Introduction to SMT

Lecture 2, 2012

Nikolaj Bjørner
Microsoft Research
DTU Winter course January 3rd 2012
Organized by Hanne Riis Nielson, Flemming Nielson
Background Reading

Satisfiability Modulo Theories: Introduction & Applications

Leonardo de Moura
Microsoft Research
One Microsoft Way
Redmond, WA 98052
leonardo@microsoft.com

Nikolaj Bjørner
Microsoft Research
One Microsoft Way
Redmond, WA 98052
nbjorner@microsoft.com

Abstract

Many satisfaction problems arise in many diverse areas of software and hardware verification, type inference in program analysis, test-case generation, scheduling, and graph problems. These areas share a common trait, they include a core component using logical analysis for describing states and transformations between them. The most well-known constraint satisfaction problem is propositional satisfiability, SAT, where the goal is to determine if a formula over Boolean variables, formed using connectives can be made true by choosing true/false for its variables. Some problems are more naturally expressed using richer languages, such as arithmetic. A satisfiability (of arithmetic) is then required to capture the meaning of these formulas. Solvers for such formulations are commonly called Satisfiability Modulo Theories (SMT) solvers.

SMT solvers have been the focus of increased recent attention thanks to technological advances and industrial applications. Yet, they draw on a combination of some of the most fundamental areas in computer science as well as discoveries from the past century of symbolic logic. They combine the problem of Boolean Satisfiability with domains, such as, those studied in convex optimization and term-manipulating symbolic systems. They involve the decision problem, completeness and incompleteness of logical theories, and finally complexity theory. In this article, we present an overview of the field of Satisfiability Modulo Theories, and some of its applications.

key driving factor [4]. An important ingredient is a common interchange format for benchmarks, called SMT-LIB [33], and the classification of benchmarks into various categories depending on which theories are required. Conversely, a growing number of applications are able to generate benchmarks in the SMT-LIB format to further inspire improving SMT solvers.

There is a relatively long tradition of using SMT solvers in select and specialized contexts. One prolific case is theorem proving systems such as ACL2 [26] and PVS [32]. Those use decision procedures to discharge lemmas encountered during interactive proofs. SMT solvers have also been used for a long time in the context of program verification and extended static checking [21], where verification is focused on assertion checking. Recent progress in SMT solvers, however, has enabled their use in a set of diverse applications, including interactive theorem provers and extended static checkers, but also in the context of scheduling, planning, test-case generation, model-based testing and program development, static program analysis, program synthesis, and run-time analysis, among several others.

We begin by introducing a motivating application and a simple instance of it that we will use as a running example.

1.1 An SMT Application - Scheduling

Consider the classical job shop scheduling decision problem. In this problem, there are $n$ jobs, each composed of $m$ tasks of varying duration that have to be performed consecutively on $m$ machines. The start of a new task can be delayed as long as needed in order to wait for a machine to become available, but tasks cannot be interrupted once started.

...
Plan

- General overview of what is SMT
  - Compare with SAT, first-order theorem proving,..

- Refresher on SAT and modern DPLL

- Introduction to SMT solving techniques

- Selected SMT applications (more Jan 4, 5th)
**Takeaways:**

- Syntax and semantics of SAT/SMT.
- Algorithmic principles of modern SAT solvers
- Algorithmic principles of modern SMT solvers
Some context
On Theorem Proving
Symbolic Engines: SAT, FTP and SMT

- **SAT:** Propositional Satisfiability.
  \[(\text{Tie} \lor \text{Shirt}) \land (\neg \text{Tie} \lor \neg \text{Shirt}) \land (\neg \text{Tie} \lor \text{Shirt})\]

- **FTP:** First-order Theorem Proving.
  \[\forall X,Y,Z \ [X*(Y*Z) = (X*Y)*Z] \]
  \[\forall X \ [X*\text{inv}(X) = e] \ \forall X \ [X*e = e]\]

- **SMT:** Satisfiability Modulo background Theories
  \[b + 2 = c \land A[3] \neq A[c-b+1]\]}
# SAT - Milestones

<table>
<thead>
<tr>
<th>Year</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960</td>
<td>Davis-Putnam procedure</td>
</tr>
<tr>
<td>1962</td>
<td>Davis-Logeman-Loveland</td>
</tr>
<tr>
<td>1984</td>
<td>Binary Decision Diagrams</td>
</tr>
<tr>
<td>1992</td>
<td>DIMACS SAT challenge</td>
</tr>
<tr>
<td>1994</td>
<td>SATO: clause indexing</td>
</tr>
<tr>
<td>1997</td>
<td>GRASP: conflict clause learning</td>
</tr>
<tr>
<td>1998</td>
<td>Search Restarts</td>
</tr>
<tr>
<td>2001</td>
<td>zChaff: 2-watch literal, VSIDS</td>
</tr>
<tr>
<td>2005</td>
<td>Preprocessing techniques</td>
</tr>
<tr>
<td>2007</td>
<td>Phase caching</td>
</tr>
<tr>
<td>2008</td>
<td>Cache optimized indexing</td>
</tr>
<tr>
<td>2009</td>
<td>In-processing, clause management</td>
</tr>
<tr>
<td>2010</td>
<td>Blocked clause elimination</td>
</tr>
</tbody>
</table>

Problems impossible 10 years ago are trivial today.

---

**Concept**

2002 2010

---

**Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20min timeout**

- **CPU time (in seconds)**
- **Number of problems solved**

- **Datasets:**
  - Lin#sat 02
  - Zchaff 02
  - Berkmin 561 02
  - Forklift 03
  - Siege 03
  - Zchaff 04
  - SatElite 05
  - Minisat 2.0.06
  - Picosat 07
  - Rsat 07
  - Minisat 2.1.08
  - Picosat 09
  - Glucose 09
  - Cliq 09
  - Cryptominisat 10
  - Lingeling 10
  - Minisat 2.2.10

---

**Millions of variables from HW designs**

---

Courtesy Daniel le Berre
### FTP - Milestones

<table>
<thead>
<tr>
<th>Year</th>
<th>Milestone</th>
<th>Who</th>
<th>Year</th>
<th>Milestone</th>
<th>Who</th>
</tr>
</thead>
<tbody>
<tr>
<td>1930</td>
<td>Hebrand's theorem</td>
<td>Herbrand</td>
<td>1970</td>
<td>Completion and saturation procedures</td>
<td>many people and provers</td>
</tr>
<tr>
<td>1934</td>
<td>Sequent calculi</td>
<td>Gentzen</td>
<td>1970</td>
<td>Knuth-Bendix ordering</td>
<td>Knuth; Bendix</td>
</tr>
<tr>
<td>1934</td>
<td>Inverse method</td>
<td>Gentzen</td>
<td>1971</td>
<td>Selection function</td>
<td>Kowalski; Kuehner</td>
</tr>
<tr>
<td>1955</td>
<td>Semantic tableaux</td>
<td>Beth</td>
<td>1972</td>
<td>Built-in equational theories</td>
<td>Plotkin</td>
</tr>
<tr>
<td></td>
<td>Herbrand-based theorem</td>
<td>Wang Hao</td>
<td>1972</td>
<td>Prolog</td>
<td>Colmerauer</td>
</tr>
<tr>
<td>1960</td>
<td>proving</td>
<td>Davis; Putnam</td>
<td>1974</td>
<td>Saturation algorithms</td>
<td>Overbeek</td>
</tr>
<tr>
<td>1960</td>
<td>Ordered resolution</td>
<td>Davis; Logemann; Loveland</td>
<td>1975</td>
<td>Completeness of paramodulation</td>
<td>Brand</td>
</tr>
<tr>
<td>1962</td>
<td>DLL</td>
<td>Davis; Logemann; Loveland</td>
<td>1975</td>
<td>AC-unification</td>
<td>Stickel</td>
</tr>
<tr>
<td>1963</td>
<td>First-order inverse method</td>
<td>Maslov</td>
<td>1976</td>
<td>Resolution as a decision procedure</td>
<td>Joyner</td>
</tr>
<tr>
<td>1965</td>
<td>Unification</td>
<td>J. Robinson</td>
<td>1979</td>
<td>Basic paramodulation</td>
<td>Degtyarev</td>
</tr>
<tr>
<td>1965</td>
<td>First-order resolution</td>
<td>J. Robinson</td>
<td>1980</td>
<td>Lexicographic path orderings</td>
<td>Kamin; Levy</td>
</tr>
<tr>
<td>1965</td>
<td>Subsumption</td>
<td>J. Robinson</td>
<td>1985</td>
<td>Theory resolution</td>
<td>Stickel</td>
</tr>
<tr>
<td>1967</td>
<td>Orderings</td>
<td>Slagle</td>
<td>1986</td>
<td>Transformation</td>
<td>Plaisted; Greenbaum</td>
</tr>
<tr>
<td>1967</td>
<td>Demodulation or rewriting</td>
<td>Wos; G. Robinson; Carson; Shalla</td>
<td>1988</td>
<td>Superposition</td>
<td>Zhang</td>
</tr>
<tr>
<td>1968</td>
<td>Model elimination</td>
<td>Loveland</td>
<td>1988</td>
<td>Model construction</td>
<td>Zhang</td>
</tr>
<tr>
<td>1969</td>
<td>Paramodulation</td>
<td>G. Robinson; Wos</td>
<td>1989</td>
<td>Term indexing</td>
<td>Stickel; Overbeek</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Some success stories:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- **Open Problems (of 25 years):**                                                                                             |
  - XCB: $X \equiv ((X \equiv Y) \equiv (Z \equiv Y)) \equiv Z$                                                                  |
  - Knowledge Ontologies                                                                                                         |
  - GBs of formulas                                                                                                              |

**Courtesy Andrei Voronkov, Manchester U**
SMT - Milestones

<table>
<thead>
<tr>
<th>Year</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1977</td>
<td>Efficient Equality Reasoning</td>
</tr>
<tr>
<td>1979</td>
<td>Theory Combination Foundations</td>
</tr>
<tr>
<td>1979</td>
<td>Arithmetic + Functions</td>
</tr>
<tr>
<td>1982</td>
<td>Combining Canonizing Solvers</td>
</tr>
<tr>
<td>1992-8</td>
<td>Systems: PVS, Simplify, STeP, SVC</td>
</tr>
<tr>
<td>2002</td>
<td>Theory Clause Learning</td>
</tr>
<tr>
<td>2005</td>
<td>SMT competition</td>
</tr>
<tr>
<td>2006</td>
<td>Efficient SAT + Simplex</td>
</tr>
<tr>
<td>2007</td>
<td>Efficient Equality Matching</td>
</tr>
<tr>
<td>2009</td>
<td>Combinatory Array Logic, ...</td>
</tr>
</tbody>
</table>

Includes progress from SAT:

15KLOC + 285KLOC = Z3

Z3 (of ’07) Time On Boogie Regression

Z3 Time On VCC Regression

1 sec

Nov 08

March 09
Introducing SMT by examples
Satisfiability Modulo Theories (SMT)

Is formula $\varphi$ satisfiable modulo theory $T$?

SMT solvers have specialized algorithms for $T$. 
Satisfiability Modulo Theories (SMT)

$x + 2 = y \Rightarrow f(\text{select}(\text{store}(a, x, 3), y - 2)) = f(y - x + 1)$

Array Theory  Arithmetic  Uninterpreted Functions

\[
\text{select}(\text{store}(a, i, v), i) = v \\
i \neq j \Rightarrow \text{select}(\text{store}(a, i, v), j) = \text{select}(a, j)
\]
\[ b + 2 = c \quad \text{and} \quad f(\text{select}(\text{store}(a, b, 3), c-2)) \neq f(c-b+1) \]
b + 2 = c and f(select(store(a, b, 3), c - 2)) ≠ f(c - b + 1)
b + 2 = c \text{ and } f(\text{select}(\text{store}(a,b,3), c-2)) \neq f(c-b+1)
Satisfiability Modulo Theories (SMT)

\[ b + 2 = c \text{ and } f(\text{select}(\text{store}(a, b, 3), c - 2)) \neq f(c - b + 1) \]
\[ b + 2 = c \quad \text{and} \quad f(\text{select}(\text{store}(a, b, 3), c-2)) \neq f(c-b+1) \]

Substituting \( c \) by \( b+2 \)
b + 2 = c and \( f(\text{select(\text{store}(a,b,3), b+2-2)}) \neq f(b+2-b+1) \)

Simplifying
b + 2 = c and \( f(\text{select}(\text{store}(a, b, 3), b)) \neq f(3) \)
b + 2 = c and f(select(store(a,b,3), b)) ≠ f(3)

Applying array theory axiom

\[ \forall a, i, v: \text{select}(\text{store}(a, i, v), i) = v \]
b + 2 = c and \( f(3) \neq f(3) \)

Inconsistent/Unsatisfiable
Job Shop Scheduling

Machines

Tasks

Jobs

$P = NP$?

$\zeta(s) = 0 \Rightarrow s = \frac{1}{2} + ir$
**Constraints:**

**Precedence:** between two tasks of the same job

**Resource:** Machines execute at most one job at a time

\[[\text{start}_{2,2}..\text{end}_{2,2}] \cap [\text{start}_{4,2}..\text{end}_{4,2}] = \emptyset\]
Constraints:

**Precedence:**

\[ \begin{align*}
    t_{2,3} & \text{ - start time of job 2 on mach 3} \\
    d_{2,3} & \text{ - duration of job 2 on mach 3} \\
    t_{2,3} + d_{2,3} & \leq t_{2,4}
\end{align*} \]

**Resource:**

\[ [\text{start}_{2,2}..\text{end}_{2,2}] \cap [\text{start}_{4,2}..\text{end}_{4,2}] = \emptyset \]

Encoding:

Not convex

\[ t_{2,2} + d_{2,2} \leq t_{4,2} \quad \lor \quad t_{4,2} + d_{4,2} \leq t_{2,2} \]
### Job Shop Scheduling

<table>
<thead>
<tr>
<th>(d_{i,j})</th>
<th>Machine 1</th>
<th>Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Job 2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job 3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\[\text{max} = 8\]

**Solution**

\[t_{1,1} = 5, \ t_{1,2} = 7, \ t_{2,1} = 2,\]
\[t_{2,2} = 6, \ t_{3,1} = 0, \ t_{3,2} = 3\]

**Encoding**

\[(t_{1,1} \geq 0) \land (t_{1,2} \geq t_{1,1} + 2) \land (t_{1,2} + 1 \leq 8) \land \]
\[(t_{2,1} \geq 0) \land (t_{2,2} \geq t_{2,1} + 3) \land (t_{2,2} + 1 \leq 8) \land \]
\[(t_{3,1} \geq 0) \land (t_{3,2} \geq t_{3,1} + 2) \land (t_{3,2} + 3 \leq 8) \land \]
\([(t_{1,1} \geq t_{2,1} + 3) \lor (t_{2,1} \geq t_{1,1} + 2)] \land \]
\([(t_{1,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{1,1} + 2)] \land \]
\([(t_{2,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{2,1} + 3)] \land \]
\([(t_{1,2} \geq t_{2,2} + 1) \lor (t_{2,2} \geq t_{1,2} + 1)] \land \]
\([(t_{1,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{1,2} + 1)] \land \]
\([(t_{2,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{2,2} + 1)]\)
Job Shop Scheduling

(t_{1,1} \geq 0) \land (t_{1,2} \geq t_{1,1} + 2) \land (t_{1,2} + 1 \leq 8) \land
(t_{2,1} \geq 0) \land (t_{2,2} \geq t_{2,1} + 3) \land (t_{2,2} + 1 \leq 8) \land
(t_{3,1} \geq 0) \land (t_{3,2} \geq t_{3,1} + 2) \land (t_{3,2} + 3 \leq 8) \land
((t_{1,1} \geq t_{2,1} + 3) \lor (t_{2,1} \geq t_{1,1} + 2)) \land
((t_{1,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{1,1} + 2)) \land
((t_{2,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{2,1} + 3)) \land
((t_{1,2} \geq t_{2,2} + 1) \lor (t_{2,2} \geq t_{1,2} + 1)) \land
((t_{1,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{1,2} + 1)) \land
((t_{2,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{2,2} + 1))

\begin{align*}
z - t_{1,1} & \leq 0 \\
z - t_{2,1} & \leq 0 \\
z - t_{3,1} & \leq 0 \\
t_{3,2} - z & \leq 5 \\
t_{3,1} - t_{3,2} & \leq -2 \\
t_{2,1} - t_{3,1} & \leq -3 \\
t_{1,1} - t_{2,1} & \leq -2
\end{align*}

Efficient solvers:
- Floyd-Warshall algorithm
- Ford-Fulkerson algorithm

\[ z - z = 5 - 2 - 3 - 2 = -2 < 0 \]
**Modern DPLL in a nutshell**

- **Initialize** \( \epsilon \models F \)
  
  \( F \) is a set of clauses

- **Decide** \( M \models F \Rightarrow M, \ell \models F \)
  
  \( \ell \) is unassigned

- **Propagate** \( M \models F, C \lor \ell \Rightarrow M, \ell^{C\lor\ell} \models F, C \lor \ell \)
  
  \( C \) is false under \( M \)

- **Conflict** \( M \models F, C \Rightarrow M \models F, C \models C \)
  
  \( C \) is false under \( M \)

- **Resolve** \( M \models F \models C' \lor \neg \ell \Rightarrow M \models F, C' \lor C \)
  
  \( \ell^{C\lor\ell} \in M \)

- **Learn** \( M \models F \models C \Rightarrow M \models F, C \models C \)

- **Backjump** \( M \models \neg \ell M' \models F \models C \lor \ell \Rightarrow M \ell^{C\lor\ell} \models F \)
  
  \( C \) has no literals in \( M' \)

- **Unsat** \( M \models F \models \emptyset \Rightarrow \text{Unsat} \)

- **Sat** \( M \models F \Rightarrow M \)
  
  \( F \) true under \( M \)

- **Restart** \( M \models F \Rightarrow \epsilon \models F \)

Adapted and modified from [Nieuwenhuis, Oliveras, Tinelli J.ACM 06]
SMT solving
**DPLL(T) solver interaction**

**T- Propagate**  
\[ M | F, C \lor \ell \Rightarrow M, \ell^{C \lor \ell} | F, C \lor \ell \quad \text{C is false under } T + M \]

**T- Conflict**  
\[ M | F \Rightarrow M | F | \neg M' \quad M' \subseteq M \text{ and } M' \text{ is false under } T \]

**T- Propagate**  
\[ a > b, b > c \quad | \quad F, a \leq c \lor b \leq d \Rightarrow \]
\[ a > b, b > c, b \leq d^{a \leq c \lor b \leq d} \quad | \quad F, a \leq c \lor b \leq d \]

**T- Conflict**  
\[ M | F \Rightarrow M | F, a \leq b \lor b \leq c \lor c < a \]
\[ \text{where } a > b, b > c, a \leq c \subseteq M \]
Main components of modern SMT Solvers

Purification

\[ x + 2 = y \land f \left( \text{read} \left( \text{write}(a, x, 3), y - 2 \right) \right) \neq f \left( y - x + 1 \right) \]

\[ x + 2 = y \land f \left( v \right) \neq f \left( y - x + 1 \right) \land \]

\[ \text{read} \left( \text{write}(a, x, 3), y - 2 \right) = v \]

Note:
read is just another name for select,
write is just another name for store
Purification

\[ x + 2 = y \land f(v) \neq f(y - x + 1) \land \]
\[ \text{read(write}(a, x, 3), y - 2) = v \]

\[ x + 2 = y \land u = y - x + 1 \land \]
\[ \text{read(write}(a, x, 3), y - 2) = v \land \]
\[ f(v) \neq f(u) \]
Main components of modern SMT Solvers

Purification

\[ x + 2 = y \land u = y - x + 1 \land \]
\[ \text{read(write}(a, x, 3), y - 2) = v \land \]
\[ f(v) \neq f(u) \]

Arithmetic \[ x + 2 = y \land u = y - x + 1 \land z = 3 \land w = y - 2 \land \]

Arrays \[ \text{read(write}(a, x, z), w) = v \land \]

Functions \[ f(v) \neq f(u) \]
Propositional Abstraction

\[ x + 2 = y \land u = y - x + 1 \land z = 3 \land w = y - 2 \land \]

\[ read(\text{write}(a, x, z), w) = v \land \]

\[ f(v) \neq f(u) \]

\[ x + 2 = y \land u = y - x + 1 \land z = 3 \land w = y - 2 \land \]

\[ read(\text{write}(a, x, z), w) = v \land \]

\[ f(v) \neq f(u) \]

\[ p_1 \]

\[ p_2 \]

\[ p_3 \]

\[ p_4 \]

\[ p_5 \]

\[ \neg p_6 \]
Main components of modern SMT Solvers

Propositional Assignment

\[ x + 2 = y \land u = y - x + 1 \land z = 3 \land w = y - 2 \land \]
\[ p_1 \]
\[ p_2 \]
\[ p_3 \]
\[ p_4 \]
\[ \text{read(write}(a, x, z), w) = v \land \]
\[ p_5 \]
\[ f(v) \neq f(u) \]
\[ \neg p_6 \]

Using SAT solver

\[ p_1 \leftarrow \text{true}, \]
\[ p_2 \leftarrow \text{true}, \]
\[ p_3 \leftarrow \text{true}, \]
\[ p_4 \leftarrow \text{true}, \]
\[ p_5 \leftarrow \text{true}, \]
\[ p_6 \leftarrow \text{false} \]
Main components of modern SMT Solvers

Theory Solving

**Arithmetic**
\[ x + 2 = y \land u = y - x + 1 \land z = 3 \land w = y - 2 \]
\[ w = x \]

**Arrays**
\[ \text{read}(\text{write}(a, x, z), w) = v \]
\[ u = z = 3 \]
\[ z = v \]

**Free functions**
\[ f(v) \neq f(u) \]

Theories exchange equalities between shared variables.
The running example was easy. But what about:

\[ x = f(z), f(x) \neq f(y), 0 \leq x \leq 1, 0 \leq y \leq 1, z = y - 1 \]

Either \( x = 0 \) or \( x = 1 \).

\[ y = 0 \text{ or } y = 1 \quad (x, y \text{ are integers}) \]

Arithmetic module needs to somehow learn that \( x \neq y \).

- Integer linear arithmetic is non-convex.
Combining Theories

An Appetizer
Delayed Theory Combination solution [2006 Bruttomesso et.al.]

Add equality literals for every pair of shared variables:

\[ x = f(z), f(x) \neq f(y), 0 \leq x \leq 1, 0 \leq y \leq 1, z = y - 1 \]

\[ (x = y \lor x \neq y), (x = z \lor x \neq z), (y = z \lor y \neq z) \]

😊: Solvers work completely independently.
😊: Works with non-convex theories.
',{eqn: 1}: O(n^2) up-front cost. No use of propagation.
Idea:
- Have solvers produce models.
- Use models to introduce equalities on demand.
  If $M \models \text{Assertions, and } M \models x = y$
  Then guess $x = y$

😊: No up-front $O(n^2)$ cost of adding equalities
😊: Works with non-convex theories
😊: Models are conservative approximations

$M \models \text{Assertions} \models x = y$ then $M \models x = y$
$x = f(z), f(x) \neq f(y), 0 \leq x \leq 1, 0 \leq y \leq 1, z = y - 1$
### Model-based - Example

<table>
<thead>
<tr>
<th>$\mathcal{T}_E$</th>
<th>$\mathcal{T}_A$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Literals</strong></td>
<td><strong>Model</strong></td>
</tr>
<tr>
<td>$x = f(z)$</td>
<td>$x^E = *_1$</td>
</tr>
<tr>
<td>$f(x) \neq f(y)$</td>
<td>$y^E = *_2$</td>
</tr>
<tr>
<td>$z$</td>
<td>$z^E = *_3$</td>
</tr>
<tr>
<td>$f(x)$</td>
<td>$f^E = {*_1 \mapsto <em>_4;</em>_2 \mapsto <em>_5,</em>_3 \mapsto *_1,}$</td>
</tr>
<tr>
<td>$f(y)$</td>
<td></td>
</tr>
</tbody>
</table>

Assume $x = y$
# Model-based - Example

<table>
<thead>
<tr>
<th>( T_\mathcal{E} )</th>
<th>( T_\mathcal{A} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Literals</strong></td>
<td><strong>Eq. Classes</strong></td>
</tr>
<tr>
<td>( x = f(z) )</td>
<td>( { x, y, f(z) } )</td>
</tr>
<tr>
<td>( f(x) \neq f(y) )</td>
<td>( { z } )</td>
</tr>
<tr>
<td>( x = y )</td>
<td>( { f(x), f(y) } )</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Unsatisfiable
<table>
<thead>
<tr>
<th>$\mathcal{T}_E$</th>
<th>$\mathcal{T}_A$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Literals</strong></td>
<td><strong>Eq. Classes</strong></td>
</tr>
<tr>
<td>$x = f(z)$</td>
<td>${x, f(z)}$</td>
</tr>
<tr>
<td>$f(x) \neq f(y)$</td>
<td>${y}$</td>
</tr>
<tr>
<td>$x \neq y$</td>
<td>${z}$</td>
</tr>
<tr>
<td></td>
<td>${f(x)}$</td>
</tr>
<tr>
<td></td>
<td>${f(y)}$</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Backtrack, and assert $x \neq y$.

$\mathcal{T}_A$ model need to be fixed.
## Model-based - Example

<table>
<thead>
<tr>
<th>$T_\mathcal{E}$</th>
<th>$T_\mathcal{A}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Literals</strong></td>
<td><strong>Eq. Classes</strong></td>
</tr>
<tr>
<td>$x = f(z)$</td>
<td>${x, f(z)}$</td>
</tr>
<tr>
<td>$f(x) \neq f(y)$</td>
<td>${y}$</td>
</tr>
<tr>
<td>$x \neq y$</td>
<td>${z}$</td>
</tr>
<tr>
<td></td>
<td>${f(x)}$</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assume $x = z$
# Model-based - Example

<table>
<thead>
<tr>
<th>$T_\varepsilon$</th>
<th>$T_A$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Literals</strong></td>
<td><strong>Eq. Classes</strong></td>
</tr>
<tr>
<td>$x = f(z)$</td>
<td>${x, z, f(x), f(z)}$</td>
</tr>
<tr>
<td>$f(x) \neq f(y)$</td>
<td>${y}$</td>
</tr>
<tr>
<td>$x \neq y$</td>
<td>${f(y)}$</td>
</tr>
<tr>
<td>$x = z$</td>
<td>$f^\varepsilon = {*_1 \mapsto *_1, \text{ else } \mapsto *_4}$</td>
</tr>
</tbody>
</table>

Satisfiable