SAT-based techniques for integer linear constraints

GCAI 2015 (invited talk)

Robert Nieuwenhuis

Barcelogic.com - Computer Science Department
BarcelonaTech (UPC)

Barcelogic
We make efficiency simple for you

UPC
UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Thanks for inviting me, bringing me back to this wonderful country!

Between SAT and ILP

	0-1 solutions		$\mathbb{Z}$ solutions	
	feasibility	optimizing	feasibility	optimizing
clauses	SAT			
cardinality constr.				
linear constraints				ILP

Between SAT and ILP

	0-1 sols		$\mathbb{Z}$ sols		$\mathbb{Q}/\mathbb{Z}$ sols	
	feas.	opt.	feas.	opt.	feas.	opt.
clauses	SAT					
cardinality constr.						
linear constraints				ILP		MIPs

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

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 $Sol: \{x_1 \dots x_n\} \rightarrow \mathbb{Z}$ to:

$$
\begin{array}{llcll}
\text{Minimize:} & c_1\ x_1 + \dots + c_n\ x_n & & & \text{(or maximize)} \\
\text{Subject To:} & c_{11}\ x_1 + \dots + c_{1n}\ x_n & \leq & c_{10} & \\
 & \dots & & \dots & \text{(or with } \geq, =, <, >\text{)} \\
 & c_{m1}\ x_1 + \dots + c_{mn}\ x_n & \leq & c_{m0} &
\end{array}
$$

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

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

$$x \vee \overline{y} \vee \overline{z} \quad \equiv \quad x + (1 - y) + (1 - z) \geq 1$$

CPLEX and Gurobi

- 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

	0-1 solutions		$\mathbb{Z}$ solutions	
	feasibility	optimizing	feasibility	optimizing
clauses	SAT			
cardinality constr.				
linear constr.	0-1 ILP(P-B)	0-1 ILP (P-B)		ILP

Cardinality constraints:

$$x_1 + \ldots + x_n \leq k \qquad \text{(or with } \geq, =, <, >\text{)}$$

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

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:

$\overline{1} \vee 2,\ \ \overline{3} \vee 4,\ \ \overline{5} \vee \overline{6},\ \ 6 \vee \overline{5} \vee \overline{2}$

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:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	

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:**		
	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	

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:**		
	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	(Decide)
1 2 3	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	

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:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	

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:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3 4 5	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	

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:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3 4 5	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4 5 $\overline{6}$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$		

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:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
$1\ 2\ 3$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2\ 3\ 4$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
$1\ 2\ 3\ 4\ 5$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2\ 3\ 4\ 5\ \overline{6}$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$		*CONFLICT*!

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

CDCL = Conflict-Driven Clause-Learning backtracking algorithm

Candidate Solution:	**Four clauses:**		
	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
1 2 3	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
1 2 3 4 5	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4 5 $\overline{6}$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(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:**		
	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
$1\ 2\ 3$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2\ 3\ 4$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
$1\ 2\ 3\ 4\ 5$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2\ 3\ 4\ 5\ \overline{6}$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Backtrack)
$1\ 2\ 3\ 4\ \overline{5}$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$		

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:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3 4 5	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4 5 $\overline{6}$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Backtrack)
1 2 3 4 $\overline{5}$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$		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

Candidate Solution:	**Four clauses:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3 4 5	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4 5 $\overline{6}$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Backtrack)
1 2 3 4 $\overline{5}$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$		solution found!

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

Backtrack vs. Backjump

Same example. Remember: Backtrack gave 1 2 3 4 $\overline{5}$.

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

$$
\begin{array}{llll}
\emptyset & \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} & \Rightarrow & (\text{Decide}) \\
\vdots \qquad\qquad \vdots & \qquad\qquad \vdots & & \\
1\ 2\ 3\ 4\ 5\ \overline{6} & \overline{1} \lor 2,\ \overline{3} \lor 4,\ \overline{5} \lor \overline{6},\ 6 \lor \overline{5} \lor \overline{2} & \Rightarrow & (\text{Backjump})
\end{array}
$$

Backtrack vs. Backjump

Same example. Remember: Backtrack gave $1\ 2\ 3\ 4\ \overline{5}$.

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

$$\begin{array}{llll}
\emptyset & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide}) \\
\vdots \qquad\qquad \vdots & \qquad\qquad \vdots & & \\
1\ 2\ 3\ 4\ 5\ \overline{6} & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Backjump}) \\
1\ 2\ \overline{5} & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & \ldots
\end{array}$$

Backtrack vs. Backjump

Same example. Remember: Backtrack gave $1\ 2\ 3\ 4\ \overline{5}$.

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

$$\begin{array}{lll} \emptyset & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad (\text{Decide}) \\ \vdots & \vdots \qquad \vdots & \\ 1\ 2\ 3\ 4\ 5\ \overline{6} & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad (\text{Backjump}) \\ 1\ 2\ \overline{5} & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad \dots \end{array}$$

Backjump =

1. Conflict Analysis: "Find" a backjump clause $C \vee l$ (here, $\overline{2} \vee \overline{5}$)
   - 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: $\ldots 6 \ldots \overline{7} \ldots 9$ and clauses:

$\overline{9} \vee \overline{6} \vee 7 \vee \overline{8}$, $8 \vee 7 \vee \overline{5}$, $\overline{6} \vee 8 \vee 4$, $\overline{4} \vee \overline{1}$, $\overline{4} \vee 5 \vee 2$, $5 \vee 7 \vee \overline{3}$, $1 \vee \overline{2} \vee 3$

UnitPropagate gives $\ldots 6 \ldots \overline{7} \ldots 9\ \overline{8}\ \overline{5}\ 4\ \overline{1}\ 2\ \overline{3}$. Conflict w/ $1 \vee \overline{2} \vee 3$!

C.An. = do resolutions with reason clauses backwards from conflict:

$$
\dfrac{8 \vee 7 \vee \overline{5} \qquad \dfrac{\overline{6} \vee 8 \vee 4 \qquad \dfrac{\overline{4} \vee \overline{1} \qquad \dfrac{\overline{4} \vee 5 \vee 2 \qquad \dfrac{5 \vee 7 \vee \overline{3} \qquad 1 \vee \overline{2} \vee 3}{5 \vee 7 \vee 1 \vee \overline{2}}}{\overline{4} \vee 5 \vee 7 \vee 1}}{5 \vee 7 \vee \overline{4}}}{\overline{6} \vee 8 \vee 7 \vee 5}}{8 \vee 7 \vee \overline{6}}
$$

until get clause with only 1 literal of last decision level. "1-UIP"

Can use this backjump clause $8 \vee 7 \vee \overline{6}$ to Backjump to $\ldots 6 \ldots \overline{7}\ 8$.

Yes, but why is CDCL really that good?

Three key ingredients (I think):

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

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: $f(g(a)) \neq f(c) \lor g(a) = d, \qquad g(a) = c, \qquad c \neq d$

Example 2: several (how many?) combined theories:

2 clauses: $A = write(B, i+1, x), \qquad read(A, j+3) = y \lor f(i-1) \neq f(j+1)$

Typical verification examples, where SMT is method of choice.

The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

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

The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

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

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

The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

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

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

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

The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

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

The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a)=d}_{2}, \qquad \underbrace{g(a)=c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

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

   SAT solver returns model $[\ 1,\ 2,\ 3,\ \overline{4}\ ]$

The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a)=d}_{2}, \qquad \underbrace{g(a)=c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4}\ \}$ 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 $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4},\ \ 1 \vee \overline{3} \vee 4\ \}$ 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

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a)=d}_{2}, \qquad \underbrace{g(a)=c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

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

The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a))\neq f(c)}_{\overline{1}} \vee \underbrace{g(a)=d}_{2}, \qquad \underbrace{g(a)=c}_{3}, \qquad \underbrace{c\neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1}\vee 2,\ \ 3,\ \ \overline{4}\ \}$ 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 $\{\ \overline{1}\vee 2,\ \ 3,\ \ \overline{4},\ \ 1\vee\overline{3}\vee 4\ \}$ 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 $\{\overline{1}\vee 2,\ \ 3,\ \ \overline{4},\ 1\vee\overline{3}\vee 4,\ \ \overline{1}\vee\overline{2}\vee\overline{3}\vee 4\ \}$ 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

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)

DPLL(T) = DPLL(X) engine + *T*-Solvers

- Modular and flexible: can plug in any *T*-Solvers into the DPLL(X) engine.
- *T*-Solvers specialized and fast in Theory Propagation:
  - Propagate literals that are theory consequences
  - more pruning in improved lazy SMT
  - *T*-Solver also guides search, instead of only validating it
  - fully exploited in conflict analysis (non-trivial)
- 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 *lit* : resolve with (small) explanation
  $l_1 \wedge \ldots \wedge l_n \rightarrow lit$
  or, equivalently,
  $\bar{l}_1 \vee \ldots \vee \bar{l}_n \ \vee \ lit$ 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): $0 \nleq x \ \vee\ 1 \nleq y \ \vee\ z \leq 0$

- 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

$\Rightarrow$ 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!

SAT		ILP	
clause	$l_1 \vee \ldots \vee l_n$	*linear constraint*	$a_1x_1 + \cdots + a_nx_n \leq a_0$
0-1 variable	x	*integer* variable	x
positive literal	x	*lower bound*	$a \leq x$
negative literal	$\overline{x}$	*upper bound*	$x \leq a$
unit propagation		*bound* propagation	
decide any literal		decide any *bound*	
resolution inference		*cut* inference	

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

$$\begin{array}{rrrrrrrrrrl}
5 \cdot (& 4x & + & 4y & + & 2z & \leq & 3 &) & & \\
2 \cdot (& -10x & + & y & - & z & \leq & 0 &) & + & \\
\hline
 & & & 22y & + & 8z & \leq & 15 & & & = \quad 11y + 4z \leq 7
\end{array}$$

Learned cuts can be stronger than SMT clauses!

0-1 example:

$$
\begin{array}{lrcl}
C_1: & x+y-z & \leq & 1 \\
C_2: & -2x+3y+z-u & \leq & 1 \\
C_3: & 2x-3y+z+u & \leq & 0
\end{array}
$$

C_3 conflict!	
$1 \leq u$	C_2
$1 \leq z$	C_1
$1 \leq y$	*decision*
$1 \leq x$	*decision*
bound	reason

Stack ↑

$$\text{resolution}(C_2, C_3) = \frac{1 \not\leq y \vee 1 \not\leq z \vee 1 \leq u \qquad 1 \not\leq x \vee 1 \not\leq z \vee 1 \not\leq u}{1 \not\leq x \vee 1 \not\leq y \vee 1 \not\leq z}$$

which is: $\quad x \leq 0 \vee y \leq 0 \vee z \leq 0 \quad \equiv \quad x+y+z \leq 2$

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

which is: $\quad z \leq 0$

The rounding problem (even in 0-1 case):

$C_1: \;\; x+y-2z \leq 1$

$C_2: \;\; x+y+2z \leq 3$

C_2 conflict!	
$1 \leq z$	C_1
$1 \leq y$	*decision*
$1 \leq x$	*decision*
bound	reason

by rounding $\lceil 1/2 \rceil \leq z$

$$\text{cut}(C_1, C_2) \;=\; \frac{x+y-2z \leq 1 \qquad x+y+2z \leq 3}{2x+2y \leq 4}$$

which is: $\quad x+y \leq 2$

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 $x + y - z \leq 1 \quad (\equiv \quad 1 \not\leq x \vee 1 \not\leq y \vee 1 \leq z)$.
- $\text{Coeff}(z) = \pm 1$: 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 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.

- Idea: Dual conflict analysis: cuts+SMT.
  If no Backjump from cuts, do SMT one.

  Learn no clause on bounds, 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 Conflicting Set (CS) of bounds:
  - Invariant: $CS \subseteq$ stack, and $CS \cup S$ is infeasible.
  - Each confl.an. step: Replace topmost bound of CS by its reason set and attempt the corresponding cut.

Example

$$C_0: \quad x \quad -3y \quad -3z \quad \le \quad 1$$
$$C_1: \quad -2x \quad +3y \quad +2z \quad \le \quad -2$$
$$C_2: \quad 3x \quad -3y \quad +2z \quad \le \quad -1$$

and initial bounds:

$-2 \le z \quad z \le 2$
$1 \le y \quad y \le 4$
$-2 \le x \quad x \le 3$

Stack:

bound	reason set	reason constraint
$2 \le y$	$\{\ 1 \le x,\ z \le -2\ \}$	$C_0: \ x - 3y - 3z \le 1$
$x \le 1$	$\{\ y \le 2,\ z \le -2\ \}$	$C_0: \ x - 3y - 3z \le 1$
$z \le -2$	*decision*	
$z \le -1$	$\{\ x \le 2,\ 1 \le y\ \}$	$C_1: \ -2x + 3y + 2z \le -2$
$x \le 2$	*decision*	
$z \le 0$	$\{\ x \le 3,\ 1 \le y\ \}$	$C_1: \ -2x + 3y + 2z \le -2$
$y \le 2$	$\{\ x \le 3,\ -2 \le z\ \}$	$C_1: \ -2x + 3y + 2z \le -2$
$1 \le x$	$\{\ 1 \le y,\ -2 \le z\ \}$	$C_1: \ -2x + 3y + 2z \le -2$
$-2 \le z$	*initial*	
...	...	

Example (II)

We had:

$2 \le y$	$\{\, 1 \le x,\ z \le -2 \,\}$	$C_0: \quad x - 3y - 3z \le 1$
$x \le 1$	$\{\, y \le 2,\ z \le -2 \,\}$	$C_0: \quad x - 3y - 3z \le 1$
$z \le -2$	*decision*	
$z \le -1$	$\{\, x \le 2,\ 1 \le y \,\}$	$C_1: \quad -2x + 3y + 2z \le -2$
$x \le 2$	*decision*	
$z \le 0$	$\{\, x \le 3,\ 1 \le y \,\}$	$C_1: \quad -2x + 3y + 2z \le -2$
$y \le 2$	$\{\, x \le 3,\ -2 \le z \,\}$	$C_1: \quad -2x + 3y + 2z \le -2$
$1 \le x$	$\{\, 1 \le y,\ -2 \le z \,\}$	$C_1: \quad -2x + 3y + 2z \le -2$
$-2 \le z$	*initial*	
$\ldots$	$\ldots$	
bound	reason set	reason constraint

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

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

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

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

Example (III)

New bound $2 \leq x$ at dl 1 triggers two more propagations:

$2 \leq y$	$\{\, 2 \leq x,\ z \leq -1 \,\}$	$C_0: \quad x - 3y - 3z \leq 1$
$-1 \leq z$	$\{\, x \leq 2 \,\}$	$C_3: \quad -x - z \leq -1$
$2 \leq x$	$\{\, z \leq -1 \,\}$	$C_3: \quad -x - z \leq -1$
$z \leq -1$	$\{\, x \leq 2,\ 1 \leq y \,\}$	$C_1: \quad -2x + 3y + 2z \leq -2$
$x \leq 2$	*decision*	
$z \leq 0$	$\{\, x \leq 3,\ 1 \leq y \,\}$	$C_1: \quad -2x + 3y + 2z \leq -2$
$y \leq 2$	$\{\, x \leq 3,\ -2 \leq z \,\}$	$C_1: \quad -2x + 3y + 2z \leq -2$
$1 \leq x$	$\{\, 1 \leq y,\ -2 \leq z \,\}$	$C_1: \quad -2x + 3y + 2z \leq -2$
$-2 \leq z$	*initial*	

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.

Example (finished!)

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 $a_1x_1 + \ldots + a_nx_n$ $\;(= \text{maximize } -a_1x_1 - \ldots - a_nx_n\;)$

- First find arbitrary solution S_0
- Repeat after each new solution S_i:
  – add constraint $a_1x_1 + \ldots + a_nx_n < cost(S_i)$
  – 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)

Implementation

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

CPLEX and Gurobi

- 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× algorithmic speedup + 2,000× hardware speedup."
- We compare here with their latest versions (on 4 cores)

IntSat

naive little C++ program (1 core)

IntSat

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
  - ...

DEMOS

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:

$$\text{Combine}: \quad \frac{p \geq c \qquad q \geq d}{np + mq \geq nc + md} \quad \text{where} \quad n, m \in \mathbb{N}$$

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

- 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 $a | p$ do Divide and reduce to case A).

   C2) Otherwise, Combine on $b\,x_j$, maximal var x_j in p with $a \not| b$.

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 $0 \leq x$ 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!