$ although $F \setminus \{C_2, C_3\} \text{ SAT}$,
  - because $F \setminus \{C_3\} \text{ SAT}$!
Minimal Correction Subsets (MCSes)

$M \subseteq F$ is a \textit{minimal correction subset} of $F$ if

\begin{itemize}
  \item $F \setminus M \in \text{SAT}$, and
  \item $F \setminus (M \setminus \{C\}) \in \text{UNSAT}$ for all $C \in M$.
\end{itemize}

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

\textbf{Example}

\begin{itemize}
  \item $C_1 : (x)$
  \item $C_2 : (\neg x \lor z)$
  \item $C_3 : (y)$
  \item $C_4 : (\neg x \lor \neg y)$
  \item $C_5 : (x \lor y)$
  \item $C_6 : (\neg y \lor \neg z)$
  \item $\text{MUS}(F) = \{\{C_1, C_3, C_4\}, \{C_1, C_2, C_3, C_6\}\}$
  \item $F \setminus \{C_1\} \in \text{SAT} \leadsto \{C_1\} \in \text{MCS}(F)$.
  \item $F \setminus \{C_2, C_4\} \in \text{SAT}$,
    $\quad F \setminus \{C_2\}, F \setminus \{C_4\} \in \text{UNSAT}$
    $\leadsto \{C_2, C_4\} \in \text{MCS}(F)$.
  \item $\{C_2, C_3\} \notin \text{MCS}(F)$
    although $F \setminus \{C_2, C_3\} \in \text{SAT}$,
    because $F \setminus \{C_3\} \in \text{SAT}$!
\end{itemize}
Hitting Set Duality:
Connection between MUSes and MCSes

**Hitting Sets**

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

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

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

- Subset-minimal hitting sets

**Example**

Consider sets $A = \{a, b\}$, $B = \{c, d\}$, $C = \{d, e, f\}$.

- Hitting set of $\{A, B, C\}$: $\{b, d, f\}$ — not irreducible
- Hitting set of $\{A, B, C\}$: $\{a, d\}$ — irreducible. Also e.g. $\{b, c, e\}$. 

SAT (Lecture 6)  
Spring 2016  
May 20 24 / 32
Hitting Set Duality:  
Connection between MUSes and MCSes

**Hitting Sets**

Let $S$ be a collection of sets.

A set $H$ is a hitting set of $S$ is $H \cap S \neq \emptyset$ for all $S \in S$.

A hitting set $H$ is irreducible if no $H' \subset H$ is a hitting set of $S$.

- Subset-minimal hitting sets

**Example**

Consider sets $A = \{a, b\}$, $B = \{c, d\}$, $C = \{d, e, f\}$.

- Hitting set of $\{A, B, C\}$: $\{b, d, f\}$ — not irreducible
- Hitting set of $\{A, B, C\}$: $\{a, d\}$ — irreducible. Also e.g. $\{b, c, e\}$. 

SAT (Lecture 6)  
Spring 2016  
May 20  
24 / 32
Hitting Set Duality: Connection between MUSes and MCSes

Hitting Sets

Let \( S \) be a collection of sets.

A set \( H \) is a hitting set of \( S \) if \( H \cap S \neq \emptyset \) for all \( S \in S \).

A hitting set \( H \) is irreducible if no \( H' \subset H \) is a hitting set of \( S \).

- Subset-minimal hitting sets

Example

Consider sets \( A = \{a, b\} \), \( B = \{c, d\} \), \( C = \{d, e, f\} \).

- Hitting set of \( \{A, B, C\} \): \( \{b, d, f\} \) — not irreducible
- Hitting set of \( \{A, B, C\} \): \( \{a, d\} \) — irreducible. Also e.g. \( \{b, c, e\} \).
Hitting Set Duality

- In order restore consistency of an unsatisfiable CNF formula $F$: must remove clauses from $F$ so that all MUSes disappear.
- Correction subsets of $F$ are hitting sets of MUS($F$).
- Each MCS of $F$ is an irreducible hitting set of MUS($F$).

Hitting Set Duality

For any UNSAT CNF formula $F$:

- $M \in \text{MCS}(F)$ iff $M$ is an irreducible hitting set of MUS($F$).
- $M \in \text{MUS}(F)$ iff $M$ is an irreducible hitting set of MCS($F$).
Hitting Set Duality: Example

Example:

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

- \( \text{MUS}(F) = \{\{C_1, C_3, C_4\}, \{C_1, C_2, C_3, C_6\}\} \)
- \( \text{MCS}(F) = \{\{C_1\}, \{C_3\}, \{C_2, C_4\}, \{C_4, C_6\}\} \)
Further Problems over MUSes

**MUS membership**

Input: a CNF formula $F$ and clause $C \in F$.
Question: Does $C$ belong to some MUS of $F$?
- $\Sigma^P_2$–complete problem
- Related functional problem: Given $F$, compute $\bigcup \text{MUS}(F)$.

**Smallest MUS**

Input: a CNF formula $F$.
Task: compute a smallest-cardinality MUS of $F$.
- Related $\Sigma^P_2$–complete decision problem:
  Given $F$ and an integer $k$, is there a $M \in \text{MUS}(F)$ with $|M| \leq k$

**MUS enumeration**

Given $F$, compute $\text{MUS}(F)$. 

---

**SAT (Lecture 6)**

Spring 2016

May 20 27 / 32
MUSes for High-Level Constraints

How to identify \textit{minimally inconsistent subsets} of higher level constraints $C_1, \ldots, C_m$?

- For example: inconsistency of CSPs — finite-domain constraints

Is it enough to:

1. encode each $C_i$ as a CNF formula $\text{CNF}(C_i)$,
2. apply an MUS extractor on the resulting CNF formula $\bigwedge_{i=1}^{m} \text{CNF}(C_i)$ to obtain an MUS $M$, and
3. map $M$ back to the high-level constraints by \textit{including} $C_i$ \textit{iff} at least one clause in $\text{CNF}(C_i)$ is in $M$.

No!
MUSes for High-Level Constraints

How to identify *minimally inconsistent subsets* of higher level constraints \( C_1, \ldots, C_m \)?

- For example: inconsistency of CSPs — finite-domain constraints

Is it enough to:

1. encode each \( C_i \) as a CNF formula \( \text{CNF}(C_i) \),
2. apply an MUS extractor on the resulting CNF formula \( \bigwedge_{i=1}^{m} \text{CNF}(C_i) \) to obtain an MUS \( M \), and
3. map \( M \) back to the high-level constraints by *including* \( C_i \) iff at least one clause in \( \text{CNF}(C_i) \) is in \( M \).

*No!*
MUSes for High-Level Constraints

How to identify *minimally inconsistent subsets* of higher level constraints $C_1, \ldots, C_m$?

- For example: inconsistency of CSPs — finite-domain constraints

Is it enough to:

1. encode each $C_i$ as a CNF formula $\text{CNF}(C_i)$,
2. apply an MUS extractor on the resulting CNF formula $\bigwedge_{i=1}^{m} \text{CNF}(C_i)$ to obtain an MUS $M$, and
3. map $M$ back to the high-level constraints by *including* $C_i$ iff at least one clause in $\text{CNF}(C_i)$ is in $M$.

No!

Why?
MUSes for High-Level Constraints

How to identify \textit{minimally inconsistent subsets} of higher level constraints $C_1, \ldots, C_m$?

- For example: inconsistency of CSPs — finite-domain constraints

Is it enough to:

1. encode each $C_i$ as a CNF formula $\text{CNF}(C_i)$,
2. apply an MUS extractor on the resulting CNF formula $\bigwedge_{i=1}^m \text{CNF}(C_i)$ to obtain an MUS $M$, and
3. map $M$ back to the high-level constraints by \textit{including} $C_i$ iff at least one clause in $\text{CNF}(C_i)$ is in $M$.

\textit{No!}

The mapping in step 3 does not guarantee to give a \textit{minimally} inconsistent subset of constraints!
Group MUSes give a natural way of finding minimally inconsistent subsets of constraints using MUS extractors for CNF formulas.

Basic Idea

Assume a finite-domain constraint $\mathcal{C}$.

Let $\text{CNF}(\mathcal{C}) = \bigwedge_{i=1}^{m} C_i$ be a CNF encoding of $\mathcal{C}$.

For MUS extraction:

- View $\bigwedge_{i=1}^{m}$ as a clause-group $\{C_1, \ldots, C_m\}$.
- Define MUS extraction over clause-groups.
Group MUSes

Example

\[ G_1 = \begin{cases} 
  C_1 : (x) \\
  C_2 : (y) 
\end{cases} \]
\[ G_2 = \begin{cases} 
  C_3 : (\neg x \lor \neg y) \\
  C_4 : (x \lor y) 
\end{cases} \]
\[ G_3 = \begin{cases} 
  C_5 : (\neg x \lor z) \\
  C_6 : (\neg y \lor \neg z) 
\end{cases} \]

- \( F = G_1 \cup G_2 \cup G_3 \), where \( G_i \)'s are clause-groups:
  \( G_1 = \{ C_1, C_2 \} \), \( G_2 = \{ C_3, C_4 \} \), \( G_3 = \{ C_5, C_6 \} \).
  - \( \{ G_1, G_2 \} \) is a group-MUS of \( F \):
    \( G_1 \cup G_2 \in UNSAT \)
    \( G_1, G_2 \in SAT \).
  - \( \{ G_1, G_3 \} \) is a group-MUS, too.
**Group MUSes**

### Example

\( G_1 = \{ C_1 : (x), C_2 : (y) \} \)

\( G_2 = \{ C_3 : (\neg x \lor \neg y), C_4 : (x \lor y) \} \)

\( G_3 = \{ C_5 : (\neg x \lor z), C_6 : (\neg y \lor \neg z) \} \)

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

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

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

Example

\[ G_1 = \{ C_1 : (x), C_2 : (y) \} \]
\[ G_2 = \{ C_3 : (\neg x \lor \neg y), C_4 : (x \lor y) \} \]
\[ G_3 = \{ C_5 : (\neg x \lor z), C_6 : (\neg y \lor \neg z) \} \]

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

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

\[ \{G_1, G_3\} \text{ is a group-MUS, too.} \]
Group MUSes

Definition

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

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

Example: a group = CNF representation of a gate in a circuit.

Note the special role of \( G_0 \): the “background” clauses.

Example: output constraint in circuit.

\( G_0 \in \text{UNSAT} \iff \) unique GMUS \( \emptyset \).

MUS extraction: a special case of group-MUS extraction

Group-MUS extraction can be reduced to MUS extraction
Summary

Take-home message

- Iterative use of SAT solvers allows for solving complex problems
- Incremental SAT: using CDCL SAT solvers incrementally via the assumption interface
- MUS extraction is an important problem both in theory and practice

Study goals

- Basics of iterative & incremental use of SAT solvers
- MUS extraction: basic definitions and complexity
- Hitting set duality
- Group MUSes

Next time:
Algorithms for MUS extraction