Theory Solvers

Given a theory $T$, a Theory Solver for $T$ takes as input a set $\Phi$ of literals and determines whether $\Phi$ is $T$-satisfiable.

$\Phi$ is $T$-satisfiable iff there is some model $M$ of $T$ such that each formula in $\Phi$ holds in $M$.

We next consider some examples of theory solvers.

Congruence Closure and $QF_{UF}$

Recall that $QF_{UF}$ is the theory with only equality and uninterpreted function symbols.

If $\Gamma$ is a set of equalities and $\Delta$ is a set of disequalities, then the satisfiability of $\Gamma \cup \Delta$ in $QF_{UF}$ can be determined as follows [NO80, DST80] :

- Let $\tau$ be the set of terms appearing in $\Gamma \cup \Delta$.
- Let $\sim$ be the equivalence relation on $\tau$ induced by $\Gamma$ (i.e. $t_1 \sim t_2$ iff $t_1 = t_2 \in \Gamma$ or $t_2 = t_1 \in \Gamma$).
- Let $\sim^*$ be the congruence closure of $\sim$, obtained by closing $\sim$ with respect to the congruence property:
  \[
  \overline{s} = \overline{t} \rightarrow f(\overline{s}) = f(\overline{t}).
  \]
- $\Gamma \cup \Delta$ is satisfiable iff for each $s \neq t \in \Delta$, $s \not\sim^* t$. 

Roadmap

Theory Solvers

- Examples of Theory Solvers
- Combining Theory Solvers
- Extending Theory Solvers for SMT

From SAT to SMT

- Abstract DPLL
- Abstract DPLL Modulo Theories
- Key Optimizations
- Quantifier Instantiation
**A Solver for QF_UF**

`union` and `find` are abstract operations for manipulating equivalence classes.

`union(x, y)` makes `y` the new equivalence class representative for `x`.

`find(x)` returns the unique representative for the equivalence class containing `x`.

The **signature** of a term is defined as:

\[
sig(f(x_1, \ldots, x_n)) = f(find(x_1), \ldots, find(x_n))
\]

**Example**

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

<table>
<thead>
<tr>
<th><code>t</code></th>
<th><code>find(t)</code></th>
<th><code>sig(t)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td><code>a</code></td>
<td><code>a</code></td>
</tr>
<tr>
<td><code>f(a)</code></td>
<td><code>f(a)</code></td>
<td><code>f(a)</code></td>
</tr>
<tr>
<td><code>f(f(a))</code></td>
<td><code>f(f(a))</code></td>
<td><code>f(f(a))</code></td>
</tr>
<tr>
<td><code>f(f(f(a)))</code></td>
<td><code>f(f(f(a)))</code></td>
<td><code>f(f(f(a)))</code></td>
</tr>
<tr>
<td><code>b</code></td>
<td><code>b</code></td>
<td><code>b</code></td>
</tr>
<tr>
<td><code>g(a, b)</code></td>
<td><code>g(a, b)</code></td>
<td><code>g(a, b)</code></td>
</tr>
<tr>
<td><code>g(f(a), b)</code></td>
<td><code>g(f(a), b)</code></td>
<td><code>g(f(a), b)</code></td>
</tr>
</tbody>
</table>

A Solver for QF_UF

\[
CC(\Gamma, \Delta)
\]

while \( \Gamma \neq \emptyset \)

Remove some equality \( a = b \) from \( \Gamma \);

\[
\text{Merge}(\text{find}(a), \text{find}(b));
\]

if \( \text{find}(a) = \text{find}(b) \) for some \( a \neq b \in \Delta \) then

return \( \text{False} \);

return \( \text{True} \);

\[
\text{Merge}(a, b)
\]

if \( a = b \) then return;

Let \( A \) be the set of terms containing \( a \) as an argument

\[
\text{union}(a, b);
\]

foreach \( x \in A \)

if \( \text{sig}(x) = \text{sig}(y) \) for some \( y \) then

\[
\text{Merge}(\text{find}(x), \text{find}(y));
\]

**Example**

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

<table>
<thead>
<tr>
<th><code>t</code></th>
<th><code>find(t)</code></th>
<th><code>sig(t)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td><code>a</code></td>
<td><code>a</code></td>
</tr>
<tr>
<td><code>f(a)</code></td>
<td><code>f(a)</code></td>
<td><code>f(a)</code></td>
</tr>
<tr>
<td><code>f(f(a))</code></td>
<td><code>f(f(a))</code></td>
<td><code>f(f(a))</code></td>
</tr>
<tr>
<td><code>f(f(f(a)))</code></td>
<td><code>f(f(f(a)))</code></td>
<td><code>f(f(f(a)))</code></td>
</tr>
<tr>
<td><code>b</code></td>
<td><code>b</code></td>
<td><code>b</code></td>
</tr>
<tr>
<td><code>g(a, b)</code></td>
<td><code>g(a, b)</code></td>
<td><code>g(a, b)</code></td>
</tr>
<tr>
<td><code>g(f(a), b)</code></td>
<td><code>g(f(a), b)</code></td>
<td><code>g(f(a), b)</code></td>
</tr>
</tbody>
</table>
Example

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

<table>
<thead>
<tr>
<th>(t)</th>
<th>(\text{find}(t))</th>
<th>(\text{sig}(t))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(a)</td>
<td>(a)</td>
</tr>
<tr>
<td>(f(a))</td>
<td>(f(a))</td>
<td>(f(a))</td>
</tr>
<tr>
<td>(f(f(a)))</td>
<td>(a)</td>
<td>(f(f(a)))</td>
</tr>
<tr>
<td>(f(f(f(a))))</td>
<td>(f(f(a)))</td>
<td>(f(a))</td>
</tr>
<tr>
<td>(b)</td>
<td>(b)</td>
<td>(b)</td>
</tr>
<tr>
<td>(g(a, b))</td>
<td>(g(a, b))</td>
<td>(g(a, b))</td>
</tr>
<tr>
<td>(g(f(a), b))</td>
<td>(g(f(a), b))</td>
<td>(g(f(a), b))</td>
</tr>
</tbody>
</table>

Example

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

<table>
<thead>
<tr>
<th>(t)</th>
<th>(\text{find}(t))</th>
<th>(\text{sig}(t))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(a)</td>
<td>(a)</td>
</tr>
<tr>
<td>(f(a))</td>
<td>(f(a))</td>
<td>(f(a))</td>
</tr>
<tr>
<td>(f(f(a)))</td>
<td>(a)</td>
<td>(f(f(a)))</td>
</tr>
<tr>
<td>(f(f(f(a))))</td>
<td>(f(f(a)))</td>
<td>(f(a))</td>
</tr>
<tr>
<td>(b)</td>
<td>(b)</td>
<td>(b)</td>
</tr>
<tr>
<td>(g(a, b))</td>
<td>(g(a, b))</td>
<td>(g(a, b))</td>
</tr>
<tr>
<td>(g(f(a), b))</td>
<td>(g(f(a), b))</td>
<td>(g(f(a), b))</td>
</tr>
</tbody>
</table>

Example

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

<table>
<thead>
<tr>
<th>(t)</th>
<th>(\text{find}(t))</th>
<th>(\text{sig}(t))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(a)</td>
<td>(a)</td>
</tr>
<tr>
<td>(f(a))</td>
<td>(f(a))</td>
<td>(f(a))</td>
</tr>
<tr>
<td>(f(f(a)))</td>
<td>(a)</td>
<td>(f(f(a)))</td>
</tr>
<tr>
<td>(f(f(f(a))))</td>
<td>(f(f(a)))</td>
<td>(f(a))</td>
</tr>
<tr>
<td>(b)</td>
<td>(b)</td>
<td>(b)</td>
</tr>
<tr>
<td>(g(a, b))</td>
<td>(g(a, b))</td>
<td>(g(a, b))</td>
</tr>
<tr>
<td>(g(f(a), b))</td>
<td>(g(f(a), b))</td>
<td>(g(f(a), b))</td>
</tr>
</tbody>
</table>
### Example

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

<table>
<thead>
<tr>
<th>( t )</th>
<th>( \text{find}(t) )</th>
<th>( \text{sig}(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( a )</td>
<td>( a )</td>
</tr>
<tr>
<td>( f(a) )</td>
<td>( f(a) )</td>
<td>( f(a) )</td>
</tr>
<tr>
<td>( f(f(a)) )</td>
<td>( a )</td>
<td>( f(f(a)) )</td>
</tr>
<tr>
<td>( f(f(f(a))) )</td>
<td>( f(a) )</td>
<td>( f(a) )</td>
</tr>
<tr>
<td>( b )</td>
<td>( b )</td>
<td>( b )</td>
</tr>
<tr>
<td>( g(a, b) )</td>
<td>( g(a, b) )</td>
<td>( g(a, b) )</td>
</tr>
<tr>
<td>( g(f(a), b) )</td>
<td>( g(f(a), b) )</td>
<td>( g(f(a), b) )</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>( t )</th>
<th>( \text{find}(t) )</th>
<th>( \text{sig}(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( a )</td>
<td>( a )</td>
</tr>
<tr>
<td>( f(a) )</td>
<td>( f(a) )</td>
<td>( f(a) )</td>
</tr>
<tr>
<td>( f(f(a)) )</td>
<td>( a )</td>
<td>( f(f(a)) )</td>
</tr>
<tr>
<td>( f(f(f(a))) )</td>
<td>( f(a) )</td>
<td>( f(a) )</td>
</tr>
<tr>
<td>( b )</td>
<td>( b )</td>
<td>( b )</td>
</tr>
<tr>
<td>( g(a, b) )</td>
<td>( g(a, b) )</td>
<td>( g(a, b) )</td>
</tr>
<tr>
<td>( g(f(a), b) )</td>
<td>( g(f(a), b) )</td>
<td>( g(f(a), b) )</td>
</tr>
</tbody>
</table>

### Example

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

<table>
<thead>
<tr>
<th>( t )</th>
<th>( \text{find}(t) )</th>
<th>( \text{sig}(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( a )</td>
<td>( a )</td>
</tr>
<tr>
<td>( f(a) )</td>
<td>( f(a) )</td>
<td>( f(a) )</td>
</tr>
<tr>
<td>( f(f(a)) )</td>
<td>( a )</td>
<td>( f(f(a)) )</td>
</tr>
<tr>
<td>( f(f(f(a))) )</td>
<td>( f(a) )</td>
<td>( f(a) )</td>
</tr>
<tr>
<td>( b )</td>
<td>( b )</td>
<td>( b )</td>
</tr>
<tr>
<td>( g(a, b) )</td>
<td>( g(a, b) )</td>
<td>( g(a, b) )</td>
</tr>
<tr>
<td>( g(f(a), b) )</td>
<td>( g(f(a), b) )</td>
<td>( g(f(a), b) )</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>( t )</th>
<th>( \text{find}(t) )</th>
<th>( \text{sig}(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( a )</td>
<td>( a )</td>
</tr>
<tr>
<td>( f(a) )</td>
<td>( a )</td>
<td>( f(a) )</td>
</tr>
<tr>
<td>( f(f(a)) )</td>
<td>( a )</td>
<td>( f(f(a)) )</td>
</tr>
<tr>
<td>( f(f(f(a))) )</td>
<td>( a )</td>
<td>( f(a) )</td>
</tr>
<tr>
<td>( b )</td>
<td>( b )</td>
<td>( b )</td>
</tr>
<tr>
<td>( g(a, b) )</td>
<td>( g(a, b) )</td>
<td>( g(a, b) )</td>
</tr>
<tr>
<td>( g(f(a), b) )</td>
<td>( g(f(a), b) )</td>
<td>( g(f(a), b) )</td>
</tr>
</tbody>
</table>
\begin{example}
\[
f(f(a)) = a \land f(f(f(a))) = a \land g(a, b) \neq g(f(a), b)
\]
\begin{tabular}{|c|c|c|}
  \hline
  t & \text{find}(t) & \text{sig}(t) \\
  \hline
  a & a & a \\
  f(a) & a & f(a) \\
  f(f(a)) & a & f(a) \\
  f(f(f(a))) & a & f(a) \\
  b & b & b \\
  g(a, b) & g(a, b) & g(a, b) \\
  g(f(a), b) & g(f(a), b) & g(f(a), b) \\
  \hline
\end{tabular}
\end{example}
### Example

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

<table>
<thead>
<tr>
<th>( t )</th>
<th>( \text{find}(t) )</th>
<th>( \text{sig}(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( a )</td>
<td>( a )</td>
</tr>
<tr>
<td>( f(a) )</td>
<td>( a )</td>
<td>( f(a) )</td>
</tr>
<tr>
<td>( f(f(a)) )</td>
<td>( a )</td>
<td>( f(a) )</td>
</tr>
<tr>
<td>( f(f(f(a))) )</td>
<td>( a )</td>
<td>( f(a) )</td>
</tr>
<tr>
<td>( b )</td>
<td>( b )</td>
<td>( b )</td>
</tr>
<tr>
<td>( g(a, b) )</td>
<td>( g(a, b) )</td>
<td>( g(a, b) )</td>
</tr>
<tr>
<td>( g(f(a), b) )</td>
<td>( g(a, b) )</td>
<td>( g(a, b) )</td>
</tr>
</tbody>
</table>

### Difference Logic

In difference logic [NO05], we are interested in the satisfiability of a conjunction of arithmetic atoms.

Each atom is of the form \( x - y \triangleq c \), where \( x \) and \( y \) are variables, \( c \) is a numeric constant, and \( \triangleq \in \{ =, <, \leq, >, \geq \} \).

The variables can range over either the integers (\( QF\_IDL \)) or the reals (\( QF\_RDL \)).

\[ \text{find}(g(a, b)) = \text{find}(g(f(a), b)) \rightarrow \text{Unsatisfiable} \]
### Difference Logic

The first step is to rewrite everything in terms of $\leq$:

- $x - y = c \implies x - y \leq c \land x - y \geq c$
- $x - y \geq c \implies y - x \leq -c$
- $x - y > c \implies y - x < -c$
- $x - y < c \implies x - y \leq c - 1$ (integers)
- $x - y < c \implies x - y \leq c - \delta$ (reals)

Now we have a conjunction of literals, all of the form $x - y \leq c$.

From these literals, we form a weighted directed graph with a vertex for each variable.

For each literal $x - y \leq c$, there is an edge $x \overset{c}{\rightarrow} y$.

The set of literals is satisfiable iff there is no cycle for which the sum of the weights on the edges is negative.

There are a number of efficient algorithms for detecting negative cycles in graphs [CG96].

#### Example: $QF_{IDL}$

$$x - y = 5 \land z - y \geq 2 \land z - x > 2 \land w - x = 2 \land z - w < 0$$

- $x - y = 5 \implies x - y \leq 5 \land y - x \leq -5$
- $z - y \geq 2 \implies y - z \leq -2$
- $z - x > 2 \implies x - z \leq -3$
- $w - x = 2 \implies w - x \leq 2 \land x - w \leq -2$
- $z - w < 0 \implies z - w \leq -1$
**Roadmap**

**Theory Solvers**
- Examples of Theory Solvers
- Combining Theory Solvers
- Extending Theory Solvers for SMT

From SAT to SMT
- Abstract DPLL
- Abstract DPLL Modulo Theories
- Key Optimizations
- Quantifier Instantiation

---

**Combining Theory Solvers**

Theory solvers become much more useful if they can be used together.

\[
\text{mux}_\text{sel} = 0 \rightarrow \text{mux}_\text{out} = \text{select}(\text{regfile, addr}) \\
\text{mux}_\text{sel} = 1 \rightarrow \text{mux}_\text{out} = \text{ALU}(\text{alu0, alu1})
\]

For such formulas, we are interested in satisfiability with respect to a *combination* of theories.

Fortunately, there exist methods for combining theory solvers. The standard technique for this is the Nelson-Oppen method [NO79, TH96].

---

**The Nelson-Oppen Method**

The Nelson-Oppen method is applicable when:
1. The theories have *no shared symbols* (other than equality).
2. The theories are *stably-infinite.*
   A theory $T$ is *stably-infinite* if every $T$-satisfiable quantifier-free formula is satisfiable in an infinite model.
3. The formulas to be tested for satisfiability are *quantifier-free*

Many theories fit these criteria, and extensions exist in some cases when they do not.
The Nelson-Oppen Method

If there exists an arrangement such that both $Sat_1$ and $Sat_2$ succeed, then $\phi$ is $T_1 \cup T_2$-satisfiable.

If no such arrangement exists, then $\phi$ is $T_1 \cup T_2$-unsatisfiable.

Example

Consider the following $QF_UFLIA$ formula:

$$\phi = 1 \leq x \land x \leq 2 \land f(x) \neq f(1) \land f(x) \neq f(2).$$

We first convert $\phi$ to a separate form:

$$\phi_{UF} = f(x) \neq f(y) \land f(x) \neq f(z)$$
$$\phi_{LIA} = 1 \leq x \land x \leq 2 \land y = 1 \land z = 2$$

The shared variables are $\{x, y, z\}$. There are 5 possible arrangements based on equivalence classes of $x$, $y$, and $z$.

Example

$$\phi_{UF} = f(x) \neq f(y) \land f(x) \neq f(z)$$
$$\phi_{LIA} = 1 \leq x \land x \leq 2 \land y = 1 \land z = 2$$

The shared variables are $\{x, y, z\}$. There are 5 possible arrangements based on equivalence classes of $x$, $y$, and $z$.

Therefore, $\phi$ is unsatisfiable.

Roadmap

Theory Solvers

- Examples of Theory Solvers
- Combining Theory Solvers
- Extending Theory Solvers for SMT

From SAT to SMT

- Abstract DPLL
- Abstract DPLL Modulo Theories
- Key Optimizations
- Quantifier Instantiation
Desirable Characteristics of Theory Solvers

Theory solvers must be able to determine whether a conjunction of literals is satisfiable.

However, in order to integrate a theory solver into a modern SMT solver, it is helpful if the theory solvers can do more.

Some desirable characteristics of theory solvers include:

- **Incrementality** - easy to add new literals or backtrack to a previous state
- **Layered/Lazy** - able to detect simple inconsistencies quickly, able to detect difficult inconsistencies eventually
- **Equality Propagating** - If theory solvers can detect when two terms are equivalent, this greatly simplifies the search for a satisfying arrangement

Lazy SMT

Theory solvers check the satisfiability of conjunctions of literals.

What about more general Boolean structure?

What is needed is a combination of **Boolean reasoning** and **theory reasoning**.

The **eager** approach to SMT does this by encoding theory reasoning as a Boolean satisfiability problem.

Here, I will focus on the **lazy** approach in which both a Boolean engine and a theory solver work together to solve the problem [dMRS02, BDS02].
Abstract DPLL

We start with an abstract description of DPLL, the algorithm used by most SAT solvers \cite{NOT06}.

- Abstract DPLL uses states and transitions to model the progress of the algorithm.
- Most states are of the form $M \parallel F$, where
  - $M$ is a sequence of annotated literals denoting a partial truth assignment, and
  - $F$ is the CNF formula being checked, represented as a set of clauses.
- The initial state is $\emptyset \parallel F$, where $F$ is to be checked for satisfiability.
- Transitions between states are defined by a set of conditional transition rules.

Abstract DPLL Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnitProp</td>
<td>$M \parallel F, C \lor l$</td>
<td>$M l \parallel F, C \lor l$ if $l$ is undefined in $M$</td>
</tr>
<tr>
<td>PureLiteral</td>
<td>$M \parallel F$</td>
<td>$M l \parallel F$ if $l$ occurs in some clause of $F$ and $l$ is undefined in $M$</td>
</tr>
<tr>
<td>Decide</td>
<td>$M \parallel F$</td>
<td>$M l^d \parallel F$ if $l$ or $\neg l$ occurs in a clause of $F$ and $l$ is undefined in $M$</td>
</tr>
<tr>
<td>Backtrack</td>
<td>$M l^d N \parallel F, C$</td>
<td>$M \neg l \parallel F, C$ if $M l^d N \parallel \neg C$ and $N$ contains no decision literals</td>
</tr>
<tr>
<td>Fail</td>
<td>$M \parallel F, C$</td>
<td>$fail$ if $M \parallel \neg C$ and $M$ contains no decision literals</td>
</tr>
</tbody>
</table>
Example

\[
\begin{align*}
0 \mid & \quad 1 \lor \overline{2}, \quad \overline{1} \lor \overline{2}, \quad 2 \lor 3, \quad \overline{3} \lor 2, \quad 1 \lor 4 \quad \Rightarrow \quad \text{(PureLiteral)} \\
4 \mid & \quad 1 \lor \overline{2}, \quad \overline{1} \lor \overline{2}, \quad 2 \lor 3, \quad \overline{3} \lor 2, \quad 1 \lor 4 \quad \Rightarrow \quad \text{(Decide)} \\
4 \mid 1 \mid 2 \mid & \quad 1 \lor \overline{2}, \quad \overline{1} \lor \overline{2}, \quad 2 \lor 3, \quad \overline{3} \lor 2, \quad 1 \lor 4 \quad \Rightarrow \quad \text{(UnitProp)} \\
4 \mid 1 \mid 2 \mid & \quad 1 \lor \overline{2}, \quad \overline{1} \lor \overline{2}, \quad 2 \lor 3, \quad \overline{3} \lor 2, \quad 1 \lor 4 \quad \Rightarrow \quad \text{(Backtrack)} \\
4 \mid 1 \mid 2 \mid 3 & \quad 1 \lor \overline{2}, \quad \overline{1} \lor \overline{2}, \quad 2 \lor 3, \quad \overline{3} \lor 2, \quad 1 \lor 4 \quad \Rightarrow \quad \text{(Fail)} \\
\end{align*}
\]

Result: \textit{Unsatisfiable}

Roadmap

Theory Solvers
- Examples of Theory Solvers
- Combining Theory Solvers
- Extending Theory Solvers for SMT

From SAT to SMT
- Abstract DPLL
- Abstract DPLL Modulo Theories
- Key Optimizations
- Quantifier Instantiation

Abstract DPLL Modulo Theories

The \textit{Abstract DPLL Modulo Theories} framework extends the Abstract DPLL framework to include theory reasoning [NOT06].

Assume we have a theory \( T \) and a solver \( \text{Sat}_T \) that can check satisfiability of conjunctions of literals in \( T \).

Suppose we want to check the \( T \)-satisfiability of an arbitrary (quantifier-free) formula \( \phi \).

We start by converting \( \phi \) to CNF.

We can then use the Abstract DPLL rules, allowing any first-order literal where before we had propositional literals.
The Abstract DPLL Modulo Theories framework extends the Abstract DPLL framework to include theory reasoning \[\text{NOT}06\].

Assume we have a theory $T$ and a solver $\text{Sat}_T$ that can check satisfiability of conjunctions of literals in $T$.

Suppose we want to check the $T$-satisfiability of an arbitrary (quantifier-free) formula $\phi$.

We start by converting $\phi$ to CNF.

What other changes do we need to make to Abstract DPLL so it will work for SMT?

The first change is to the definition of a final state. A final state is now:

- the special fail state: $\text{fail}$, or
- $M \models F$, where $M \models F$, and $\text{Sat}_T(M)$ reports satisfiable.

What happens if we reach a state in which: $M \models F$, $M \models \neg F$, and $\text{Sat}_T(M)$ reports unsatisfiable? (call this a pseudo-final state)

We need to backtrack. The DPLL rules will take care of this automatically if we add a clause $C$ such that $M \models \neg C$.

What clause should we add? How about $\neg M$?

The justification for adding $\neg M$ is that $\models_T \neg M$.

Note that $\Gamma \models_T \phi$ denotes that $\phi$ holds whenever both $\Gamma$ and $T$ are satisfied.

We can generalize this to allow any clause $C$ to be added as long as $F \models_T C$. The following modified Learn rule allows this (we also modify the Forget rule in an analogous way):

\[
\begin{align*}
\text{Theory Learn} : & \\
M \not\models F & \implies M \not\models F, C & \text{if all atoms of } C \text{ occur in } F \\
M \not\models F, C & \implies M \not\models F & \text{if } F \models_T C
\end{align*}
\]

\[
\begin{align*}
\text{Theory Forget} : & \\
M \not\models F, C & \implies M \not\models F & \text{if } F \models_T C
\end{align*}
\]

The resulting set of rules is almost enough to correctly implement an SMT solver. We need one more change.

A somewhat surprising observation is that the pure literal rule has to be abandoned. Why?

Propositional literals are independent of each other, but first order literals may not be.

The remaining rules form a sound and complete procedure for SMT.
**Example of Lazy SMT**

\[ g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \lor g(a) \neq d \]

\[ \begin{array}{c|c}
\emptyset & 1, \top \lor 3, \top \lor 5 \\
1 & 1, \top \lor 3, \top \lor 5 \\
1 \top^d & 1, \top \lor 3, \top \lor 5 \\
1 \top^d \top^d & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
1 \top^d & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
1 \top^d & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
1 \top^d & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
1 \top^d & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
1 \top^d & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\end{array} \]

\[ \begin{array}{c|c}
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\end{array} \]

\[ \begin{array}{c|c}
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\end{array} \]

\[ \begin{array}{c|c}
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\top & 1, \top \lor 3, \top \lor 5, \top \lor 2 \lor 4 \\
\end{array} \]

fail

---

**Roadmap**

**Theory Solvers**
- Examples of Theory Solvers
- Combining Theory Solvers
- Extending Theory Solvers for SMT

**From SAT to SMT**
- Abstract DPLL
- Abstract DPLL Modulo Theories
- Key Optimizations
- Quantifier Instantiation

---

**Key Optimizations**

We will mention three ways to improve the algorithm.

- Minimizing learned clauses
- Early conflict detection
- Theory propagation

---

**Minimizing Learned Clauses**

The main problem with the approach as described so far is that learning \( \neg M \) in every pseudo-final state is very inefficient.

To see why, recall that a pseudo-final state is:
- \( M \parallel F \), where
- \( M \models F \), and
- \( \text{Sat}_T(M) = False \)

Note that \( M \) is a sequence of literals and could be quite large.

However, it is often the case that a small subset of \( M \) is sufficient to cause an inconsistency in \( T \).
Minimizing Learned Clauses

To solve the problem, whenever $Sat_T(M)$ is called, an effort must be made to find the smallest possible subset of $M$ which is inconsistent.

There are several methods:

- Brute-force minimization (typically too slow)
- Traverse a proof tree for each inconsistency (similar to traversing an implication graph in SAT solvers)
- Ad hoc per-theory techniques

Example with Minimized Learned Clauses

$g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \lor g(a) \neq d$

| 1 | 0 | 1, \overline{g} \lor 3, \overline{g} \lor 3 | (UnitProp) |
| 2 | 1 | 1, \overline{g} \lor 3, \overline{g} \lor 3 | (UnitProp) |
| 3 | 1 \overline{2} \overline{3} | 1, \overline{g} \lor 3, \overline{g} \lor 3 | (Decide) |
| 4 | 1 \overline{2} \overline{3} | 1, \overline{g} \lor 3, \overline{g} \lor 3 | (Decide) |
| 5 | 1 \overline{2} \overline{3} \overline{4} | 1, \overline{g} \lor 3, \overline{g} \lor 3, T \lor 2 | (Theory Learn) |
| 6 | 1 \overline{2} \overline{3} \overline{4} | 1, \overline{g} \lor 3, \overline{g} \lor 3, T \lor 2 | (Backjump) |
| 7 | 1 \overline{2} \overline{3} \overline{4} | 1, \overline{g} \lor 3, \overline{g} \lor 3, T \lor 2 | (Theory Learn) |
| 8 | 1 \overline{2} \overline{3} \overline{4} | 1, \overline{g} \lor 3, \overline{g} \lor 3, T \lor 2, T \lor 3 \lor 4 | (Fail) |

fail

Example with Early Conflict Detection

$g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \lor g(a) \neq d$

| 1 | 0 | 1, \overline{g} \lor 3, \overline{g} \lor 3 | (UnitProp) |
| 2 | 1 | 1, \overline{g} \lor 3, \overline{g} \lor 3 | (UnitProp) |
| 3 | 1 \overline{2} \overline{3} | 1, \overline{g} \lor 3, \overline{g} \lor 3 | (Decide) |
| 4 | 1 \overline{2} \overline{3} | 1, \overline{g} \lor 3, \overline{g} \lor 3 | (Decide) |
| 5 | 1 \overline{2} \overline{3} \overline{4} | 1, \overline{g} \lor 3, \overline{g} \lor 3, T \lor 2 | (Theory Learn) |
| 6 | 1 \overline{2} \overline{3} \overline{4} | 1, \overline{g} \lor 3, \overline{g} \lor 3, T \lor 2 | (Backjump) |
| 7 | 1 \overline{2} \overline{3} \overline{4} | 1, \overline{g} \lor 3, \overline{g} \lor 3, T \lor 2 | (Theory Learn) |
| 8 | 1 \overline{2} \overline{3} \overline{4} | 1, \overline{g} \lor 3, \overline{g} \lor 3, T \lor 2, T \lor 3 \lor 4 | (Fail) |

fail

Early Conflict Detection

So far, we have indicated that we will check $M$ for $T$-satisfiability only when a pseudo-final state is reached.

In contrast, we could check $M$ for $T$-satisfiability every time $M$ changes, possibly resulting in earlier detection of conflicts.

Experimental results show that this approach is significantly better.

It requires $Sat_T$ to be online: able quickly to determine the consistency of incrementally more literals or to backtrack to a previous state.

It also requires that the SAT solver be instrumented to call $Sat_T$ every time a variable is assigned a value.
### Theory Propagation

A final improvement is to add the following rule:

**Theory Propagate**: 

\[ M \models F \implies M \models F \text{ if } \begin{cases} M \models T \ l \\ l \text{ or } \neg l \text{ occurs in } F \\ l \text{ is undefined in } M \end{cases} \]

This rule allows \( \text{Sat}_T \) to inform the SAT solver if it is able to deduce that an unassigned literal is entailed by the current set of literals \( (M) \).

Experimental results show that this can also be very helpful in practice.

Techniques for implementing theory propagation vary by solver and by theory.

### Example with Theory Propagation

\[ g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \lor g(a) \neq d \]

\[ \emptyset \models 1, 2 \lor 3, 4 \lor 3 \implies (\text{UnitProp}) \]
\[ 1 \models 1, 2 \lor 3, 4 \lor 3 \implies (\text{Theory Propagate}) \]
\[ 1 2 \models 1, 2 \lor 3, 4 \lor 3 \implies (\text{UnitProp}) \]
\[ 1 2 3 \models 1, 2 \lor 3, 4 \lor 3 \implies (\text{Theory Propagate}) \]
\[ 1 2 3 4 \models 1, 2 \lor 3, 4 \lor 3 \implies (\text{Fail}) \]

### Roadmap

**Theory Solvers**

- Examples of Theory Solvers
- Combining Theory Solvers
- Extending Theory Solvers for SMT

**From SAT to SMT**

- Abstract DPLL
- Abstract DPLL Modulo Theories
- Key Optimizations
- Quantifier Instantiation

### Quantifiers

The Abstract DPLL Modulo Theories framework can also be extended to include rules for quantifier instantiation [GBT07].

- First, we extend the notion of literal to that of an abstract literal which may include quantified formulas in place of atomic formulas.
- Add two additional rules:

  \[
  \text{Inst}_\exists : \quad M \models F \implies M \models F, (\neg \exists x. P \lor P[x/sk]) \quad \text{if } \begin{cases} \exists x P \text{ is an abstract literal in } M \\ sk \text{ is a fresh constant.} \end{cases}
  \]

  \[
  \text{Inst}_\forall : \quad M \models F \implies M \models F, (\neg \forall x. P \lor P[x/t]) \quad \text{if } \begin{cases} \forall x P \text{ is an abstract literal in } M \\ t \text{ is a ground term.} \end{cases}
  \]
**An Example**

Suppose $a$ and $b$ are constant symbols and $f$ is an uninterpreted function symbol. We show how to prove the validity of the following formula:

$$ (0 \leq b \land (\forall x. 0 \leq x \rightarrow f(x) = a)) \rightarrow f(b) = a $$

We first negate the formula and put it into abstract CNF. The result is three unit clauses:

$$ (0 \leq b) \land (\forall x. (\neg 0 \leq x \lor f(x) = a)) \land (\neg f(b) = a) $$

**Quantifiers**

The simple technique of quantifier instantiation is remarkably effective on verification benchmarks.

The main difficulty is coming up with the right terms to instantiate.

Matching techniques pioneered by Simplify [DNS03] have recently been adopted and improved by several modern SMT solvers [FJS04, BdM07, GBT07].

**References**


References


