The new Architecture and Solvers in the \textit{Barcelogic} SMT tool

Robert Nieuwenhuis, Albert Oliveras, Enric Rodriguez-Carbonell

Barcelogic Research Group, Tech. Univ. Catalonia, Barcelona

SSPV’06
August 12th, 2006, Seattle
Overview of the talk

- DPLL and Conflict Analysis
- Satisfiability Modulo Theories (SMT)
- DPLL\((T) = DPLL(X) + T\)-Solver
- Our Barcelogic DPLL\((T)\) tool
- What does DPLL\((T)\) need from T-Solver?
- Ongoing work on T-Solvers and Combination
- Some new applications of DPLL\((T)\)
- Other ongoing work
(Abstract) DPLL for propositional SAT

<table>
<thead>
<tr>
<th>Assmt.:</th>
<th>Clause set:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
</tr>
</tbody>
</table>

$\Rightarrow$ (Decide)

Other rules:
- Backjump: (generalizes Backtrack)
- Learn: (learning backjump clauses avoids "similar" conflicts)
- Forget: (removes "inactive" clauses)
(Abstract) DPLL for propositional SAT

Assmt.: Clause set:
\[ \emptyset \] \[ \bar{1} \lor 2, \; 3 \lor 4, \; 5 \lor 6, \; 6 \lor 5 \lor 2 \] \[ \Rightarrow \] (Decide)
\[ 1 \] \[ \bar{1} \lor 2, \; 3 \lor 4, \; 5 \lor 6, \; 6 \lor 5 \lor 2 \] \[ \Rightarrow \] (UnitPropagate)
(Abstract) DPLL for propositional SAT

Assmt.: Clause set:

\[
\emptyset \quad \| \quad \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{6} \lor \overline{5} \lor 2 \quad \Rightarrow \quad \text{(Decide)}
\]

1 \quad \| \quad \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{6} \lor \overline{5} \lor 2 \quad \Rightarrow \quad \text{(UnitPropagate)}

1, 2 \quad \| \quad \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{6} \lor \overline{5} \lor 2 \quad \Rightarrow \quad \text{(Decide)}
### (Abstract) DPLL for propositional SAT

<table>
<thead>
<tr>
<th>Assmt.:</th>
<th>Clause set:</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1 2 3</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
</tbody>
</table>

Other rules:
- **Backjump**: (generalizes **Backtrack**)
- **Learn**: (learning backjump clauses avoids “similar” conflicts)
- **Forget**: (removes “inactive” clauses)
(Abstract) DPLL for propositional SAT

<table>
<thead>
<tr>
<th>Assmt.</th>
<th>Clause set:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>( \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor 5 \lor \overline{2} )</td>
<td>( \Rightarrow ) (Decide)</td>
</tr>
<tr>
<td>1</td>
<td>( \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor 5 \lor \overline{2} )</td>
<td>( \Rightarrow ) (UnitPropagate)</td>
</tr>
<tr>
<td>1 2</td>
<td>( \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor 5 \lor \overline{2} )</td>
<td>( \Rightarrow ) (Decide)</td>
</tr>
<tr>
<td>1 2 3</td>
<td>( \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor 5 \lor \overline{2} )</td>
<td>( \Rightarrow ) (UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>( \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, 6 \lor 5 \lor \overline{2} )</td>
<td>( \Rightarrow ) (Decide)</td>
</tr>
<tr>
<td>Assmt.:</td>
<td>Clause set:</td>
<td>Action:</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1 2 3</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor \overline{6}$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
</tbody>
</table>
### (Abstract) DPLL for propositional SAT

<table>
<thead>
<tr>
<th>Assmt.</th>
<th>Clause set</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1 2 3</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor \overline{5} \lor \overline{2}$</td>
<td>(Backtrack)</td>
</tr>
</tbody>
</table>
**Assmt.:**

| Clause set: |  \( 
eg 1 \lor 2, \neg 3 \lor 4, \neg 5 \lor 6, 6 \lor \neg 5 \lor 2 \) | \( \Rightarrow \) |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>(Decide)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(UnitPropagate)</td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td>(Decide)</td>
<td></td>
</tr>
<tr>
<td>1 2 3</td>
<td>(UnitPropagate)</td>
<td></td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>(Decide)</td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>(UnitPropagate)</td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>(Backtrack)</td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>( \Rightarrow )</td>
<td>. . .</td>
</tr>
</tbody>
</table>
**DPLL for propositional SAT**

<table>
<thead>
<tr>
<th>Assmt.:</th>
<th>Clause set:</th>
<th>Action:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\overline{1} \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1</td>
<td>$\overline{1} \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2</td>
<td>$\overline{1} \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1 2 3</td>
<td>$\overline{1} \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>$\overline{1} \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>$\overline{1} \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>$\overline{1} \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$</td>
<td>(Backtrack)</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>$\overline{1} \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$</td>
<td>(Backtrack)</td>
</tr>
</tbody>
</table>

Other rules:

- **Backjump**: (generalizes Backtrack)
- **Learn**: (learning backjump clauses avoids “similar” conflicts)
- **Forget**: (removes “inactive” clauses)
Backtrack vs. Backjump

Same example again. Remember: Backtrack gave 1 2 3 4 5.

But note that decision level 3 4 is unrelated to the conflict $6 \lor \overline{5} \lor \overline{2}$:

$\emptyset \ || \ \overline{1} \lor 2, \ \overline{3} \lor 4, \ \overline{5} \lor \overline{6}, \ 6 \lor \overline{5} \lor \overline{2} \ \Rightarrow \ \text{(Decide)}$

$\ldots \ || \ \ldots$

$1 \ 2 \ 3 \ 4 \ 5 \ 6 \ || \ \overline{1} \lor 2, \ \overline{3} \lor 4, \ \overline{5} \lor \overline{6}, \ 6 \lor \overline{5} \lor \overline{2} \ \Rightarrow \ \text{(Backjump)}$
Backtrack vs. Backjump

Same example again. Remember: Backtrack gave 1 2 3 4 5.

But note that decision level 3 4 is unrelated to the conflict 6 ∨ 5 ∨ 2:

\[
\emptyset \quad \parallel \quad \bar{1} \lor 2, \; \bar{3} \lor 4, \; \bar{5} \lor 6, \; 6 \lor \bar{5} \lor 2 \quad \Rightarrow \quad \text{(Decide)}
\]

\[
1 \; 2 \; 3 \; 4 \; 5 \; \bar{6} \quad \parallel \quad \ldots
\]

\[
1 \; 2 \; 3 \; 4 \; 5 \; \bar{6} \quad \parallel \quad \bar{1} \lor 2, \; \bar{3} \lor 4, \; \bar{5} \lor 6, \; 6 \lor \bar{5} \lor 2 \quad \Rightarrow \quad \text{(Backjump)}
\]

\[
1 \; 2 \; \bar{5} \quad \parallel \quad \bar{1} \lor 2, \; \bar{3} \lor 4, \; \bar{5} \lor 6, \; 6 \lor \bar{5} \lor 2 \quad \Rightarrow \quad \ldots
\]

The Backjump rule is:

\[
M \; l \; N \parallel F, \; C \quad \Rightarrow \quad M \; l' \parallel F, \; C \quad \text{IF}
\]

\[
\begin{cases}
C \text{ is false in } M \; l \; N, \text{ and} \\
\text{there is some clause } C' \lor l' \\
- \text{ entailed by } F, C \\
- \text{s.t. } C' \text{ is false in } M
\end{cases}
\]

\[
C' \lor l' \text{ is called the backjump clause. In our example, it is } \bar{2} \lor \bar{5}.
\]
Conflict analysis: find backjump clause

Consider assignment: \( \ldots 6 \ldots \overline{7} \ldots 9 \) and let \( F \) contain:

\[
\overline{9} \lor \overline{6} \lor \overline{7} \lor \overline{8}, \quad 8 \lor \overline{7} \lor \overline{5}, \quad \overline{6} \lor \overline{8} \lor \overline{4}, \quad \overline{4} \lor \overline{1}, \quad \overline{4} \lor \overline{5} \lor \overline{2}, \quad \overline{5} \lor \overline{7} \lor \overline{3}, \quad \overline{1} \lor \overline{2} \lor \overline{3}.
\]

UnitPropagate: \( \ldots 6 \ldots \overline{7} \ldots 9 \ \overline{8} \ \overline{5} \ 4 \ \overline{1} \ 2 \ \overline{3} \). Conflict with \( \overline{1} \lor \overline{2} \lor \overline{3} \)!

Implication Graph:

Can use \( 8 \lor \overline{7} \lor \overline{6} \) for Backjump to \( \ldots 6 \ldots \overline{7} \ 8 \).
Confl. analysis: find backjump clause (2)

Same example: assignment \( \ldots 6 \ldots \overline{7} \ldots 9 \) and let \( F \) contain:
\[
\overline{9} v \overline{6} v 7 v 8, \quad 8 v 7 v \overline{5}, \quad \overline{6} v 8 v 4, \quad \overline{4} v \overline{1}, \quad \overline{4} v 5 v 2, \quad 5 v 7 v \overline{3}, \quad 1 v \overline{2} v 3.
\]

UnitPropagate: \( \ldots 6 \ldots \overline{7} \ldots 9 \ \overline{8} \ \overline{5} \ \overline{4} \ \overline{1} \ \overline{2} \ \overline{3}. \) Conflict with \( 1 v \overline{2} v 3! \)

Do Resolutions in reverse order backwards from conflict:

\[
\begin{array}{c}
5 v 7 v \overline{3} \\
\overline{4} v 5 v 2 \\
\overline{4} v \overline{1} \\
\overline{6} v 8 v 4 \\
8 v 7 v \overline{5} \\
\end{array} \\
\begin{array}{c}
1 v \overline{2} v 3 \\
5 v 7 v 1 v \overline{2} \\
\overline{4} v 5 v 7 v 1 \\
5 v 7 v 4 \\
\overline{6} v 8 v 7 v 5 \\
8 v 7 v 6
\end{array}
\]

until reaching clause with only 1 lit. of current decision level.

Can use this clause \( 8 v 7 v \overline{6} \) for Backjump to \( \ldots 6 \ldots \overline{7} \ \overline{8}. \)
Abstract DPLL results

A DPLL procedure for $F$ is any derivation: $\emptyset \parallel F \Rightarrow \ldots \Rightarrow S$ where $S$ is a final state (no rule applies). It always terminates.
Abstract DPLL results

A DPLL procedure for $F$ is any derivation: $\emptyset \parallel F \Rightarrow \ldots \Rightarrow S$ where $S$ is a final state (no rule applies). It always terminates.

One can easily prove that, if the final state $S$ is:

- $\text{fail}$ then $F$ is unsat.
- of the form $M \parallel F$ then $M$ is a model
Abstract DPLL results

A DPLL procedure for $F$ is any derivation: $\emptyset \parallel F \Rightarrow \ldots \Rightarrow S$ where $S$ is a final state (no rule applies). It always terminates.

One can easily prove that, if the final state $S$ is:

- **fail** then $F$ is unsat.
- of the form $M \parallel F$ then $M$ is a model

Abstract DPLL provides formal and uniform proofs of correctness and completeness of many variants, strategies and ... extensions to e.g., SAT Modulo Theories (SMT)...

Barcelogic - Tech. Univ. Catalonia (UPC)
Overview of the talk

- DPLL and Conflict Analysis
- **Satisfiability Modulo Theories (SMT)**
- $DPLL(T) = DPLL(X) + T$-Solver
- Our Barcelogic $DPLL(T)$ tool
- What does $DPLL(T)$ need from $T$-Solver?
- Ongoing work on $T$-Solvers and Combination
- Some new applications of $DPLL(T)$
- Other ongoing work
Some problems are more naturally expressed in richer logics than just propositional logic, e.g:

- Software/Hardware verification needs reasoning about equality, arithmetic, data structures, ...

SMT consists of deciding the satisfiability of a (ground) FO formula with respect to a background theory $T$

Example: $T$ is Equality with Uninterpreted Functions (EUF):

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

Example: (combined theories)

$$A = write(B, a+1, 4) \land (read(A, b+3) = 2 \lor f(a-1) \neq f(b+1))$$

Wide range of applications
The Eager approach to SMT

- Methodology: translate problem into equisatisfiable propositional formula and use off-the-shelf SAT solver [Bryant, Velev, Pnueli, Lahiri, Seshia, Strichman, ...]

- Why “eager”? Search uses all theory information from the beginning

- Characteristics:
  + Can use best available SAT solver
  - Sophisticated encodings are needed for each theory
  - Sometimes translation and/or solving too slow

Main Challenge for alternative approaches is to combine:

- DPLL-based techniques for handling the boolean structure with
- Efficient theory solvers for conjunctions of T-literals
The Lazy approach to SMT

Same example: consider EUF and

\[
\begin{align*}
g(a) &= c \quad \land \quad \left( f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \right) \quad \land \quad c \neq d \\
&\quad \begin{array}{ll}
\text{1} \quad \text{2} \quad \text{3} \quad \text{4}
\end{array}
\end{align*}
\]

1. Send \{1, \bar{2} \lor 3, \bar{4}\} to SAT solver
The Lazy approach to SMT

Same example: consider EUF and

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

1. Send \{1, \overline{2} \lor 3, \overline{4}\} to SAT solver

SAT solver returns model [1, \overline{2}, \overline{4}]
The Lazy approach to SMT

Same example: consider EUF and

\[
\begin{align*}
g(a) = c & \quad \land \quad ( f(g(a)) \neq f(c) \lor g(a) = d ) \quad \land \quad c \neq d \\
1 & \quad \land \quad 2 & \quad \land \quad 3 & \quad \land \quad 4
\end{align*}
\]

1. Send \{ 1, 2 \lor 3, 4 \} to SAT solver

SAT solver returns model \[ 1, 2, 4 \]

Theory solver says \[ 1, 2, 4 \] is \( T \)-inconsistent
The Lazy approach to SMT

Same example: consider EUF and

\[\begin{align*}
g(a) &= c \quad \land \quad ( f(g(a)) \neq f(c) \quad \lor \quad g(a) = d ) \quad \land \quad c \neq d \\
1 \quad \land \quad 2 & \quad \land \quad 3 & \quad \land \quad 4
\end{align*}\]

1. Send \{ 1, 2 \lor 3, 4 \} to SAT solver
   
   SAT solver returns model [1, 2, 4]
   
   Theory solver says [1, 2, 4] is T-inconsistent

2. Send \{ 1, 2 \lor 3, 4, 1 \lor 2 \lor 4 \} to SAT solver
The Lazy approach to SMT

Same example: consider EUF and

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

1. Send \{ 1, 2 \lor 3, 4 \} to SAT solver

   SAT solver returns model [1, 2, 4]

   Theory solver says [1, 2, 4] is \( T \)-inconsistent

2. Send \{ 1, 2 \lor 3, 4, 1 \lor 2 \lor 4 \} to SAT solver

   SAT solver returns model [1, 2, 3, 4]
Same example: consider \( \text{EUF} \) and

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

1. Send \( \{1, \overline{2} \lor 3, \overline{4} \} \) to SAT solver
   - SAT solver returns model \([1, \overline{2}, \overline{4}]\)
   - Theory solver says \([1, \overline{2}, \overline{4}]\) is \(T\)-inconsistent

2. Send \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4\} \) to SAT solver
   - SAT solver returns model \([1, 2, 3, \overline{4}]\)
   - Theory solver says \([1, 2, 3, \overline{4}]\) is \(T\)-inconsistent
The Lazy approach to SMT

Same example: consider EUF and

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

1. Send \{ 1, \overline{2} \lor 3, \overline{4} \} to SAT solver
   SAT solver returns model [1, \overline{2}, \overline{4}]
   Theory solver says [1, \overline{2}, \overline{4}] is \( T \)-inconsistent

2. Send \{ 1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4 \} to SAT solver
   SAT solver returns model [1, 2, 3, \overline{4}]
   Theory solver says [1, 2, 3, \overline{4}] is \( T \)-inconsistent

3. Send \{ 1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4, \overline{1} \lor \overline{2} \lor 3 \lor 4 \} to SAT solver
The Lazy approach to SMT

Same example: consider EUF and

\[
\begin{align*}
g(a) &= c \\ &\quad \land (f(g(a)) \neq f(c) \lor g(a) = d) \\ &\quad \land c \neq d
\end{align*}
\]

1. Send \{1, 2\lor 3, 4\} to SAT solver

   SAT solver returns model [1, 2, 4]

   Theory solver says [1, 2, 4] is T-inconsistent

2. Send \{1, 2\lor 3, 4, 1\lor 2\lor 3\lor 4\} to SAT solver

   SAT solver returns model [1, 2, 3, 4]

   Theory solver says [1, 2, 3, 4] is T-inconsistent

3. Send \{1, 2\lor 3, 4, 1\lor 2\lor 3\lor 4, 1\lor 2\lor 3\lor 4\} to SAT solver

   SAT solver says UNSAT
Lazy approach (2)

- Why “lazy”? Theory information used lazily when checking $T$-consistency of propositional models

- Characteristics:
  + Modular and flexible
  - Theory information does not guide the search

- Tools: CVC-Lite, ICS, MathSAT, TSAT+, Verifun, ...
Optimized Lazy approach

- Check $T$-consistency only of full propositional models
Optimized Lazy approach

- Check $T$-consistency only of full propositional models
- Check $T$-consistency of partial assignment while being built

Upon a $T$-inconsistency, do conflict analysis of the explanation and backjump.
Optimized Lazy approach

- Check \( T \)-consistency only of full propositional models
- Check \( T \)-consistency of partial assignment while being built
- Given a \( T \)-inconsistent assignment \( M \), add \( \neg M \) as a clause
Optimized Lazy approach

- Check $T$-consistency only of full propositional models
- Check $T$-consistency of partial assignment while being built

- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause
- Given a $T$-inconsistent assignment $M$, find an explanation (a small $T$-inconsistent subset of $M$) and add it as a clause
Optimized Lazy approach

- Check $T$-consistency only of full propositional models
- Check $T$-consistency of partial assignment while being built

- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause
- Given a $T$-inconsistent assignment $M$, find an explanation (a small $T$-inconsistent subset of $M$) and add it as a clause

- Upon a $T$-inconsistency, add clause and restart
Optimized Lazy approach

- Check $T$-consistency only of full propositional models
- Check $T$-consistency of partial assignment while being built

- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause
- Given a $T$-inconsistent assignment $M$, find an explanation (a small $T$-inconsistent subset of $M$) and add it as a clause

- Upon a $T$-inconsistency, add clause and restart
- Upon a $T$-inconsistency, do conflict analysis of the explanation and Backjump
Overview of the talk

- DPLL and Conflict Analysis
- Satisfiability Modulo Theories (SMT)
- \( \text{DPLL}(T) = \text{DPLL}(X) + T\text{-Solver} \)
- Our Barcelogic DPLL\((T)\) tool
- What does DPLL\((T)\) need from T-Solver?
- Ongoing work on T-Solvers and Combination
- Some new applications of DPLL\((T)\)
- Other ongoing work
Our DPLL($T$) approach

\[ \text{DPLL}(T) = \text{DPLL}(X) \text{ engine} + T\text{-Solver} \]

- **Modular and flexible**, as CLP($X$) in Constraint Logic Progr.: can plug in any $T$-Solver into the DPLL($X$) engine.

- **Theory Propagation**: more pruning in optimized lazy SMT
  \[ \text{T-Propagate: } M \parallel F \implies M l \parallel F \text{ IF } \left\{ \begin{array}{l} M \models_T l \end{array} \right\} \]

- $T$-Solver also guides search, instead of only validating it

- [Armando et al]: Add $\neg l$. If $T$-inconsistent then infer $l$.
  But in DPLL($T$):
  - $T$-Solvers specialized and fast in Theory Propagation
  - Fully exploited in conflict analysis (non-trivial)
    Not any explanation of a theory propagation is ok!
DPLL($T$) Example

Notation used: Abstract DPLL Modulo Theories.

Consider again same example with EUF:

\[
\begin{align*}
g(a) &= c \quad \land \quad \begin{cases}
1 & \quad ( f(g(a)) \neq f(c) \quad \lor \quad g(a) = d ) \\
2 & \quad \land \quad c \neq d
\end{cases} \\
\emptyset & \quad \lor \quad 1, \overline{2} \lor 3, \overline{4} \quad \Rightarrow \quad \text{(UnitPropagate)}
\end{align*}
\]
DPLL($T$) Example

Notation used: Abstract DPLL Modulo Theories.

Consider again same example with EUF:

\[
\begin{align*}
g(a) &= c \\
\left[ \begin{array}{c}
1 \\
2 \\
3 \\
4
\end{array} \right] &\quad \land \quad \left( \begin{array}{c}
f(g(a)) \neq f(c) \\
g(a) = d
\end{array} \right) \\
&\quad \land \quad c \neq d
\end{align*}
\]

\[
\emptyset \quad || \quad 1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad \text{(UnitPropagate)}
\]

\[
1 \quad || \quad 1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad \text{(T-Propagate)}
\]
DPLL($T$) Example

Notation used: Abstract DPLL Modulo Theories.

Consider again same example with EUF:

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

\[
\begin{align*}
\varnothing & \quad || \quad 1, \ 2 \lor 3, \ 4 \quad \Rightarrow \quad (\text{UnitPropagate}) \\
1 & \quad || \quad 1, \ 2 \lor 3, \ 4 \quad \Rightarrow \quad (\text{T-Propagate}) \\
1 \ 2 & \quad || \quad 1, \ 2 \lor 3, \ 4 \quad \Rightarrow \quad (\text{UnitPropagate})
\end{align*}
\]
DPLL\(^T\) Example

Notation used: Abstract DPLL Modulo Theories.

Consider again same example with EUF:

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

\[
\begin{align*}
\emptyset & \quad || \quad 1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad \text{(UnitPropagate)} \\
1 & \quad || \quad 1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad \text{(T-Propagate)} \\
1 \ 2 & \quad || \quad 1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad \text{(UnitPropagate)} \\
1 \ 2 \ 3 & \quad || \quad 1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad \text{(T-Propagate)}
\end{align*}
\]
Notation used: Abstract DPLL Modulo Theories.

Consider again same example with EUF:

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

\[
\begin{align*}
1 & \quad \lor \quad 2 \lor 3, \quad 4 & \Rightarrow & \quad \text{(UnitPropagate)} \\
1 2 & \quad \lor \quad 1, \quad 2 \lor 3, \quad 4 & \Rightarrow & \quad \text{(T-Propagate)} \\
1 2 3 & \quad \lor \quad 1, \quad 2 \lor 3, \quad 4 & \Rightarrow & \quad \text{(UnitPropagate)} \\
1 2 3 4 & \quad \lor \quad 1, \quad 2 \lor 3, \quad 4 & \Rightarrow & \quad \text{fail}
\end{align*}
\]
DPLL($T$) Example

Notation used: Abstract DPLL Modulo Theories.

Consider again same example with EUF:

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

1 \quad \land \quad 2 \quad 3 \quad 4

\[
\emptyset \quad || \quad 1, \ 2 \lor \ 3, \ 4 \quad \Rightarrow \quad \text{(UnitPropagate)}
\]

1 \quad || \quad 1, \ 2 \lor \ 3, \ 4 \quad \Rightarrow \quad \text{(T-Propagate)}

1 \ 2 \quad || \quad 1, \ 2 \lor \ 3, \ 4 \quad \Rightarrow \quad \text{(UnitPropagate)}

1 \ 2 \ 3 \quad || \quad 1, \ 2 \lor \ 3, \ 4 \quad \Rightarrow \quad \text{(T-Propagate)}

1 \ 2 \ 3 \ 4 \quad || \quad 1, \ 2 \lor \ 3, \ 4 \quad \Rightarrow \quad \text{fail}

No search in this example
Conflict analysis in DPLL(\(T\))

New kind of arrows (reasons) in implication graph. Each literal \(lit\) is in the partial assignment due to one of:

- **Decide** (no arrow)
- **UnitPropagate** with clause \(C\): resolve with \(C\)
- **T-Propagate**: resolve with (small) explanation
  \[l_1 \land \ldots \land l_n \rightarrow lit\]
  provided by \(T\)-Solver

Too new \(T\)-explanations are forbidden!

How should it be implemented?

- **UnitPropagate**: store a pointer to clause \(C\), as in SAT solvers
- **T-Propagate**: (pre-)compute explanations at each **T-Propagate**?
  - If possible, only on demand, during conflict analysis
  - typically only one Explain for every 250 **T-Propagate**.
  - depends on \(T\)
Our *Barcelogic DPLL*(\(T\)) tool

- **DPLL(\(X\))** is a state-of-the-art SAT engine:
  - **features à la Chaff**: two watched literals, 1UIP learning, VSIDS-like decision heuristics, ...
  - **new features**: binary clause reasoning, subsumption, lemma simplification, ...
  - see SAT Race 2006

- **\(T\)–**Solvers** for**:
  - Real/Integer Difference Logic (IDL/RDL):
  - Equality with Uninterpreted Functions (EUF)
  - Linear Real Arithmetic (LRA)
  - Linear Integer Arithmetic (LIA) (forthcoming)
  - Arrays
### Barcelogic at SMT-COMP’05

Participated in 4 (of 7) divisions:

<table>
<thead>
<tr>
<th>Division</th>
<th>Top-3 Systems</th>
<th># Pbs Solv.</th>
<th>Time (secs.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EUF (50 pbs.):</td>
<td>Barcelogic</td>
<td>39</td>
<td>8358</td>
</tr>
<tr>
<td></td>
<td>Yices</td>
<td>37</td>
<td>9601</td>
</tr>
<tr>
<td></td>
<td>MathSAT</td>
<td>33</td>
<td>12386</td>
</tr>
<tr>
<td>RDL (50):</td>
<td>Barcelogic</td>
<td>41</td>
<td>6341</td>
</tr>
<tr>
<td></td>
<td>Yices</td>
<td>37</td>
<td>9668</td>
</tr>
<tr>
<td></td>
<td>MathSAT</td>
<td>37</td>
<td>10408</td>
</tr>
<tr>
<td>IDL (51):</td>
<td>Barcelogic</td>
<td>47</td>
<td>3531</td>
</tr>
<tr>
<td></td>
<td>Yices</td>
<td>47</td>
<td>4283</td>
</tr>
<tr>
<td></td>
<td>MathSAT</td>
<td>46</td>
<td>4295</td>
</tr>
<tr>
<td>UFIDL (49):</td>
<td>Barcelogic</td>
<td>45</td>
<td>2705</td>
</tr>
<tr>
<td></td>
<td>Yices</td>
<td>36</td>
<td>9789</td>
</tr>
<tr>
<td></td>
<td>MathSAT</td>
<td>22</td>
<td>17255</td>
</tr>
</tbody>
</table>

Other tools:
- CVC-Lite (Barrett)
- Ario (Sakallah)
- Sateen (Somenzi)
- ...

Timeout = 600s.
Overview of the talk

- DPLL and Conflict Analysis
- Satisfiability Modulo Theories (SMT)
- \( \text{DPLL}(T) = \text{DPLL}(X) + T\text{-Solver} \)
- Our Barcelogic DPLL(\(T\)) tool
- What does DPLL(\(T\)) need from T-Solver?
- Ongoing work on T-Solvers and Combination
- Some new applications of DPLL(\(T\))
- Other ongoing work
What does DPLL($T$) need from $T$-Solver?

- $T$-consistency check of a set of literals $M$, with:
  - Explain of $T$-inconsistency: find (small) $T$-inconsistent subset of $M$ [minimal wrt. size?, wrt. $\subseteq$?]
  - Incrementality: if $l$ is added to $M$, check for $M \cup l$ faster than reprocessing $M \cup l$ from scratch.

- Theory propagation: find input $T$-consequences of $M$, with:
  - Explain $T$-Propagate of $l$: find (small) subset of $M$ that $T$-entails $l$ (needed in conflict analysis).

- Backtrack $n$: undo last $n$ literals added
A standard Difference Logic solver

- Given $M = \{a-b \leq 2, \ b-c \leq 3, \ c-a \leq -7\}$, construct weighted graph $G(M)$

  ![Graph Diagram]

- $M$ is $T$-inconsistent iff $G(M)$ has a negative cycle
- Bellmann-Ford-like algorithms to find such cycles in $O(nm)$
- Irredundant inconsistent subsets are negative cycles
- What about theory propagation?
Our CAV’05 DL Solver

Key idea: exhaustive theory propagation avoids consistency checks: $M \models T$-inconsistent iff $M \models_T \neg l$. Hence we would have added $\neg l$ right after $M$.

For detecting all consequences of a new literal $a - b \leq k$:

$c - d \leq k_1$ is $T$-entailed iff there is a path from $c$ to $d$ with length at most $k_1$. Hence $T$-Solver checks all shortest paths

\[ c \xrightarrow{k'} * a \xrightarrow{k} b \xrightarrow{k''} * d \]

and finds all input literals entailed by $c - d \leq k' + k + k''$

Complexity: $O(nm + N)$, being $N$ the number of input literals

Irredundant explanations for $c - d \leq k$ given by the shortest path from $c$ to $d$
Analyzing our CAV’05 solver

CHARACTERISTICS:

- TheoryProp is invoked even if UnitProp still applicable
- Cannot get rid of the exhaustiveness requirement if TheoryProp is too expensive

IDEAL SITUATION:

- Cheaper reasoning should be done first:
  1. Apply UnitProp exhaustively
  2. If no conflict, then check $T$-consistency of model
  3. If model $T$-consistent apply TheoryProp (if wanted)

- Some of the computations of the consistency check should be reused in TheoryProp
CHECK CONSISTENCY:

- Check $T$-consistency of model using Bellman-Ford-like algorithm (each newly added literal in $O(m + n \log n)$)

- Gives potential function $\pi$ s.t. for each each edge $a \xrightarrow{k} b$ we have $\pi(a) + k - \pi(b) \geq 0$

THEORY PROPAGATION:

- Addition of $a \xrightarrow{k} b$ entails $c - d \leq k'$ only if

- Shortest path computation more efficient using reduced costs, since they are non-negative
Traditionally simplex method preferred over Fourier-Motzkin elimination because:

- It is efficient in practice
- Less memory, also for Incrementality and backtracking
Linear Arithmetic Solver: Ongoing work

Traditionally simplex method preferred over Fourier-Motzkin elimination because:
- It is efficient in practice
- Less memory, also for Incrementality and backtracking

Most solvers implement the tableau simplex method:
- Pivoting is expensive as it requires to update all coefficients of the linear program
Linear Arithmetic Solver: Ongoing work

Traditionally simplex method preferred over Fourier-Motzkin elimination because:
- It is efficient in practice
- Less memory, also for Incrementality and backtracking

Most solvers implement the tableau simplex method:
- Pivoting is expensive as it requires to update all coefficients of the linear program

Our alternative: revised simplex method
- Pivoting is cheap as it just needs to incrementally update the inverse matrix corresponding to dependent variables.
- Method of choice for LP community (if sparse)
Typical structure of benchmarks:

1. 40-80% of atoms are bounds of the form $\pm x \leq k$
2. 80-90% of atoms belong to difference logic
Benchmark-goaled Linear Arithmetic

Typical structure of benchmarks:
1. 40-80 % of atoms are bounds of the form $\pm x \leq k$
2. 80-90 % of atoms belong to difference logic

Application to consistency checks:
1. Bounded simplex method
2. Lagrangian relaxation
Typical structure of benchmarks:
1. 40-80% of atoms are bounds of the form $\pm x \leq k$
2. 80-90% of atoms belong to difference logic

Application to consistency checks:
1. Bounded simplex method
2. Lagrangian relaxation

Application to theory propagation:
1. Propagate bounds of the model

$$x \leq 1 \land 2x + y \leq 1 \land x - 2y \leq 3 \implies x \leq 3$$
Benchmark-goaled Linear Arithmetic

Typical **structure of benchmarks:**
1. 40-80 % of atoms are **bounds** of the form \( \pm x \leq k \)
2. 80-90 % of atoms belong to **difference logic**

**Application to consistency checks:**
1. Bounded simplex method
2. Lagrangian relaxation

**Application to theory propagation:**
1. Propagate **bounds** of the model
   \[
   x \leq 1 \land 2x + y \leq 1 \land x - 2y \leq 3 \implies x \leq 3
   \]
2. Propagate **difference logic** fragment of the model
   \[
   x - y \leq 1 \land y - z \leq 2 \land y = x + 2z \implies x - z \leq 3
   \]
New ideas to be added soon

[Dutertre&DeMoura, CAV’06]:

- Very nice simple ideas, extremely good results
- Initial translation into equalities + bounds. E.g., replace $2x - 3y + 5z \leq 12$ by $2x - 3y + 5z = s$ and $s \leq 12$
- The equalities never change, atoms sent to (and retracted from) T-Solver are bounds.
- Allows for initial simplifications
- Little work on backtracking
- Can identify cheap T-Propagate cases
Expensive Theories, Combination

Splitting on demand [Barrett N O Tinelli, LPAR’06]:

- Some $T$-Solvers need internal case splits (non-convex $T$)
- Idea: $T$-Solver must request $\text{DPLL}(X)$ engine to do them.

Advantages:

- $\text{DPLL}(X)$ is much better in doing case splits
- Centralized decision heuristic not disturbed by other ones
- $T$-Solver simpler: no splitting infrastructure needed
- Weaker requirements for $T$-Solver:
  complete “if all demanded splits have been done”

Resulting architecture naturally includes an efficient $\text{DPLL}(T_1 \ldots T_n)$ Nelson-Oppen-based combination
Overview of the talk

- DPLL and Conflict Analysis
- Satisfiability Modulo Theories (SMT)
- \( \text{DPLL}(T) = \text{DPLL}(X) + T\text{-Solver} \)
- Our Barcelogic DPLL\((T)\) tool
- What does DPLL\((T)\) need from T-Solver?
- Ongoing work on T-Solvers and Combination
- Some new applications of DPLL\((T)\)
- Other ongoing work
Application to Predicate Abstraction

Predicate Abstraction:
- gives finite-state abstractions from infinite-state systems
- abstraction efficiently analyzed using Boolean techniques
- many applications to verification

Key operation:
- **INPUT:** a formula $\varphi$ and set of predicates $P$
- **OUTPUT:** the most precise approximation of $\varphi$ using $P$, either
  - $F_P(\varphi)$: weakest formula over $P$ $T$-entailing $\varphi$ or
  - $G_P(\varphi)$: strongest formula over $P$ $T$-entailed by $\varphi$.

Use of Barcelogic: See CAV’06 for details!
- Use All-SAT SMT + BDD to get all models over $P$ of $\varphi$
- Extract (compact) approximation from BDD
Experimental results for P. Abstraction

**Microsoft** SLAM (device drivers verification):

- Initially, **ZAP** [Ball et al, CAV’04] was used for p. abstraction
- Specialized **Symbolic Decision Procedures (SDPs)** [Lahiri et al, CAV’05] obtained **100x speedup** factor over ZAP
- **Barcelogic** gives **another 100x speedup** over SDPs

**Hardware and protocol verification problems**
(70pbs, over ≈ 25 preds) [Lahiri and Bryant, CAV’04]:

- **Barcelogic** gives 25x – 100x speedup over UCLID

**Benchmarks from the verification of programs with linked lists**
(30pbs, ≈ 20 preds) [Qaader and Lahiri, POPL’06]:

- **Barcelogic** gives 30x – 40x speedup over UCLID
Application to optimization problems

- **Aim:** find SAT/SMT models $M$ with *minimal* $\text{cost}(M)$.

- **Branch and bound in Barcelogic:** See SAT’06 for details!
  - Theory $T = \text{function } \text{cost} \land \text{best } M \text{ so far.}$
    - After each new solution, $T$ is strengthened

- **(Weighted) Max-SAT:**
  - $\text{cost}(M) = \text{sum of weights of clauses that are false in } M$
  - *Specialized rules*, e.g: if units $l$ and $\neg l$ detected, add smallest of their weights to cost

- **Barcelogic** improves best Weighted CSP/PB solvers on most larger problems

- **Max-SMT:** Modeled and solved well-known hard Radio Freq. Assignment Problems with *distance constraints*: Diff. Logic.
  - *Barcelogic* with no specialized heuristics beats best Weighted CSP solver (with its best heuristic)
Other ongoing/future work

- Bit vector arithmetic
- Adding support for quantifiers
- Efficient interpolation modulo $T$
- Other less-standard applications of SMT: e.g., CSP’s, FO finite model finding, ...

Thank you!