A Survey of Theory Solvers

Emina Torlak
emina@cs.washington.edu
Today
Today

Last lecture

• Introduction to Satisfiability Modulo Theories (SMT)
Today

Last lecture

• Introduction to Satisfiability Modulo Theories (SMT)

Today

• A quick survey of theory solvers
• An in-depth look at the core theory solver (Theory of Equality)
Today

Last lecture
  • Introduction to Satisfiability Modulo Theories (SMT)

Today
  • A quick survey of theory solvers
  • An in-depth look at the core theory solver (Theory of Equality)

Reminder
  • Homework 1 due today at 11pm
  • Homework 2 coming out
  • Email us your project topic and brief abstract by 11pm on Thursday
Satisfiability Modulo Theories (SMT)

\[ x = g(y) \]
\[ 2x + y > 5 \]
\[ (b \gg 2) = c \]
\[ \vdots \]
\[ a[i] = x \]
Satisfiability Modulo Theories (SMT)

x = g(y)
2x + y > 5
(b >> 2) = c
⋮
a[i] = x

Theories

First-Order Logic

SMT solver

(un)satisfiable
Satisfiability Modulo Theories (SMT)

\[ x = g(y) \]
\[ 2x + y > 5 \]
\[ (b >> 2) = c \]
\[ \vdots \]
\[ a[i] = x \]

Theories

First-Order Logic

SMT solver

Core solver

DPLL(T)

Theory solver

\( \text{(un)satisfiable} \)
A brief survey of common theory solvers

- \( x = g(y) \)
- Core solver

- \( 2x + y > 5 \)
  - Theory solver

- \( 2i + j > 5 \)
  - Theory solver

- \( (b >> 2) = c \)
  - Theory solver

- \( a[i] = x \)
  - Theory solver
A brief survey of common theory solvers

\[ x = g(y) \]

Equality and UF

<table>
<thead>
<tr>
<th>Expression</th>
<th>Theory Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 2x + y &gt; 5 )</td>
<td>Linear Real Arithmetic</td>
</tr>
<tr>
<td>( 2i + j &gt; 5 )</td>
<td>Linear Integer Arithmetic</td>
</tr>
<tr>
<td>( (b &gt;&gt; 2) = c )</td>
<td>Fixed-Width Bitvectors</td>
</tr>
<tr>
<td>( a[i] = x )</td>
<td>Arrays</td>
</tr>
</tbody>
</table>
**A brief survey of common theory solvers**

\[ x = g(y) \]

**Equality and UF**

\[ 2x + y > 5 \]

- **Linear Real Arithmetic**

\[ 2i + j > 5 \]

- **Linear Integer Arithmetic**

\[ (b >> 2) = c \]

- **Fixed-Width Bitvectors**

\[ a[i] = x \]

- **Arrays**

- **Conjunctions** of linear constraints over \( \mathbb{R} \)

  - Can be decided in polynomial time, but in practice solved with the **General Simplex** method (worst case exponential)

  - Can also be decided with **Fourier-Motzkin** elimination (exponential)
A brief survey of common theory solvers

\[ x = g(y) \]

Equality and UF

- Conjunctions of linear constraints over \( \mathbb{Z} \)
- Branch-and-bound (based on Simplex)
- Omega Test (extension of Fourier-Motzkin)
- Small-Domain Encoding used for arbitrary combinations of linear constraints over \( \mathbb{Z} \)
- NP-complete

2x + y > 5
Linear Real Arithmetic

2i + j > 5
Linear Integer Arithmetic

(b >> 2) = c
Fixed-Width Bitvectors

a[i] = x
Arrays
A brief survey of common theory solvers

- $x = g(y)$
  - Equality and UF

- $2x + y > 5$
  - Linear Real Arithmetic

- $2i + j > 5$
  - Linear Integer Arithmetic

- $(b >> 2) = c$
  - Fixed-Width Bitvectors

- $a[i] = x$
  - Arrays

- Arbitrary combination of constraints over bitvectors
- Bit blasting (reduction to SAT)
- NP-complete
A brief survey of common theory solvers

\[ x = g(y) \]

Equality and UF

- \( 2x + y > 5 \)  
  - Linear Real Arithmetic
- \( 2i + j > 5 \)  
  - Linear Integer Arithmetic
- \((b >> 2) = c\)  
  - Fixed-Width Bitvectors
- \( a[i] = x \)  
  - Arrays

- Conjunctions of constraints over read/write terms in the theory of arrays
- Reduce to \( T= \) satisfiability
- NP-complete (because the reduction introduces disjunctions)
A brief survey of common theory solvers

<table>
<thead>
<tr>
<th>Expression</th>
<th>Theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2x + y &gt; 5$</td>
<td>Linear Real Arithmetic</td>
</tr>
<tr>
<td>$2i + j &gt; 5$</td>
<td>Linear Integer Arithmetic</td>
</tr>
<tr>
<td>$(b &gt;&gt; 2) = c$</td>
<td>Fixed-Width Bitvectors</td>
</tr>
<tr>
<td>$a[i] = x$</td>
<td>Arrays</td>
</tr>
</tbody>
</table>

- $x = g(y)$
- Equality and UF
- Conjunctions of equality constraints over uninterpreted functions
- Congruence closure
- Polynomial time
Theory of equality and UF ($T\equiv$)

Signature (all symbols)
- $\{=, a, b, c, \ldots, f, g, \ldots, p, q, \ldots\}$

Axioms
- reflexivity: $\forall x. \ x = x$
- symmetry: $\forall x, y. \ x = y \rightarrow y = x$
- transitivity: $\forall x, y, z. \ x = y \land y = z \rightarrow x = z$
- congruence: $\forall x_1, \ldots, x_n, y_1, \ldots, y_n. \ (\land_{1 \leq i \leq n} x_i = y_i) \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$
- congruence: $\forall x_1, \ldots, x_n, y_1, \ldots, y_n. \ (\land_{1 \leq i \leq n} x_i = y_i) \rightarrow p(x_1, \ldots, x_n) \leftrightarrow p(y_1, \ldots, y_n)$
Theory of equality and UF ($T\equiv$)

Signature (all symbols)
• $\{=, a, b, c, \ldots, f, g, \ldots, p, q, \ldots\}$

Axioms
• reflexivity: $\forall x. \ x = x$
• symmetry: $\forall x, y. \ x = y \rightarrow y = x$
• transitivity: $\forall x, y, z. \ x = y \land y = z \rightarrow x = z$
• congruence: $\forall x_1, \ldots, x_n, y_1, \ldots, y_n. \ (\land_{1 \leq i \leq n} x_i = y_i) \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Replace predicates with equality constraints over functions:
• introduce a fresh constant $t$
• for each predicate $p$, introduce a fresh function $f_p$
• $p(x_1, \ldots, x_n) \iff f_p(x_1, \ldots, x_n) = t$

✗ confluence: $\forall x_1, \ldots, x_n, y_1, \ldots, y_n. \ (\land_{1 \leq i \leq n} x_i = y_i) \rightarrow p(x_1, \ldots, x_n) \leftrightarrow p(y_1, \ldots, y_n)$
Theory of equality and UF ($T=\equiv$)

Signature (all function symbols)
- $\{=, a, b, c, \ldots, f, g, \ldots\}$

Axioms
- reflexivity: $\forall x. \; x = x$
- symmetry: $\forall x, y. \; x = y \rightarrow y = x$
- transitivity: $\forall x, y, z. \; x = y \land y = z \rightarrow x = z$
- congruence: $\forall x, y. \; \bigwedge (x_i = y_i) \rightarrow f(x) = f(y)$
Theory of equality and UF ($T=\equiv$)

Signature (all function symbols)

- $\{=, a, b, c, \ldots, f, g, \ldots\}$

Axioms

- Reflexivity: $\forall x.\ x = x$
- Symmetry: $\forall x, y.\ x = y \rightarrow y = x$
- Transitivity: $\forall x, y, z.\ x = y \land y = z \rightarrow x = z$
- Congruence: $\forall x, \overline{y}.\ \bigwedge (x_i = y_i) \rightarrow f(x) = f(\overline{y})$

$T=\equiv$ models

- All structures $\langle U, I \rangle$ that satisfy the axioms of $T=\equiv$
Theory of equality and UF (T=)

Signature (all function symbols)
- \{=, a, b, c, \ldots, f, g, \ldots\}

Axioms
- Reflexivity: \( \forall x. \ x = x \)
- Symmetry: \( \forall x, y. \ x = y \rightarrow y = x \)
- Transitivity: \( \forall x, y, z. \ x = y \land y = z \rightarrow x = z \)
- Congruence: \( \forall x, y. \ \bigwedge (x_i = y_i) \rightarrow f(x) = f(y) \)

T= models
- all structures \( \langle U, I \rangle \) that satisfy the axioms of T=
Is a conjunction of $T = \text{ literals}$ satisfiable?

$$f(f(f(a))) = a \land f(f(f(f(f(a)))))) = a \land f(a) \neq a$$
Is a conjunction of \( T= \) literals satisfiable?

\[ f^3(a) = a \land f^5(a) = a \land f(a) \neq a \]
Congruence closure algorithm

\[ f^3(a) = a \land f^5(a) = a \land f(a) \neq a \]
Congruence closure algorithm

- Place each subterm of F into its own congruence class

\[ f^3(a) = a \land f^5(a) = a \land f(a) \neq a \]
**Congruence closure algorithm**

- Place each subterm of F into its own congruence class
- For each positive literal $t_1 = t_2$ in F

\[ f^3(a) = a \land f^5(a) = a \land f(a) \neq a \]
Congruence closure algorithm

• Place each subterm of $F$ into its own congruence class
• For each positive literal $t_1 = t_2$ in $F$
  • Merge the classes for $t_1$ and $t_2$

$f^3(a) = a \land f^5(a) = a \land f(a) \neq a$
Congruence closure algorithm

- Place each subterm of F into its own congruence class
- For each positive literal $t_1 = t_2$ in F
  - Merge the classes for $t_1$ and $t_2$
  - Propagate the resulting congruences

$$f^3(a) = a \land f^5(a) = a \land f(a) \neq a$$
**Congruence closure algorithm**

- Place each subterm of F into its own congruence class
- For each positive literal $t_1 = t_2$ in F
  - Merge the classes for $t_1$ and $t_2$
  - Propagate the resulting congruences

$\text{f}^3(a) = a \land \text{f}^5(a) = a \land \text{f}(a) \neq a$
• Place each subterm of F into its own congruence class
• For each positive literal $t_1 = t_2$ in F
  • Merge the classes for $t_1$ and $t_2$
  • Propagate the resulting congruences
Congruence closure algorithm

- Place each subterm of F into its own congruence class
- For each positive literal \( t_1 = t_2 \) in F
  - Merge the classes for \( t_1 \) and \( t_2 \)
  - Propagate the resulting congruences

\[
\begin{align*}
\text{Congruence closure algorithm} \\
\text{Place each subterm of F into its own congruence class} \\
\text{For each positive literal } t_1 = t_2 \text{ in F} \\
\quad \text{Merge the classes for } t_1 \text{ and } t_2 \\
\quad \text{Propagate the resulting congruences}
\end{align*}
\]
• Place each subterm of F into its own congruence class
• For each positive literal \( t_1 = t_2 \) in F
  • Merge the classes for \( t_1 \) and \( t_2 \)
  • Propagate the resulting congruences

Congruence closure algorithm

\[
f^3(a) = a \land f^5(a) = a \land f(a) \neq a
\]
Congruence closure algorithm

- Place each subterm of \( F \) into its own congruence class
- For each positive literal \( t_1 = t_2 \) in \( F \)
  - Merge the classes for \( t_1 \) and \( t_2 \)
  - Propagate the resulting congruences

\[
f^3(a) = a \land f^5(a) = a \land f(a) \neq a
\]
Congruence closure algorithm

- Place each subterm of F into its own congruence class
- For each positive literal $t_1 = t_2$ in F
  - Merge the classes for $t_1$ and $t_2$
  - Propagate the resulting congruences
- If F has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT
Congruence closure algorithm

- Place each subterm of F into its own congruence class
- For each positive literal $t_1 = t_2$ in F
  - Merge the classes for $t_1$ and $t_2$
  - Propagate the resulting congruences
- If F has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT
Congruence closure algorithm: another example

- Place each subterm of $F$ into its own congruence class
- For each positive literal $t_1 = t_2$ in $F$
  - Merge the classes for $t_1$ and $t_2$
  - Propagate the resulting congruences
- If $F$ has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT

$$f(x) = f(y) \land x \neq y$$
• Place each subterm of $F$ into its own congruence class
• For each positive literal $t_1 = t_2$ in $F$
  • Merge the classes for $t_1$ and $t_2$
  • Propagate the resulting congruences
• If $F$ has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
• Otherwise, output SAT
Congruence closure algorithm: another example

• Place each subterm of $F$ into its own congruence class
• For each positive literal $t_1 = t_2$ in $F$
  • Merge the classes for $t_1$ and $t_2$
  • Propagate the resulting congruences
• If $F$ has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
• Otherwise, output SAT
Congruence closure algorithm: another example

- Place each subterm of $F$ into its own congruence class
- For each positive literal $t_1 = t_2$ in $F$
  - Merge the classes for $t_1$ and $t_2$
  - Propagate the resulting congruences
- If $F$ has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT

\[ f(x) = f(y) \land x \neq y \]
**Congruence closure algorithm: another example**

- Place each subterm of F into its own congruence class
- For each positive literal $t_1 = t_2$ in F
  - Merge the classes for $t_1$ and $t_2$
  - Propagate the resulting congruences
- If F has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT

\[ f(x) = f(y) \land x \neq y \]
A binary relation $R$ is an **equivalence relation** if it is reflexive, symmetric, and transitive.

An equivalence relation $R$ is a **congruence relation** iff $\forall x, y. \bigwedge R(x_i, y_i) \rightarrow R(f(x), f(y))$.

The **equivalence class** of an element $s \in S$ under an equivalence relation $R$: 

$$\{ s' \in S | R(s, s') \}$$

An equivalence class is called a **congruence class** if $R$ is a congruence relation.
A binary relation $R$ is an **equivalence relation** if it is reflexive, symmetric, and transitive.

An equivalence relation $R$ is a **congruence relation** iff

$$\forall x, y. \land R(x_i, y_i) \rightarrow R(f(x), f(y))$$
A binary relation $R$ is an equivalence relation if it is reflexive, symmetric, and transitive.

An equivalence relation $R$ is a congruence relation iff

$$\forall x, y. \land R(x_i, y_i) \rightarrow R(f(x), f(y))$$

The equivalence class of an element $s \in S$ under an equivalence relation $R$:

$$\{ s' \in S \mid R(s, s') \}$$

What is the equivalence class of 9 under $\equiv_3$?
A binary relation $R$ is an equivalence relation if it is reflexive, symmetric, and transitive.

An equivalence relation $R$ is a congruence relation iff

$$\forall x, y. \bigwedge R(x_i, y_i) \rightarrow R(f(x), f(y))$$

The equivalence class of an element $s \in S$ under an equivalence relation $R$:

$$\{ s' \in S | R(s, s') \}$$

An equivalence class is called a congruence class if $R$ is a congruence relation.
The equivalence closure $R^E$ of a binary relation $R$ is the smallest equivalence relation that contains $R$. 
The equivalence closure $R^E$ of a binary relation $R$ is the smallest equivalence relation that contains $R$.

What is the equivalence closure of $R = \{\langle a, b \rangle, \langle b, c \rangle, \langle d, d \rangle\}$?
The equivalence closure $R^E$ of a binary relation $R$ is the smallest equivalence relation that contains $R$.

What is the equivalence closure of $R = \{\langle a, b \rangle, \langle b, c \rangle, \langle d, d \rangle\}$?

$R^E = \{\langle a, a \rangle, \langle b, b \rangle, \langle c, c \rangle, \langle d, d \rangle, \langle a, b \rangle, \langle b, a \rangle, \langle b, c \rangle, \langle c, b \rangle, \langle a, c \rangle, \langle c, a \rangle\}$
The congruence closure algorithm computes the congruence closure of the equality relation over terms asserted by a conjunctive quantifier-free formula in $T_\equiv$.

The equivalence closure $R^E$ of a binary relation $R$ is the smallest equivalence relation that contains $R$.

The congruence closure $R^C$ of a binary relation $R$ is the smallest congruence relation that contains $R$.
Congruence closure algorithm: data structure

\[ f(a, b) = a \land f(f(a, b), b) \neq a \]
Congruence closure algorithm: data structure

- Represent subterms with a DAG

\[ f(a, b) = a \land f(f(a, b), b) \neq a \]
Congruence closure algorithm: data structure

- Represent subterms with a DAG
- Each node has a find pointer to another node in its congruence class (or to itself if it is the representative)

\[ f(a, b) = a \land f(f(a, b), b) \neq a \]
Congruence closure algorithm: data structure

- Represent subterms with a DAG
- Each node has a **find** pointer to another node in its congruence class (or to itself if it is the **representative**)
- Each representative has a **ccp** field that stores all parents of all nodes in its congruence class.

\[
f(a, b) = a \land f(f(a, b), b) \neq a
\]
Congruence closure algorithm: union-find

\[ f(a, b) = a \land f(f(a, b), b) \neq a \]
Congruence closure algorithm: union-find

- \text{FIND} returns the representative of a node's equivalence class.

\[
f(a, b) = a \land f(f(a, b), b) \neq a
\]
Congruence closure algorithm: union-find

- **FIND** returns the representative of a node’s equivalence class.
- **UNION** combines equivalence classes for nodes $i_1$ and $i_2$:
  - $n_1, n_2 \leftarrow \text{FIND}(i_1), \text{FIND}(i_2)$
  - $n_1.\text{find} \leftarrow n_2$
  - $n_2.\text{ccp} \leftarrow n_1.\text{ccp} \cup n_2.\text{ccp}$
  - $n_1.\text{ccp} \leftarrow \emptyset$

Diagram:

- $f(a, b) = a \land f(f(a, b), b) \neq a$

**Diagram:**

- Node 1 (f)
  - $\{1, 2\}$
- Node 2 (f)
  - $\emptyset$
- Node 3 (a)
  - $\{1, 2\}$
- Node 4 (b)
  - $\{1, 2\}$
• **FIND** returns the representative of a node’s equivalence class.

• **UNION** combines equivalence classes for nodes $i_1$ and $i_2$:
  - $n_1, n_2 \leftarrow \text{FIND}(i_1), \text{FIND}(i_2)$
  - $n_1.\text{find} \leftarrow n_2$
  - $n_2.\text{ccp} \leftarrow n_1.\text{ccp} \cup n_2.\text{ccp}$
  - $n_1.\text{ccp} \leftarrow \emptyset$

What is $\text{UNION}(1, 2)$?

$$f(a, b) = a \land f(f(a, b), b) \neq a$$
Congruence closure algorithm: union-find

- **FIND** returns the representative of a node’s equivalence class.
- **UNION** combines equivalence classes for nodes $i_1$ and $i_2$:
  - $n_1, n_2 \leftarrow \text{FIND}(i_1), \text{FIND}(i_2)$
  - $n_1.\text{find} \leftarrow n_2$
  - $n_2.\text{ccp} \leftarrow n_1.\text{ccp} \cup n_2.\text{ccp}$
  - $n_1.\text{ccp} \leftarrow \emptyset$

$$f(a, b) = a \land f(f(a, b), b) \neq a$$
Congruence closure algorithm: congruent

- **CONGRUENT** takes as input two nodes and returns true iff their
  - functions are the same
  - corresponding arguments are in the same congruence class

\[ f(a, b) = a \land f(f(a, b), b) \neq a \]
Congruence closure algorithm: congruent

- **CONGRUENT** takes as input two nodes and returns true iff their
  - functions are the same
  - corresponding arguments are in the same congruence class

\[ f(a, b) = a \land f(f(a, b), b) \neq a \]
**Congruence closure algorithm: merge**

\[
\text{MERGE} (i_1, i_2) \quad \begin{align*}
n_1, n_2 &\leftarrow \text{FIND}(i_1), \text{FIND}(i_2) \\
\text{if } n_1 = n_2 \text{ then return} \\
p_1, p_2 &\leftarrow n_1.\text{cpp}, n_2.\text{cpp} \\
\text{UNION}(n_1, n_2) \\
\text{for each } t_1, t_2 \in p_1 \times p_2 \\
\text{ if } \text{CONGRUENT}(t_1, t_2) \text{ then } \\
\text{MERGE}(t_1, t_2)
\end{align*}
\]

\[
f(a, b) = a \land f(f(a, b), b) \neq a
\]

\[
\begin{align*}
1: & f \\
2: & f \\
3: & a \\
4: & b
\end{align*}
\]

\[
\begin{align*}
\{1\} & \\
\{2\} & \\
\{1, 2\} & \neq a
\end{align*}
\]
Congruence closure algorithm: merge

\[
\text{MERGE } (i_1, i_2)
\]

\[
\begin{align*}
n_1, n_2 & \leftarrow \text{FIND}(i_1), \text{FIND}(i_2) \\
\text{if } n_1 = n_2 \text{ then return} \\
p_1, p_2 & \leftarrow n_1.cpp, n_2.cpp \\
\text{UNION}(n_1, n_2) \\
\text{for each } t_1, t_2 \in p_1 \times p_2 \\
\text{if } \text{CONGRUENT}(t_1, t_2) \text{ then} \\
\text{MERGE}(t_1, t_2)
\end{align*}
\]

\[
f(a, b) = a \land f(f(a, b), b) \neq a
\]
Congruence closure algorithm: merge

\[
\text{MERGE} (i_1, i_2) \\
\begin{align*}
&n_1, n_2 \leftarrow \text{FIND}(i_1), \text{FIND}(i_2) \\
&\text{if } n_1 = n_2 \text{ then return} \\
&p_1, p_2 \leftarrow n_1\text{.cpp}, n_2\text{.cpp} \\
&\text{UNION}(n_1, n_2) \\
&\text{for each } t_1, t_2 \in p_1 \times p_2 \\
&\quad \text{if } \text{CONGRUENT}(t_1, t_2) \text{ then} \\
&\quad \text{MERGE}(t_1, t_2)
\end{align*}
\]

\[
f(a, b) = a \land f(f(a, b), b) \neq a
\]
**Congruence closure algorithm: merge**

**MERGE** \((i_1, i_2)\)

\(n_1, n_2 \leftarrow \text{FIND}(i_1), \text{FIND}(i_2)\)

**if** \(n_1 = n_2\) **then** **return**

\(p_1, p_2 \leftarrow n_1\.cpp, n_2\.cpp\)

\(\text{UNION}(n_1, n_2)\)

**for** each \(t_1, t_2 \in p_1 \times p_2\)

**if** \(\text{CONGRUENT}(t_1, t_2)\) **then**

**MERGE** \((t_1, t_2)\)

\[ f(a, b) = a \land f(f(a, b), b) \neq a \]
Summary

Today

• A brief survey of theory solvers
• Congruence closure algorithm for deciding conjunctive $T=\$ formulas

Next lecture

• Combining theories