SAT-based techniques for integer linear constraints

GCAI 2015 (invited talk)

Robert Nieuwenhuis

Barcelogic.com - Computer Science Department
BarcelonaTech (UPC)
Thanks for inviting me, bringing me back to this wonderful country!
### Between SAT and ILP

<table>
<thead>
<tr>
<th></th>
<th>0-1 solutions</th>
<th>(\mathbb{Z}) solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>feasibility</td>
<td>optimizing</td>
</tr>
<tr>
<td>clauses</td>
<td>SAT</td>
<td></td>
</tr>
<tr>
<td>cardinality constr.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear constraints</td>
<td></td>
<td>ILP</td>
</tr>
<tr>
<td>linear constraints</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Between SAT and ILP

<table>
<thead>
<tr>
<th></th>
<th>0-1 sols</th>
<th>( \mathbb{Z} ) sols</th>
<th>( \mathbb{Q}/\mathbb{Z} ) sols</th>
</tr>
</thead>
<tbody>
<tr>
<td>clauses</td>
<td>feas.</td>
<td>opt.</td>
<td>feas.</td>
</tr>
<tr>
<td>cardinality constr.</td>
<td>SAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear constraints</td>
<td></td>
<td>ILP</td>
<td>MIPs</td>
</tr>
</tbody>
</table>
Outline of this talk

- SAT and ILP
Outline of this talk

- SAT and ILP
- Commercial ILP tools
Outline of this talk

- SAT and ILP
- Commercial ILP tools
- Between SAT and ILP
Outline of this talk

- SAT and ILP
- Commercial ILP tools
- Between SAT and ILP
- CDCL SAT solvers. Why do they work so well?
Outline of this talk

- SAT and ILP
- Commercial ILP tools
- Between SAT and ILP
- CDCL SAT solvers. Why do they work so well?
- What is SMT? Why does it work so well?
Outline of this talk

- SAT and ILP
- Commercial ILP tools
- Between SAT and ILP
- CDCL SAT solvers. Why do they work so well?
- What is SMT? Why does it work so well?
- ILP as an SMT problem. Hybrids: SMT + bottleneck encodings

GCAI'15  SAT-based techniques for integer linear constraints
Outline of this talk

- SAT and ILP
- Commercial ILP tools
- Between SAT and ILP
- CDCL SAT solvers. Why do they work so well?
- What is SMT? Why does it work so well?
- ILP as an SMT problem. Hybrids: SMT + bottleneck encodings
- Going beyond: Constraint Learning. (It can beat clause learning!)
Outline of this talk

- SAT and ILP
- Commercial ILP tools
- Between SAT and ILP
- CDCL SAT solvers. Why do they work so well?
- What is SMT? Why does it work so well?
- ILP as an SMT problem. Hybrids: SMT + bottleneck encodings
- Going beyond: Constraint Learning. (It can beat clause learning!)
- Solving the rounding problem, 0-1 case, \( \mathbb{Z} \) case
Outline of this talk

- SAT and ILP
- Commercial ILP tools
- Between SAT and ILP
- CDCL SAT solvers. Why do they work so well?
- What is SMT? Why does it work so well?
- ILP as an SMT problem. Hybrids: SMT + bottleneck encodings
- Going beyond: Constraint Learning. (It can beat clause learning!)
- Solving the rounding problem, 0-1 case, \( \mathbb{Z} \) case
- Cutsat and IntSat. Evaluation. Demo (if time).
Outline of this talk

- SAT and ILP
- Commercial ILP tools
- Between SAT and ILP
- CDCL SAT solvers. Why do they work so well?
- What is SMT? Why does it work so well?
- ILP as an SMT problem. Hybrids: SMT + bottleneck encodings
- Going beyond: Constraint Learning. (It can beat clause learning!)
- Solving the rounding problem, 0-1 case, $\mathbb{Z}$ case
- Cutsat and IntSat. Evaluation. Demo (if time).
- Simple completeness proofs for cutting planes
Outline of this talk

- SAT and ILP
- Commercial ILP tools
- Between SAT and ILP
- CDCL SAT solvers. Why do they work so well?
- What is SMT? Why does it work so well?
- ILP as an SMT problem. Hybrids: SMT + bottleneck encodings
- Going beyond: Constraint Learning. (It can beat clause learning!)
- Solving the rounding problem, 0-1 case, \( \mathbb{Z} \) case
- Cutsat and IntSat. Evaluation. Demo (if time).
- Simple completeness proofs for cutting planes
- Remarks on proof systems
Integer Linear Programming (ILP)

Find solution $\text{Sol}: \{x_1 \ldots x_n\} \rightarrow \mathbb{Z}$ to:

Minimize: $c_1 x_1 + \ldots + c_n x_n$  
(or maximize)

Subject To: $c_{11} x_1 + \ldots + c_{1n} x_n \leq c_{10}$  
$\ldots$  
$\ldots$  
$c_{m1} x_1 + \ldots + c_{mn} x_n \leq c_{m0}$  

where all coefficients $c_i$ in $\mathbb{Z}$.

**SAT**: particular case of ILP with 0-1 vars and constraint clauses:

$x \lor \bar{y} \lor \bar{z} \equiv x + (1 - y) + (1 - z) \geq 1$
• Commercial OR solvers, large, quite expensive.

• ILP based on LP relaxation + Simplex + branch-and-cut + combining a large variety of techniques: problem-specific cuts, specialized heuristics, presolving...

• Extremely mature technology. Bixby:
  “From 1991 to 2012, saw 475,000 $\times$ algorithmic speedup $\times$ 2,000 $\times$ hardware speedup.”
Between SAT and ILP

<table>
<thead>
<tr>
<th></th>
<th>0-1 solutions</th>
<th>(\mathbb{Z}) solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>feasibility</td>
<td>optimizing</td>
</tr>
<tr>
<td>clauses</td>
<td>SAT</td>
<td></td>
</tr>
<tr>
<td>cardinality constr.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear constr.</td>
<td>0-1 ILP(P-B)</td>
<td>0-1 ILP (P-B)</td>
</tr>
</tbody>
</table>

Cardinality constraints:

\[ x_1 + \ldots + x_n \leq k \quad \text{(or with } \geq, =, <, >) \]
SAT and CDCL-based SAT Solvers

SAT = particular case of ILP: vars are 0-1, constraints are clauses
CDCL = Conflict-Driven Clause-Learning backtracking algorithm

Candidate Solution:

\begin{itemize}
  \item Four clauses:
    \begin{enumerate}
      \item \(1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2\)
    \end{enumerate}
\end{itemize}

\(\Rightarrow\) (Decide)

1 2

\(1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2\)

\(\Rightarrow\) (UnitPropagate)

1 2 3

\(1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2\)

\(\Rightarrow\) (Decide)

1 2 3 4

\(1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2\)

\(\Rightarrow\) (UnitPropagate)

1 2 3 4 5

\(1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2\)

\(\Rightarrow\) (Backtrack)

1 2 3 4

\(1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2\)

solution found!
SAT and CDCL-based SAT Solvers

SAT = particular case of ILP: vars are 0-1, constraints are clauses
CDCL = Conflict-Driven Clause-Learning backtracking algorithm

**Four clauses:**

\[ \overline{1} \vee 2, \; \overline{3} \vee 4, \; 5 \vee \overline{6}, \; 6 \vee \overline{5} \vee 2 \]
SAT and CDCL-based SAT Solvers

SAT = particular case of ILP: vars are 0-1, constraints are clauses
CDCL = Conflict-Driven Clause-Learning backtracking algorithm

Candidate Solution: Four clauses:

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

1

\[ \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2} \Rightarrow \]
SAT and CDCL-based SAT Solvers

SAT = particular case of ILP: vars are 0-1, constraints are clauses
CDCL = Conflict-Driven Clause-Learning backtracking algorithm

Candidate Solution: Four clauses:

\[
\begin{align*}
\overline{1} \lor 2, & \quad 3 \lor 4, \quad 5 \lor \overline{6}, \quad 6 \lor 5 \lor \overline{2} \quad \Rightarrow \quad \text{(Decide)} \\
1 \lor \overline{2}, & \quad 3 \lor 4, \quad 5 \lor \overline{6}, \quad 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(UnitPropagate)} \\
1 \lor 2 & \quad \overline{1} \lor 2, \quad 3 \lor 4, \quad 5 \lor \overline{6}, \quad 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow
\end{align*}
\]
SAT = particular case of ILP: vars are 0-1, constraints are clauses
CDCL = Conflict-Driven Clause-Learning backtracking algorithm

Candidate Solution: Four clauses:

1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2 ⇒ (Decide)
1
1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2 ⇒ (UnitPropagate)
1 2
1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2 ⇒ (Decide)
1 2 3
1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2 ⇒

Can do much better! Next: Backjump instead of Backtrack...
SAT and CDCL-based SAT Solvers

SAT = particular case of ILP: vars are 0-1, constraints are clauses
CDCL = Conflict-Driven Clause-Learning backtracking algorithm

Candidate Solution: Four clauses:

1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \quad \Rightarrow \quad (Decide)
1 \quad \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \quad \Rightarrow \quad (UnitPropagate)
1 2 \quad \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \quad \Rightarrow \quad (Decide)
1 2 3 \quad \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \quad \Rightarrow \quad (UnitPropagate)
1 2 3 4 \quad \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \quad \Rightarrow
**SAT and CDCL-based SAT Solvers**

SAT = particular case of ILP: vars are 0-1, constraints are clauses

CDCL = Conflict-Driven Clause-Learning backtracking algorithm

**Candidate Solution:**

<table>
<thead>
<tr>
<th>Variable Set</th>
<th>Clause Set</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</td>
<td>Decide</td>
</tr>
<tr>
<td>1 2 3</td>
<td>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</td>
<td>UnitPropagate</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</td>
<td>Decide</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</td>
<td>UnitPropagate</td>
</tr>
</tbody>
</table>

Next: Backjump instead of Backtrack...
SAT and CDCL-based SAT Solvers

SAT = particular case of ILP: vars are 0-1, constraints are clauses
CDCL = Conflict-Driven Clause-Learning backtracking algorithm

**Candidate Solution:**

<table>
<thead>
<tr>
<th></th>
<th>Four clauses:</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>1 2 3 4 5</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 5 6</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $\overline{5} \lor 6$, $6 \lor 5 \lor \overline{2}$</td>
<td></td>
</tr>
</tbody>
</table>
SAT and CDCL-based SAT Solvers

SAT = particular case of ILP: vars are 0-1, constraints are clauses
CDCL = Conflict-Driven Clause-Learning backtracking algorithm

**Candidate Solution:**

<table>
<thead>
<tr>
<th>Four clauses:</th>
<th>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</th>
<th>(Decide)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2</td>
<td>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1 2 3</td>
<td>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</td>
<td>(Decide)</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</td>
<td>(UnitPropagate)</td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2</td>
<td>CONFLICT!</td>
</tr>
</tbody>
</table>
SAT and CDCL-based SAT Solvers

SAT = particular case of ILP: vars are 0-1, constraints are clauses
CDCL = Conflict-Driven Clause-Learning backtracking algorithm

Candidate Solution: Four clauses:

1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \Rightarrow (Decide)

1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \Rightarrow (UnitPropagate)

1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \Rightarrow (Decide)

1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \Rightarrow (UnitPropagate)

1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \Rightarrow (Decide)

1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \Rightarrow (UnitPropagate)

1 \lor 2, 3 \lor 4, 5 \lor 6, 6 \lor 5 \lor 2 \Rightarrow (Backtrack)
SAT and CDCL-based SAT Solvers

SAT = particular case of ILP: vars are 0-1, constraints are clauses

CDCL = Conflict-Driven Clause-Learning backtracking algorithm

Candidate Solution: Four clauses:

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

1 \lor 2 \Rightarrow \text{(UnitPropagate)}

1 2 \lor 4 \Rightarrow \text{(Decide)}

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

1 2 3 4 \lor \overline{5} \Rightarrow \text{(Decide)}

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

1 2 3 4 5 \lor \overline{5} \Rightarrow \text{(Backtrack)}
SAT and CDCL-based SAT Solvers

SAT = particular case of ILP: vars are 0-1, constraints are clauses

CDCL = Conflict-Driven Clause-Learning backtracking algorithm

Candidate Solution: Four clauses:

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

1
\[ \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2} \Rightarrow (\text{UnitPropagate}) \]
1 2
\[ \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2} \Rightarrow (\text{Decide}) \]
1 2 3
\[ \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2} \Rightarrow (\text{UnitPropagate}) \]
1 2 3 4
\[ \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2} \Rightarrow (\text{Decide}) \]
1 2 3 4 5
\[ \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2} \Rightarrow (\text{UnitPropagate}) \]
1 2 3 4 5 6
\[ \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2} \Rightarrow (\text{Backtrack}) \]
1 2 3 4 5
\[ \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6}, 6 \lor \overline{5} \lor \overline{2} \] solution found!
SAT and CDCL-based SAT Solvers

SAT = particular case of ILP: vars are 0-1, constraints are clauses
CDCL = Conflict-Driven Clause-Learning backtracking algorithm

<table>
<thead>
<tr>
<th>Candidate Solution:</th>
<th>Four clauses:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$1 \lor 2$, $3 \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$</td>
<td>$\Rightarrow$ (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>$\Rightarrow$ (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>$\Rightarrow$ (Decide)</td>
</tr>
<tr>
<td>1 2 3</td>
<td>$\overline{1} \lor 2$, $\overline{3} \lor 4$, $5 \lor 6$, $6 \lor 5 \lor 2$</td>
<td>$\Rightarrow$ (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>$\Rightarrow$ (Decide)</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>$\overline{1} \lor 2$, $3 \lor 4$, $\overline{5} \lor 6$, $6 \lor 5 \lor 2$</td>
<td>$\Rightarrow$ (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>$\Rightarrow$ (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>solution found!</td>
</tr>
</tbody>
</table>

Can do much better! Next: Backjump instead of Backtrack...
Backtrack vs. Backjump

**Same example.** Remember: Backtrack gave $1 \ 2 \ 3 \ 4 \ 5$.

But: **decision level $3 \ 4$** is irrelevant for the conflict $6 \lor 5 \lor 2$:

\[
\begin{align*}
0 & \quad \overline{1} \lor 2, \ \overline{3} \lor 4, \ \overline{5} \lor \overline{6}, \ 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(Decide)} \\
\vdots & \quad \vdots \quad \vdots \quad \vdots \\
1 \ 2 \ 3 \ 4 \ 5 \ 6 & \quad \overline{1} \lor 2, \ \overline{3} \lor 4, \ \overline{5} \lor \overline{6}, \ 6 \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(Backjump)}
\end{align*}
\]
Backtrack vs. Backjump

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

But: decision level 3 4 is irrelevant for the conflict 6 ∨ 5 ∨ 2:

0     1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2 ⇒ (Decide)


1 2 3 4 5 6     1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2 ⇒ (Backjump)

1 2 5     1 ∨ 2, 3 ∨ 4, 5 ∨ 6, 6 ∨ 5 ∨ 2 ⇒ …
Backtrack vs. Backjump

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

But: decision level 3 4 is irrelevant for the conflict $6 \lor \overline{5} \lor \overline{2}$:

$$
\begin{align*}
0 & \quad \overline{1} \lor 2, \; \overline{3} \lor 4, \; \overline{5} \lor 6, \; \overline{6} \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(Decide)} \\
\vdots & \quad \vdots \\
1 2 3 4 5 6 & \quad \overline{1} \lor 2, \; \overline{3} \lor 4, \; \overline{5} \lor 6, \; \overline{6} \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \text{(Backjump)} \\
1 2 \overline{5} & \quad \overline{1} \lor 2, \; \overline{3} \lor 4, \; \overline{5} \lor 6, \; \overline{6} \lor \overline{5} \lor \overline{2} \quad \Rightarrow \quad \ldots
\end{align*}
$$

Backjump =

1. **Conflict Analysis**: “Find” a backjump clause $C \lor l$ (here, $\overline{2} \lor \overline{5}$)
   - that is a logical consequence of the clause set
   - that reveals a unit propagation of $l$ at an earlier decision level $d$ (i.e., where its part $C$ is false)

2. Return to decision level $d$ and do the propagation.
Conflict Analysis: find backjump clause

Example. Consider stack: \( \ldots 6 \ldots 7 \ldots 9 \) and clauses:
\[
9 \lor 6 \lor 7 \lor 8, \quad 8 \lor 7 \lor 5, \quad 6 \lor 8 \lor 4, \quad 4 \lor 1, \quad 4 \lor 5 \lor 2, \quad 5 \lor 7 \lor 3, \quad 1 \lor 2 \lor 3
\]
UnitPropagate gives \( \ldots 6 \ldots 7 \ldots 9 \ 8 \ 5 \ 4 \ 1 \ 2 \ 3 \). Conflict w/ \( 1 \lor 2 \lor 3 \)!

C.An. = do resolutions with reason clauses backwards from conflict:
\[
\begin{array}{c}
5 \lor 7 \lor 3 \\
4 \lor 2 \lor 3 \\
5 \lor 7 \lor 1 \lor 2 \\
4 \lor 1 \\
5 \lor 7 \lor 1 \\
6 \lor 5 \lor 7 \lor 1 \\
8 \lor 7 \lor 5 \\
6 \lor 8 \lor 7 \lor 5 \\
8 \lor 7 \lor \overline{6}
\end{array}
\]
until get clause with only 1 literal of last decision level. “1-UIP”

Can use this backjump clause \( 8 \lor 7 \lor \overline{6} \) to Backjump to \( \ldots 6 \ldots 7 \ 8 \).
Yes, but why is CDCL really *that* good?

Three *key* ingredients (I think):
Yes, but why is CDCL really \textbf{that} good?

Three \textit{key} ingredients (I think):

1. Learn at each conflict \textbf{backjump clause} as a \textit{lemma} ("nogood"):
   - makes \texttt{UnitPropagate} more powerful
   - prevents \texttt{EXP} repeated work in future \textit{similar} conflicts
Yes, but why is CDCL really that good?

Three key ingredients (I think):

1. **Learn** at each conflict backjump clause as a lemma (“nogood”):
   - makes UnitPropagate more powerful
   - prevents EXP repeated work in future similar conflicts

2. **Decide** on variables with many occurrences in Recent conflicts:
   - Dynamic activity-based heuristics
   - idea: work off, one by one, clusters of tightly related vars
     (try CDCL on two independent instances together...)
Yes, but why is CDCL really that good?

Three key ingredients (I think):

1. **Learn** at each conflict *backjump clause* as a *lemma* ("nogood"):  
   - makes *UnitPropagate* more powerful  
   - prevents *EXP* repeated work in future similar conflicts

2. **Decide** on variables with many occurrences in Recent conflicts:  
   - Dynamic activity-based heuristics  
   - idea: *work off*, one by one, clusters of tightly related vars  
     (try CDCL on two independent instances together...)

3. **Forget** from time to time *low-activity lemmas*:  
   - crucial to keep *UnitPropagate* fast and memory affordable  
   - idea: lemmas from *worked-off clusters* no longer needed!
Good vs Bad in CDCL SAT Solvers

Decades of academic and industrial efforts

Lots of $$$ from, e.g., EDA (Electronic Design Automation)

What’s GOOD? Complete solvers:

• with impressive performance
• on real-world problems from many sources, with a
• single, fully automatic, push-button, var selection strategy.
• Hence modeling is essentially declarative.

What’s BAD?

• Low-level language
• Sometimes no adequate/compact encodings: arithmetic...
  0-1 cardinality [Constraints11], P-B [JAIR12], \( \mathbb{Z} \) encodings...
• Answers “unsat” or model. Optimization not as well studied.
What is SAT Modulo Theories (SMT)?

Origin: Reasoning about equality, arithmetic, data structures such as arrays, etc., in Software/Hardware verification.

What is SMT? Deciding satisfiability of an (existential) SAT formula with atoms over a background theory \( T \)

Example 1: \( T \) is Equality with Uninterpreted Functions (EUF):
3 clauses:
\[
f(g(a)) \neq f(c) \lor g(a) = d, \quad g(a) = c, \quad c \neq d
\]

Example 2: several (how many?) combined theories:
2 clauses:
\[
A = write(B, i+1, x), \quad read(A, j+3) = y \lor f(i-1) \neq f(j+1)
\]

Typical verification examples, where SMT is method of choice.
The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002].

Same EUF example:

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

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

Aka Lemmas on demand [dMR,2002].

Same EUF example:

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

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

SAT solver returns model [ \overline{1}, \ 3, \ \overline{4} ]
The \textbf{Lazy} approach to SMT

Aka \textit{Lemmas on demand} [dMR,2002]. \textit{Same EUF example:}

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

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

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

Theory solver says \[ 1, 3, 4 \] is \textit{T}-inconsistent
The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002].

Same EUF example:

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

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

   SAT solver returns model \([ \overline{1}, \ 3, \ \overline{4} ]\)

   Theory solver says \([ \overline{1}, \ 3, \ \overline{4} ]\) is \(T\)-inconsistent

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

Aka [Lemmas on demand](#) [dMR,2002].

Same EUF example:

\[ f(g(a)) \neq f(c) \vee g(a) = d, \quad g(a) = c, \quad c \neq d \]

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

   SAT solver returns model [ 1, 3, 4 ]

   Theory solver says [ 1, 3, 4 ] is \( T \)-inconsistent

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

   SAT solver says UNSAT
The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002].

Same EUF example:

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

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

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

Aka Lemmas on demand [dMR,2002].

Same EUF example:

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

1. Send \( \{1 \lor 2, \ 3, \ \overline{4}\} \) to SAT solver
   
   SAT solver returns model \([1, \ 3, \ \overline{4}\] \)

   Theory solver says \([1, \ 3, \ \overline{4}\] \) is \(T\)-inconsistent

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

Aka Lemmas on demand [dMR,2002].

Same EUF example:

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

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

2. Send \{\overline{1} \lor 2, 3, \overline{4}, 1 \lor 3 \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 \{\overline{1} \lor 2, 3, \overline{4}, 1 \lor 3 \lor 4, \overline{1} \lor 2 \lor 3 \lor 4\} to SAT solver
   
   SAT solver says UNSAT
Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check $T$-consistency only of full propositional models

Robert Nieuwenhuis Barcelogic and UPC GCAI’15 SAT-based techniques for integer linear constraints
Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

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

Since state-of-the-art SAT solvers are all DPLL-based...

- 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
Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- 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
Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- 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
Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- 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
Our DPLL(T) approach to SMT  (JACM’06)

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

- Modular and flexible: can plug in any \( T\text{-Solvers} \) into the \text{DPLL(X) engine}.  

- \( T\text{-Solvers} \) specialized and fast in Theory Propagation:
  - Propagate literals that are theory consequences
  - more pruning in improved lazy SMT
  - \( T\text{-Solver} \) also guides search, instead of only validating it
  - fully exploited in conflict analysis (non-trivial)

- \text{DPLL(T)} approach is being quite widely adopted (cf. Google).
Conflict analysis in DPLL(T)

Need to do backward resolution with two kinds of clauses:

- **UnitPropagate** with clause $C$: resolve with $C$ (as in SAT)
- **T-Propagate** of $\text{lit}$: resolve with (small) explanation

$$l_1 \land \ldots \land l_n \rightarrow \text{lit}$$

or, equivalently,

$$\overline{l}_1 \lor \ldots \lor \overline{l}_n \lor \text{lit} \quad \text{provided by T-Solver}$$

How should it be implemented? (see again [JACM’06])

- **UnitPropagate**: store a pointer to clause $C$, as in SAT solvers
- **T-Propagate**: (pre-)compute explanations at each T-Propagate?
  - **Better** only on demand, during conflict analysis
  - typically only one Explain per $\sim 250$ T-Propagates.
  - depends on $T$. 
ILP as an SMT problem

- The **theory** is the set (conjunction) $S$ of linear constraints.

- **Decide** and **UnitPropagate bounds** $lb \leq x$ and $x \leq ub$.
  
  T-Propagate bounds simply by **bound propagation** with $S$:
  
  E.g., $\{0 \leq x, \ 1 \leq y\} \cup \{x + y + 2z \leq 2\} \implies z \leq 0$

  Explanation clause (disjunction of bounds): $0 \not\leq x \lor 1 \not\leq y \lor z \leq 0$

- **If conflict**: Analyze explanation clauses as in SAT.
  Backjump. Learn one new clause on bounds.
  Also: Forget, Restart, etc. Completeness is standard [JACM’06].

- **NB**: only new **clauses** are Learned. $S$ does not change!

Also developed as Lazy Clause Generation (LCG) by Stuckey et al.
Works **very well** on, e.g., scheduling, timetabling,...
Hybrids of SMT + “bottleneck encoding”

Why does SMT work so well? Because

• most constraints are not bottlenecks:
  they only generate few (different) explanation clauses.
• SMT generates exactly these few clauses on demand.

However,... sometimes there are bottleneck constraints $C$:

• They generate an EXP number of explanation clauses.
  All of them together, (almost) full SAT encoding of $C$.
  And a very naive encoding!
• Compact encoding (w/aux.vars) of these $C$ is needed.
• Idea: detect and encode such bottleneck $C$ on the fly!
  [Abio,Stuckey CP12], further developed with us [CP13]
Outline of this talk

- SAT and ILP
- Commercial ILP tools
- Between SAT and ILP
- CDCL SAT solvers. Why do they work so well?
- What is SMT? Why does it work so well?
- ILP as an SMT problem. Hybrids: SMT + bottleneck encodings

⇒ Going beyond: Constraint Learning. (It can beat clause learning!)
- Solving the rounding problem, 0-1 case, \( \mathbb{Z} \) case
- Cutsat and IntSat. Evaluation. Demo (if time).
- Simple completeness proofs for cutting planes
- Remarks on proof systems
People have tried... extend CDCL to ILP! Learn Constraints!

<table>
<thead>
<tr>
<th>SAT</th>
<th>ILP</th>
</tr>
</thead>
<tbody>
<tr>
<td>clause</td>
<td>linear constraint</td>
</tr>
<tr>
<td>( l_1 \lor \ldots \lor l_n )</td>
<td>( a_1 x_1 + \cdots + a_n x_n \leq a_0 )</td>
</tr>
<tr>
<td>0-1 variable</td>
<td>integer variable</td>
</tr>
<tr>
<td>( x )</td>
<td>( x )</td>
</tr>
<tr>
<td>positive literal</td>
<td>lower bound</td>
</tr>
<tr>
<td>( x )</td>
<td>( a \leq x )</td>
</tr>
<tr>
<td>negative literal</td>
<td>upper bound</td>
</tr>
<tr>
<td>( \overline{x} )</td>
<td>( x \leq a )</td>
</tr>
<tr>
<td>unit propagation</td>
<td>bound propagation</td>
</tr>
<tr>
<td>decide any literal</td>
<td>decide any bound</td>
</tr>
<tr>
<td>resolution inference</td>
<td>cut inference</td>
</tr>
</tbody>
</table>

Cut, eliminating \( x \) from \( 4x + 4y + 2z \leq 3 \) and \( -10x + y - z \leq 0 \):

\[
5 \cdot (4x + 4y + 2z \leq 3) + \\
2 \cdot (-10x + y - z \leq 0) + \\
\overline{22y + 8z \leq 15} = 11y + 4z \leq 7
\]
Learned cuts can be stronger than SMT clauses!

0-1 example:

\[ C_1 : \quad x + y - z \leq 1 \]
\[ C_2 : \quad -2x + 3y + z - u \leq 1 \]
\[ C_3 : \quad 2x - 3y + z + u \leq 0 \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>C_3 conflict!</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ≤ u</td>
<td></td>
<td>C_2</td>
<td></td>
</tr>
<tr>
<td>1 ≤ z</td>
<td></td>
<td>C_1</td>
<td></td>
</tr>
<tr>
<td>1 ≤ y</td>
<td>decision</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 ≤ x</td>
<td>decision</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stack ↑

bound reason

resolution(\(C_2, C_3\)) = \(1 \not\leq y \lor 1 \not\leq z \lor 1 \leq u \)
\[\frac{1 \not\leq x \lor 1 \not\leq z \lor 1 \not\leq u}{1 \not\leq x \lor 1 \not\leq y \lor 1 \not\leq z}\]

which is: \(x \leq 0 \lor y \leq 0 \lor z \leq 0 \equiv x + y + z \leq 2\)

\[\text{cut}(C_2, C_3) = \frac{-2x + 3y + z - u \leq 1}{2z \leq 1}\]
\[\frac{2x - 3y + z + u \leq 0}{2z \leq 1}\]

which is: \(z \leq 0\)
The rounding problem (even in 0-1 case):

\[
\begin{array}{c|c}
C_2 & \text{conflict!} \\
\hline
1 \leq z & C_1 \\
1 \leq y & \text{decision} \\
1 \leq x & \text{decision} \\
\end{array}
\]

by rounding \( \left\lceil \frac{1}{2} \right\rceil \leq z \)

\( C_1 : x + y - 2z \leq 1 \)
\( C_2 : x + y + 2z \leq 3 \)

\[
cut(C_1, C_2) = \frac{x + y - 2z \leq 1}{2x + 2y \leq 4} \]
\[
\quad \frac{x + y + 2z \leq 3}{\text{which is: } x + y \leq 2}
\]

Now conflict analysis is finished:

for \( x + y \leq 2 \) only one bound \( (1 \leq y) \) at this dl is relevant.

And we are stuck: \( x + y \leq 2 \) is too weak to force a backjump.

In fact it is a useless tautology in this 0-1 case.
Solving the rounding pb in the 0-1 case

Can always go the pure SMT way:

- Some Pseudo-Boolean (0-1 ILP) solvers only learn clauses. These are in fact SMT solvers.

But can be smarter:

- Do this only at confl.analysis steps with rounding pb! Then, since any clause on 0-1 bounds is expressible as a constraint, can cut at this step with $x + y - z \leq 1$ (≡ $\neg x \lor 1 \lor \neg y \lor 1 \leq z$).

- Coeff($z$) = ±1: no rounding pb; can always backjump.

- Even better, use cardinality explanations: [Dixon,Chai...]

See [handbook RousselEtal’09] + refs. for much more on P-B solving
Solving the rounding pb; $\mathbb{Z}$ case: Cutsat

- Very nice result [Jovanović, De Moura ’11].
- Decisions **must** make a var equal to its upper/lower bound.
- Then, during conflict analysis, for each propagated $x$, one can compute a **tight** reason, i.e., with $\text{Coeff}(x) = \pm 1$.
  This process uses a number of non-variable eliminating cuts.
- As before: then no rounding pb; can always backjump.

This learning scheme is similar to the **all-decisions** SAT one, which performs much worse than 1UIP in SAT (and also in ILP).
The IntSat Method for ILP in \( \mathbb{Z} \) [CP14]

- IntSat admits \textit{arbitrary} new bounds as decisions.
- After each conflict it can always backjump and learn new a constraint.
- It guides the search exactly as 1UIP in CDCL.

\textbf{Idea}: Dual conflict analysis: cuts+SMT.

If no Backjump from cuts, do SMT one.
Learn no clause on bounds, \textit{except if convertible into a constraint} (new!)

Technical details:

- If set of bounds \( R \) in stack + constraint \( C \) propagate bound \( B \),
  \( B \) is pushed on stack w/ reason constraint \( C \) and reason set \( R \).
- Conflict an. and cuts guided by \textbf{Conflicting Set (CS)} of bounds:
  - Invariant: \( CS \subseteq \text{stack, and } CS \cup S \text{ is infeasible.} \)
  - Each confl.an. step: Replace topmost bound of \( CS \) by its reason set and attempt the corresponding cut.
Example

\[ C_0 : \quad x - 3y - 3z \leq 1 \quad \text{and initial bounds: } \quad 1 \leq y \quad y \leq 4 \]
\[ C_1 : \quad -2x + 3y + 2z \leq -2 \]
\[ C_2 : \quad 3x - 3y + 2z \leq -1 \quad \quad -2 \leq x \quad x \leq 3 \]

Stack:

<table>
<thead>
<tr>
<th>2 \leq y</th>
<th>{ 1 \leq x, z \leq -2 }</th>
<th>C_0 : \quad x - 3y - 3z \leq 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x \leq 1</td>
<td>{ y \leq 2, z \leq -2 }</td>
<td>C_0 : \quad x - 3y - 3z \leq 1</td>
</tr>
<tr>
<td>z \leq -2</td>
<td>\text{decision}</td>
<td></td>
</tr>
<tr>
<td>z \leq -1</td>
<td>{ x \leq 2, 1 \leq y }</td>
<td>C_1 : \quad -2x + 3y + 2z \leq -2</td>
</tr>
<tr>
<td>x \leq 2</td>
<td>\text{decision}</td>
<td></td>
</tr>
<tr>
<td>z \leq 0</td>
<td>{ x \leq 3, 1 \leq y }</td>
<td>C_1 : \quad -2x + 3y + 2z \leq -2</td>
</tr>
<tr>
<td>y \leq 2</td>
<td>{ x \leq 3, -2 \leq z }</td>
<td>C_1 : \quad -2x + 3y + 2z \leq -2</td>
</tr>
<tr>
<td>1 \leq x</td>
<td>{ 1 \leq y, -2 \leq z }</td>
<td>C_1 : \quad -2x + 3y + 2z \leq -2</td>
</tr>
<tr>
<td>-2 \leq z</td>
<td>\text{initial}</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
**Example (II)**

<table>
<thead>
<tr>
<th>$2 \leq y$</th>
<th>${ 1 \leq x, z \leq -2 }$</th>
<th>$C_0: x - 3y - 3z \leq 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \leq 1$</td>
<td>${ y \leq 2, z \leq -2 }$</td>
<td>$C_0: x - 3y - 3z \leq 1$</td>
</tr>
<tr>
<td>$z \leq -2$</td>
<td></td>
<td>$\text{decision}$</td>
</tr>
<tr>
<td>$z \leq -1$</td>
<td>${ x \leq 2, 1 \leq y }$</td>
<td>$C_1: -2x + 3y + 2z \leq -2$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td></td>
<td>$\text{decision}$</td>
</tr>
<tr>
<td>$z \leq 0$</td>
<td>${ x \leq 3, 1 \leq y }$</td>
<td>$C_1: -2x + 3y + 2z \leq -2$</td>
</tr>
<tr>
<td>$y \leq 2$</td>
<td>${ x \leq 3, -2 \leq z }$</td>
<td>$C_1: -2x + 3y + 2z \leq -2$</td>
</tr>
<tr>
<td>$1 \leq x$</td>
<td>${ 1 \leq y, -2 \leq z }$</td>
<td>$C_1: -2x + 3y + 2z \leq -2$</td>
</tr>
<tr>
<td>$-2 \leq z$</td>
<td></td>
<td>$\text{initial}$</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

- **bound**  - **reason set**  - **reason constraint**

We had:

Now, conflict $C_1$, with initial CS $\{ -2 \leq z, x \leq 1, 2 \leq y \}$.

Replacing $2 \leq y$ by its r.set, CS $= \{ -2 \leq z, 1 \leq x, z \leq -2, x \leq 1 \}$.

Cut eliminating $y$ between $C_1$ and $C_0$ gives $C_3: -x - z \leq -1$.

**Early backjump** due to $z \leq -1$: add $2 \leq x$ at dl 1 and learn $C_3$.  

Robert Nieuwenhuis  
Barcelogic and UPC  
GCAI’15  
SAT-based techniques for integer linear constraints
New bound $2 \leq x$ at dl 1 triggers two more propagations:

<table>
<thead>
<tr>
<th></th>
<th>2 $\leq$ y</th>
<th>${ 2 \leq x, , z \leq -1 }$</th>
<th>$C_0: \quad x - 3y - 3z \leq 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$-1 \leq z$</td>
<td>${ x \leq 2 }$</td>
<td>$C_3: \quad -x - z \leq -1$</td>
</tr>
<tr>
<td></td>
<td>$2 \leq x$</td>
<td>${ z \leq -1 }$</td>
<td>$C_3: \quad -x - z \leq -1$</td>
</tr>
<tr>
<td></td>
<td>$z \leq -1$</td>
<td>${ x \leq 2, , 1 \leq y }$</td>
<td>$C_1: \quad -2x + 3y + 2z \leq -2$</td>
</tr>
<tr>
<td></td>
<td>$x \leq 2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$z \leq 0$</td>
<td>${ x \leq 3, , 1 \leq y }$</td>
<td>$C_1: \quad -2x + 3y + 2z \leq -2$</td>
</tr>
<tr>
<td></td>
<td>$y \leq 2$</td>
<td>${ x \leq 3, , -2 \leq z }$</td>
<td>$C_1: \quad -2x + 3y + 2z \leq -2$</td>
</tr>
<tr>
<td></td>
<td>$1 \leq x$</td>
<td>${ 1 \leq y, , -2 \leq z }$</td>
<td>$C_1: \quad -2x + 3y + 2z \leq -2$</td>
</tr>
<tr>
<td></td>
<td>$-2 \leq z$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Again conflict $C_1$. $CS = \{ x \leq 2, \, -1 \leq z, \, 2 \leq y \}$. 4-step conflict an.:

1. Replace $2 \leq y$. $CS = \{ x \leq 2, \, z \leq -1, \, 2 \leq x, \, -1 \leq z \}$. 
   Cut($C_0, C_1$) gives $C: \quad -x - z \leq -1$ as before.
2. Replace $-1 \leq z$. $CS = \{ x \leq 2, \ z \leq -1, \ 2 \leq x \}$
   No cut is made (since $z$ is negative in both $C$ and $C_3$).

3. Replace $2 \leq x$. $CS = \{ x \leq 2, \ z \leq -1 \}$; no cut (same for $x$).

4. Replace $z \leq -1$. $CS = \{ 1 \leq y, \ x \leq 2 \}$.
   Cut gives $-4x + 3y \leq -4$; early bckjmp adding $2 \leq x$ at dl 0?
   But C.An. is also finished (only one bound of this dl in $CS$): can backjump to dl 0 adding $x \not\leq 2$, i.e., $3 \leq x$ (stronger!).

After one further propagation ($-1 \leq z$), the procedure returns “infeasible” since conflict $C_2$ appears at dl 0.
Optimization

Unlike SAT, here linear constraints are first-class citizens (belong to the core language).

So can optimize doing simple branch and bound:

To minimize \( a_1 x_1 + \ldots + a_n x_n \) (\( = \) maximize \( -a_1 x_1 - \ldots - a_n x_n \))

- First find arbitrary solution \( S_0 \)
- Repeat after each new solution \( S_i \):
  - add constraint \( a_1 x_1 + \ldots + a_n x_n < \text{cost}(S_i) \)
  - re-run
  Until infeasible.

Bound propagation from these successively stronger constraints prunes a lot.
Theorem

- IntSat always finds the optimal solution (if any).
- If moreover variables are upper and lower bounded,
  - IntSat always terminates
  - it returns “infeasible” iff input is infeasible.

(See [CP’14] for details)
Proof of concept: small naive toy C++ program. Some ideas:

- Vars and coefficients are just 4-byte int's
  - cuts giving coefficients $> 2^{30}$ are simply discarded
  - so no overflow if intermediate computations in $2^{64}$ int's.

- $O(1)$-time access to current upper (lower) bound for var:
  - bounds for $x$ in stack have ptr to previous bound for $x$
  - maintain pointer to topmost (i.e., strongest) one

- Cache-efficient counter-based bound propagation:
  - occurs lists for each var (and sign)
  - only need to access actual constraint if its filter value becomes positive
• Commercial OR solvers, huge and expensive.
• Based on LP relaxation + Simplex + branch-and-cut.
• Combine a large variety of techniques:
  problem-specific cuts, specialized heuristics, presolving...
• Extremely mature technology. Bixby [5]:
  “From 1991 to 2012, saw $475,000 \times$ algorithmic speedup + $2,000 \times$
  hardware speedup.”
• We compare here with their latest versions (on 4 cores)
naive little C++ program (1 core)
naive little C++ program (1 core)

- First completely different technique that shows some competitiveness.
- Even on MIPLIB, according to miplib.zib.de, OR’s “standard test set”, including “hard” and “open” problems, up to over 150,000 constraints and 100,000 variables.
- Even with this small “toy” implementation. Lots of room for improvement (conceptual & implementation)
IntSat experiments, see [CP14]

IntSat “toy” (1-core) vs newest CPLEX and Gurobi (4-core)

1. Random optimization instances:
   - 600 vars, 750 constraints, 10s time limit
   - IntSat overall better than CPLEX, slightly worse than Gurobi.

2. MIPLIB (600 s; for all but 7 instances no solver proves optimality)
   - All 19 MIPLIB’s bounded pure ILP instances, incl. “hard” & “open” ones, up to over 150,000 constraints, 100,000 vars.
   - (toy-) IntSat frequently
     - is fastest proving feasibility
     - finds good (or optimal) solutions faster than C&G
     in particular for some of the largest instances.
Lots of improvements to explore

- **Implementation-wise:**
  - special treatments for binary variables
  - special treatments for specific kinds of constraints
  - efficient early backjumps [solved?]
  - ...

- **Conceptual improvements:**
  - decision heuristics
  - restarts and cleanups
  - optimization (“first-succeed”, initial solutions,...)
  - pre- and in-processing: extremely effective in SAT, nothing done here yet
  - MIPs
  - ...

Robert Nieuwenhuis
Barcelogic and UPC
GCAI’15
SAT-based techniques for integer linear constraints
Simple completeness proofs

(joint work with Marc Bezem)

- Theory of (0-1) ILP historically based on LP in $\mathbb{Q}$. Completeness in, e.g., Schrijver’98, uses many results from previous 300+ pages.

- Moreover, standard cutting planes rules are difficult to control:

  \[
  \frac{p \geq c}{np + mq \geq nc + md} \quad \text{where} \quad n, m \in \mathbb{N}
  \]

  \[
  \frac{a_n x_n + \ldots + a_1 x_1 \geq c}{\lceil a_n/d \rceil x_n + \ldots + \lceil a_1/d \rceil x_1 \geq \lceil c/d \rceil} \quad \text{where} \quad d \in \mathbb{N}^+
  \]

- We have new self-contained proofs, 0-1 and $\mathbb{Z}$ cases, where:
  - **Combine** factors $n, m$ always fully determined, so that the maximal var is either eliminated or increased by a precise amount
  - **Combine** on maximal vars only, one of them always with coefficient 1
  - **Divide** only if $d$ is the coefficient of the maximal var and $d|a_i$ for all $i$
Proof sketch for full ILP case.

Let $S$ over $x_1 \ldots x_n$ be bounded, closed under Combine, Divide, no contrad.

Build solution $M_i$ for each $S_i \subseteq S$ with vars in $x_1 \ldots x_i$ only, by induction on $i$.

Base case $i = 0$: trivial since $S$ has no contradictions (and $S_0$ has no vars).

Ind. step $i > 0$: extend $M_{i-1}$ to $M_i$ by defining

$$M_i(x_i) = \max \{ c - M_{i-1}(p) \mid x_i + p \geq c \text{ in } S_i \}$$

Now prove $M_i \models C$ for all $C$ in $S_i \setminus S_{i-1}$. Here $C$ can be:

A) $x_i + p \geq c$. Then $M_i \models C$ by construction of $M_i$.

B) $-ax_i + p \geq c$ with $a > 0$. Now $M_i(x_i)$ is due to some $x_i + q \geq d$ in $S_i$.

Combine them eliminating $x_i$ (note: $x_i$ is maximal in both premises).

The conclusion is in $S_{i-1}$ and entails by IH that $M_i \models C$.

C) $ax_i + p \geq c$ with $a > 1$.

C1) If $a \mid p$ do Divide and reduce to case A).

C2) Otherwise, Combine on $b x_j$, maximal var $x_j$ in $p$ with $a \nmid b$. 
Remarks on the proof systems

- More restrictive proof systems: less work, easier to automatize

- **trade-off:** such systems tend to be less “efficient” in terms of proof length.
  
  0-1: only need var.-eliminating Combine or w/ bounds $0 \leq x \text{ and } x \leq 1$.
  
  this does not look any stronger than resolution
  
  but full Combine does have short proofs for pigeon hole problem.

- Does this have any practical consequences for CDCL-based ILP provers?

- If so, are there any “controllable” appropriate intermediate systems?
• Probably no single technique will dominate.
• But these methods (such as IntSat) may become one standard tool in the toolbox.

Thank you!