Solving Logic Formulas in Linear Time

Jakob Nordström

KTH Royal Institute of Technology
Stockholm, Sweden

Datalogisk Institut på Københavns Universitet
April 13, 2018

Solving Logic Formulas in Linear Time

(At Least Surprisingly Often)

Jakob Nordström

KTH Royal Institute of Technology
Stockholm, Sweden

Datalogisk Institut på Københavns Universitet
April 13, 2018

This Is Me...

Jakob Nordström

Associate Professor

Theoretical Computer Science Group

School of Electrical Engineering and Computer Science

www.csc.kth.se/∼jakobn

jakobn@kth.se

. . . And This Is What I Do for a Living

$(x_{1,1} \lor x_{1,2} \lor x_{1,3} \lor x_{1,4} \lor x_{1,5} \lor x_{1,6} \lor x_{1,7}) \land (x_{2,1} \lor x_{2,2} \lor x_{2,3} \lor x_{2,4} \lor x_{2,5} \lor x_{2,6} \lor x_{2,7}) \land (x_{3,1} \lor x_{3,2} \lor x_{3,3} \lor x_{3,4} \lor x_{3,5} \lor x_{3,6} \lor x_{3,7}) \land (x_{4,1} \lor x_{4,2} \lor x_{4,3} \lor x_{4,4} \lor x_{4,5} \lor x_{4,6} \lor x_{4,7}) \land (x_{5,1} \lor x_{5,2} \lor x_{5,3} \lor x_{5,4} \lor x_{5,5} \lor x_{5,6} \lor x_{5,7}) \land (x_{6,1} \lor x_{6,2} \lor x_{6,3} \lor x_{6,4} \lor x_{6,5} \lor x_{6,6} \lor x_{6,7}) \land (x_{7,1} \lor x_{7,2} \lor x_{7,3} \lor x_{7,4} \lor x_{7,5} \lor x_{7,6} \lor x_{7,7}) \land (x_{8,1} \lor x_{8,2} \lor x_{8,3} \lor x_{8,4} \lor x_{8,5} \lor x_{8,6} \lor x_{8,7}) \land (\overline{x}_{1,1} \lor \overline{x}_{2,1}) \land (\overline{x}_{1,1} \lor \overline{x}_{3,1}) \land (\overline{x}_{1,1} \lor \overline{x}_{4,1}) \land (\overline{x}_{1,1} \lor \overline{x}_{5,1}) \land (\overline{x}_{1,1} \lor \overline{x}_{6,1}) \land (\overline{x}_{1,1} \lor \overline{x}_{7,1}) \land (\overline{x}_{1,1} \lor \overline{x}_{8,1}) \land (\overline{x}_{2,1} \lor \overline{x}_{3,1}) \land (\overline{x}_{2,1} \lor \overline{x}_{4,1}) \land (\overline{x}_{2,1} \lor \overline{x}_{5,1}) \land (\overline{x}_{2,1} \lor \overline{x}_{6,1}) \land (\overline{x}_{2,1} \lor \overline{x}_{7,1}) \land (\overline{x}_{2,1} \lor \overline{x}_{8,1}) \land (\overline{x}_{3,1} \lor \overline{x}_{4,1}) \land (\overline{x}_{3,1} \lor \overline{x}_{5,1}) \land (\overline{x}_{3,1} \lor \overline{x}_{6,1}) \land (\overline{x}_{3,1} \lor \overline{x}_{7,1}) \land (\overline{x}_{3,1} \lor \overline{x}_{8,1}) \land (\overline{x}_{4,1} \lor \overline{x}_{5,1}) \land (\overline{x}_{4,1} \lor \overline{x}_{6,1}) \land (\overline{x}_{4,1} \lor \overline{x}_{7,1}) \land (\overline{x}_{4,1} \lor \overline{x}_{8,1}) \land (\overline{x}_{5,1} \lor \overline{x}_{6,1}) \land (\overline{x}_{5,1} \lor \overline{x}_{7,1}) \land (\overline{x}_{5,1} \lor \overline{x}_{8,1}) \land (\overline{x}_{6,1} \lor \overline{x}_{7,1}) \land (\overline{x}_{6,1} \lor \overline{x}_{8,1}) \land (\overline{x}_{7,1} \lor \overline{x}_{8,1}) \land (\overline{x}_{1,2} \lor \overline{x}_{2,2}) \land (\overline{x}_{1,2} \lor \overline{x}_{3,2}) \land (\overline{x}_{1,2} \lor \overline{x}_{4,2}) \land (\overline{x}_{1,2} \lor \overline{x}_{5,2}) \land (\overline{x}_{1,2} \lor \overline{x}_{6,2}) \land (\overline{x}_{1,2} \lor \overline{x}_{7,2}) \land (\overline{x}_{1,2} \lor \overline{x}_{8,2}) \land (\overline{x}_{2,2} \lor \overline{x}_{3,2}) \land (\overline{x}_{2,2} \lor \overline{x}_{4,2}) \land (\overline{x}_{2,2} \lor \overline{x}_{5,2}) \land (\overline{x}_{2,2} \lor \overline{x}_{6,2}) \land (\overline{x}_{2,2} \lor \overline{x}_{7,2}) \land (\overline{x}_{2,2} \lor \overline{x}_{8,2}) \land (\overline{x}_{3,2} \lor \overline{x}_{4,2}) \land (\overline{x}_{3,2} \lor \overline{x}_{5,2}) \land (\overline{x}_{3,2} \lor \overline{x}_{6,2}) \land (\overline{x}_{3,2} \lor \overline{x}_{7,2}) \land (\overline{x}_{3,2} \lor \overline{x}_{8,2}) \land (\overline{x}_{4,2} \lor \overline{x}_{5,2}) \land (\overline{x}_{4,2} \lor \overline{x}_{6,2}) \land (\overline{x}_{4,2} \lor \overline{x}_{7,2}) \land (\overline{x}_{4,2} \lor \overline{x}_{8,2}) \land (\overline{x}_{5,2} \lor \overline{x}_{6,2}) \land (\overline{x}_{5,2} \lor \overline{x}_{7,2}) \land (\overline{x}_{5,2} \lor \overline{x}_{8,2}) \land (\overline{x}_{6,2} \lor \overline{x}_{7,2}) \land (\overline{x}_{6,2} \lor \overline{x}_{8,2}) \land (\overline{x}_{7,2} \lor \overline{x}_{8,2}) \land (\overline{x}_{1,3} \lor \overline{x}_{2,3}) \land (\overline{x}_{1,3} \lor \overline{x}_{3,3}) \land (\overline{x}_{1,3} \lor \overline{x}_{4,3}) \land (\overline{x}_{1,3} \lor \overline{x}_{5,3}) \land (\overline{x}_{1,3} \lor \overline{x}_{6,3}) \land (\overline{x}_{1,3} \lor \overline{x}_{7,3}) \land (\overline{x}_{1,3} \lor \overline{x}_{8,3}) \land (\overline{x}_{2,3} \lor \overline{x}_{3,3}) \land (\overline{x}_{2,3} \lor \overline{x}_{4,3}) \land (\overline{x}_{2,3} \lor \overline{x}_{5,3}) \land (\overline{x}_{2,3} \lor \overline{x}_{6,3}) \land (\overline{x}_{2,3} \lor \overline{x}_{7,3}) \land (\overline{x}_{2,3} \lor \overline{x}_{8,3}) \land (\overline{x}_{3,3} \lor \overline{x}_{4,3}) \land (\overline{x}_{3,3} \lor \overline{x}_{5,3}) \land (\overline{x}_{3,3} \lor \overline{x}_{6,3}) \land (\overline{x}_{3,3} \lor \overline{x}_{7,3}) \land (\overline{x}_{3,3} \lor \overline{x}_{8,3}) \land (\overline{x}_{4,3} \lor \overline{x}_{5,3}) \land (\overline{x}_{4,3} \lor \overline{x}_{6,3}) \land (\overline{x}_{4,3} \lor \overline{x}_{7,3}) \land (\overline{x}_{4,3} \lor \overline{x}_{8,3}) \land (\overline{x}_{5,3} \lor \overline{x}_{6,3}) \land (\overline{x}_{5,3} \lor \overline{x}_{7,3}) \land (\overline{x}_{5,3} \lor \overline{x}_{8,3}) \land (\overline{x}_{6,3} \lor \overline{x}_{7,3}) \land (\overline{x}_{6,3} \lor \overline{x}_{8,3}) \land (\overline{x}_{7,3} \lor \overline{x}_{8,3}) \land (\overline{x}_{1,4} \lor \overline{x}_{2,4}) \land (\overline{x}_{1,4} \lor \overline{x}_{3,4}) \land (\overline{x}_{1,4} \lor \overline{x}_{4,4}) \land (\overline{x}_{1,4} \lor \overline{x}_{5,4}) \land (\overline{x}_{1,4} \lor \overline{x}_{6,4}) \land (\overline{x}_{1,4} \lor \overline{x}_{7,4}) \land (\overline{x}_{1,4} \lor \overline{x}_{8,4}) \land (\overline{x}_{2,4} \lor \overline{x}_{3,4}) \land (\overline{x}_{2,4} \lor \overline{x}_{4,4}) \land (\overline{x}_{2,4} \lor \overline{x}_{5,4})$

A Fundamental Theoretical Problem...

$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$

A Fundamental Theoretical Problem...

$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$

- Variables should be set to true ($= 1$) or false ($= 0$)

A Fundamental Theoretical Problem...

$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$

- Variables should be set to true $(= 1)$ or false $(= 0)$
- Constraint $(x \lor \overline{y} \lor z)$: means x or z should be true or y false

A Fundamental Theoretical Problem...

$$(u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w})$$

- Variables should be set to true $(= 1)$ or false $(= 0)$
- Constraint $(x \vee \overline{y} \vee z)$: means x or z should be true or y false
- $\wedge$ means all constraints should hold simultaneously

A Fundamental Theoretical Problem...

$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$

- Variables should be set to true $(= 1)$ or false $(= 0)$
- Constraint $(x \lor \overline{y} \lor z)$: means x or z should be true or y false
- $\land$ means all constraints should hold simultaneously

Is there a truth value assignment satisfying all constraints?

A Fundamental Theoretical Problem...

$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$

- Variables should be set to true ($= 1$) or false ($= 0$)
- Constraint $(x \lor \overline{y} \lor z)$: means x or z should be true or y false
- $\land$ means all constraints should hold simultaneously

Is there a truth value assignment satisfying all constraints?
Can computers solve this satisfiability (SAT) problem efficiently?

A Fundamental Theoretical Problem...

$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$

- Variables should be set to true $(= 1)$ or false $(= 0)$
- Constraint $(x \lor \overline{y} \lor z)$: means x or z should be true or y false
- $\land$ means all constraints should hold simultaneously

Is there a truth value assignment satisfying all constraints?
Can computers solve this satisfiability (SAT) problem efficiently?

- Mentioned already in Gödel's famous letter in 1956 to von Neumann (the "father of computer science")
- Intense research in theoretical computer science ever since early 1970s
- Now one of Millennium Prize Problems in mathematics

. . . with Huge Practical Implications

- Many problems can be encoded as logic formulas, e.g.:
  - hardware verification
  - software testing
  - artificial intelligence
  - cryptography
  - bioinformatics
  - et cetera. . .
- Leads to **humongous** formulas (100,000s or 1,000,000s of variables)
- Dramatic progress last 15–20 years on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But. . . There are also small formulas (just ∼100 variables) that are completely beyond reach of even the very best SAT solvers

Purpose of This Presentation

Explain how to solve SAT in linear time

Purpose of This Presentation

Explain how to solve SAT in linear time
(well, at least surprisingly often. . .)

Purpose of This Presentation

Explain how to solve SAT in linear time
(well, at least surprisingly often. . .)

Outline in a bit more detail:

- How do state-of-the-art SAT solvers work?*

Purpose of This Presentation

Explain how to solve SAT in linear time
(well, at least surprisingly often...)

Outline in a bit more detail:

- How do state-of-the-art SAT solvers work?*
- How to to analyze SAT solver performance?

Purpose of This Presentation

Explain how to solve SAT in linear time
(well, at least surprisingly often...)

Outline in a bit more detail:

- How do state-of-the-art SAT solvers work?*
- How to to analyze SAT solver performance?
- How to go beyond current state of the art?

Purpose of This Presentation

Explain how to solve SAT in linear time
(well, at least surprisingly often. . .)

Outline in a bit more detail:

- How do state-of-the-art SAT solvers work?*
- How to to analyze SAT solver performance?
- How to go beyond current state of the art?

(*) Obviously, can't give all details in 15 minutes, but aim to cover essentials

How (Not) to Solve the SAT Problem

- Let computer check all possible assignments! Isn't this exactly the kind of monotone routine work at which computers excel?

How (Not) to Solve the SAT Problem

- Let computer check all possible assignments! Isn't this exactly the kind of monotone routine work at which computers excel?
- But how many cases to check?

How (Not) to Solve the SAT Problem

- Let computer check all possible assignments! Isn't this exactly the kind of monotone routine work at which computers excel?
- But how many cases to check?
- Suppose formula has n variables

How (Not) to Solve the SAT Problem

- Let computer check all possible assignments! Isn't this exactly the kind of monotone routine work at which computers excel?
- But how many cases to check?
- Suppose formula has n variables
- Each variable can be true or false, so all in all get 2^n different cases

How (Not) to Solve the SAT Problem

- Let computer check all possible assignments! Isn't this exactly the kind of monotone routine work at which computers excel?
- But how many cases to check?
- Suppose formula has n variables
- Each variable can be true or false, so all in all get 2^n different cases
- If formula contains, say, one million variables, we get $2^{1,000,000}$ cases (a number with more than 300,000 digits)

How (Not) to Solve the SAT Problem

- Let computer check all possible assignments! Isn't this exactly the kind of monotone routine work at which computers excel?
- But how many cases to check?
- Suppose formula has n variables
- Each variable can be true or false, so all in all get 2^n different cases
- If formula contains, say, one million variables, we get $2^{1,000,000}$ cases (a number with more than 300,000 digits)

To understand how large this number is, consider that even if every atom in the known universe was a modern supercomputer running at full speed ever since the beginning of time some 13.7 billion years ago, all of them together would only have covered a completely negligible fraction of these cases by now. So we really would not have time to wait for them to finish...

Basic Idea Behind Modern SAT Solvers

Want more refined case analysis over variable assignments

Basic Idea Behind Modern SAT Solvers

Want more refined case analysis over variable assignments

DPLL method [DP60, DLL62]

- Assign values to variables (in some smart way)
- Backtrack when conflict with falsified clause

Basic Idea Behind Modern SAT Solvers

Want more refined case analysis over variable assignments

DPLL method [DP60, DLL62]

- Assign values to variables (in some smart way)
- Backtrack when conflict with falsified clause

Conflict-driven clause learning (CDCL) [MS96, BS97, MMZ$^+$01]

- Analyse conflicts in more detail
- More efficient backtracking
- Also let conflicts guide other heuristics

Basic Idea Behind Modern SAT Solvers

Want more refined case analysis over variable assignments

DPLL method [DP60, DLL62]

- Assign values to variables (in some smart way)
- Backtrack when conflict with falsified clause

Conflict-driven clause learning (CDCL) [MS96, BS97, MMZ$^+$01]

- Analyse conflicts in more detail
- More efficient backtracking
- Also let conflicts guide other heuristics

Variable Assignments

Two kinds of assignments — illustrate on our example formula:

$$(u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w})$$

Variable Assignments

Two kinds of assignments — illustrate on our example formula:

$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$

Decision

Free choice to assign value to variable

Notation $w \overset{\mathsf{d}}{=} 0$

Variable Assignments

Two kinds of assignments — illustrate on our example formula:

$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$

$w \stackrel{\mathsf{d}}{=} 0$

Decision

Free choice to assign value to variable

Notation $w \stackrel{\mathsf{d}}{=} 0$

Variable Assignments

Two kinds of assignments — illustrate on our example formula:

$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$

$w \stackrel{\mathsf{d}}{=} 0$

Decision

Free choice to assign value to variable

Notation $w \stackrel{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $w = 0$, clause $\overline{u} \lor w$ forces $u = 0$

Notation $u \stackrel{\overline{u} \lor w}{=} 0$

Variable Assignments

Two kinds of assignments — illustrate on our example formula:

$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$

$w \stackrel{\mathsf{d}}{=} 0$

$u \stackrel{\overline{u} \lor w}{=} 0$

Decision

Free choice to assign value to variable

Notation $w \stackrel{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $w = 0$, clause $\overline{u} \lor w$ forces $u = 0$

Notation $u \stackrel{\overline{u} \lor w}{=} 0$

Variable Assignments

Two kinds of assignments — illustrate on our example formula:

$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$

$w \stackrel{\mathsf{d}}{=} 0$

$u \stackrel{\overline{u} \lor w}{=} 0$

$x \stackrel{\mathsf{d}}{=} 0$

Decision

Free choice to assign value to variable

Notation $w \stackrel{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $w = 0$, clause $\overline{u} \lor w$ forces $u = 0$

Notation $u \stackrel{\overline{u} \lor w}{=} 0$

Always propagate if possible, otherwise decide

Until satisfying assignment or conflict clause

Variable Assignments

Two kinds of assignments — illustrate on our example formula:

$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$

$w \stackrel{\mathsf{d}}{=} 0$

$u \stackrel{\overline{u} \lor w}{=} 0$

$x \stackrel{\mathsf{d}}{=} 0$

$y \stackrel{u \lor x \lor y}{=} 1$

Decision

Free choice to assign value to variable

Notation $w \stackrel{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $w = 0$, clause $\overline{u} \lor w$ forces $u = 0$

Notation $u \stackrel{\overline{u} \lor w}{=} 0$

Always propagate if possible, otherwise decide

Until satisfying assignment or conflict clause

Variable Assignments

Two kinds of assignments — illustrate on our example formula:

$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$

$w \stackrel{\mathsf{d}}{=} 0$

$u \stackrel{\overline{u} \lor w}{=} 0$

$x \stackrel{\mathsf{d}}{=} 0$

$y \stackrel{u \lor x \lor y}{=} 1$

$z \stackrel{x \lor \overline{y} \lor z}{=} 1$

Decision

Free choice to assign value to variable

Notation $w \stackrel{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $w = 0$, clause $\overline{u} \lor w$ forces $u = 0$

Notation $u \stackrel{\overline{u} \lor w}{=} 0$

Always propagate if possible, otherwise decide

Until satisfying assignment or conflict clause

Variable Assignments

Two kinds of assignments — illustrate on our example formula:

$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$

$w \stackrel{\mathsf{d}}{=} 0$

$u \stackrel{\overline{u} \lor w}{=} 0$

$x \stackrel{\mathsf{d}}{=} 0$

$y \stackrel{u \lor x \lor y}{=} 1$

$z \stackrel{x \lor \overline{y} \lor z}{=} 1$

$\stackrel{\overline{y} \lor \overline{z}}{\bot}$

Decision
Free choice to assign value to variable

Notation $w \stackrel{\mathsf{d}}{=} 0$

Unit propagation
Forced choice to avoid falsifying clause
Given $w = 0$, clause $\overline{u} \lor w$ forces $u = 0$

Notation $u \stackrel{\overline{u} \lor w}{=} 0$

Always propagate if possible, otherwise decide
Until satisfying assignment or conflict clause

Conflict-Driven Clause Learning

Time to analyse this conflict!

$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$

$w \stackrel{\mathsf{d}}{=} 0$

$u \stackrel{\overline{u} \lor w}{=} 0$

$x \stackrel{\mathsf{d}}{=} 0$

$y \stackrel{u \lor x \lor y}{=} 1$

$z \stackrel{x \lor \overline{y} \lor z}{=} 1$

$\stackrel{\overline{y} \lor \overline{z}}{\bot}$

Conflict-Driven Clause Learning

Time to analyse this conflict!

$$(u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w})$$

$w \stackrel{\mathsf{d}}{=} 0$

$u \stackrel{\overline{u} \vee w}{=} 0$

$x \stackrel{\mathsf{d}}{=} 0$

$y \stackrel{u \vee x \vee y}{=} 1$

$z \stackrel{x \vee \overline{y} \vee z}{=} 1$

$\stackrel{\overline{y} \vee \overline{z}}{\bot}$

Could backtrack by flipping last decision

Conflict-Driven Clause Learning

Time to analyse this conflict!

$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$

$w \stackrel{\mathsf{d}}{=} 0$

$u \stackrel{\overline{u} \lor w}{=} 0$

$x \stackrel{\mathsf{d}}{=} 0$

$y \stackrel{u \lor x \lor y}{=} 1$

$z \stackrel{x \lor \overline{y} \lor z}{=} 1$

$\stackrel{\overline{y} \lor \overline{z}}{\bot}$

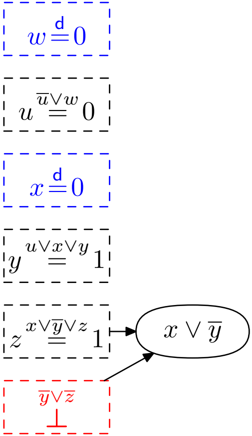
Could backtrack by flipping last decision

But want to learn from conflict and cut away as much of search space as possible

Conflict-Driven Clause Learning

Time to analyse this conflict!

$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Could backtrack by flipping last decision

But want to learn from conflict and cut away as much of search space as possible

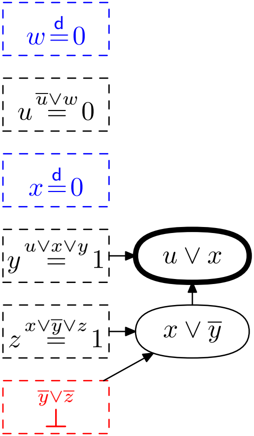
Case analysis over z for last two clauses:

- $x \lor \overline{y} \lor z$ wants $z = 1$
- $\overline{y} \lor \overline{z}$ wants $z = 0$
- Merge & remove z — must satisfy $x \lor \overline{y}$

Conflict-Driven Clause Learning

Time to analyse this conflict!

$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Could backtrack by flipping last decision

But want to learn from conflict and cut away as much of search space as possible

Case analysis over z for last two clauses:

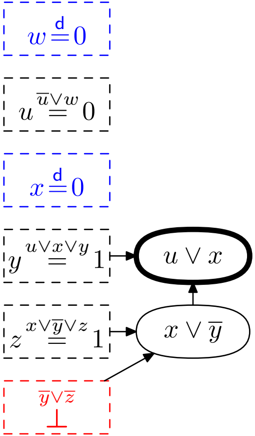
- $x \lor \overline{y} \lor z$ wants $z = 1$
- $\overline{y} \lor \overline{z}$ wants $z = 0$
- Merge & remove z — must satisfy $x \lor \overline{y}$

Repeat until only 1 variable after last decision — learn that clause and backjump

Complete Example of CDCL Execution

Backjump: roll back max # assignments so that last variable still flips

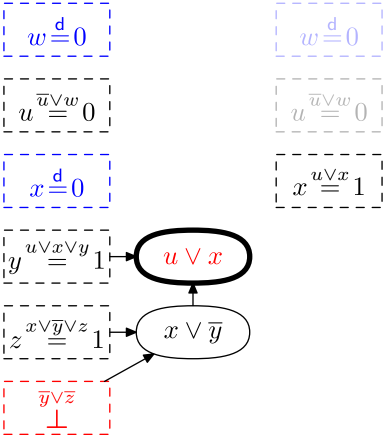
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max # assignments so that last variable still flips

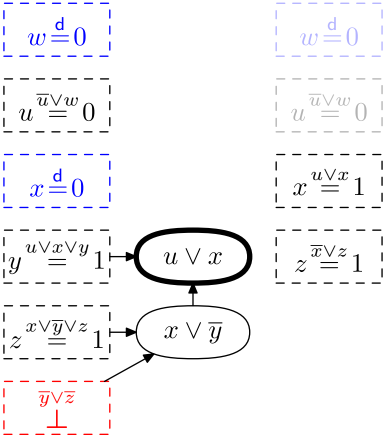
$(u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max # assignments so that last variable still flips

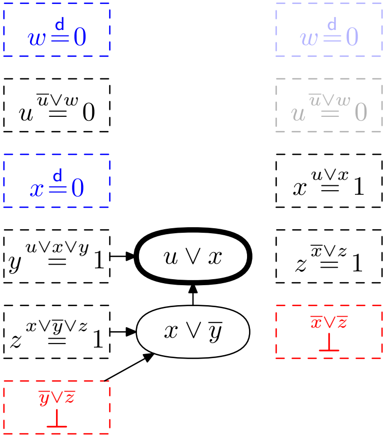
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max # assignments so that last variable still flips

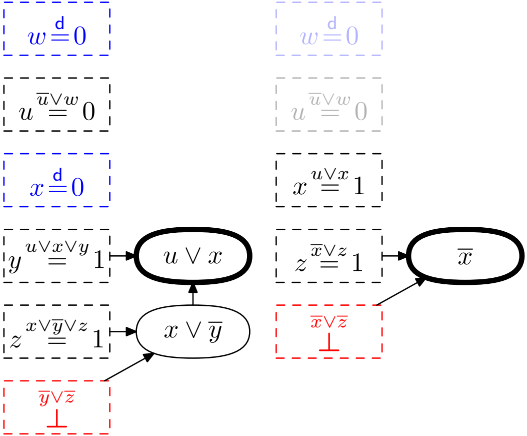
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max # assignments so that last variable still flips

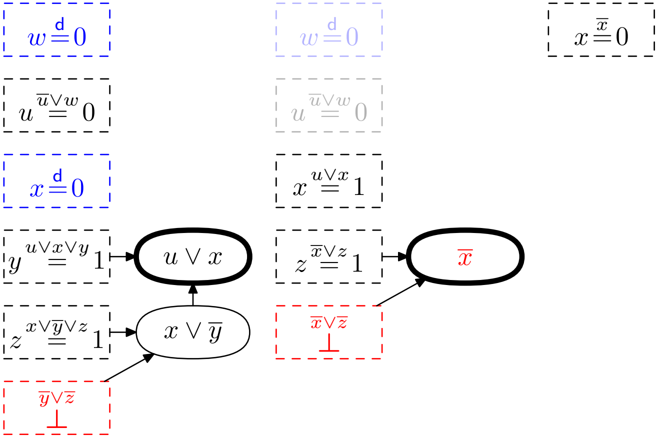
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max # assignments so that last variable still flips

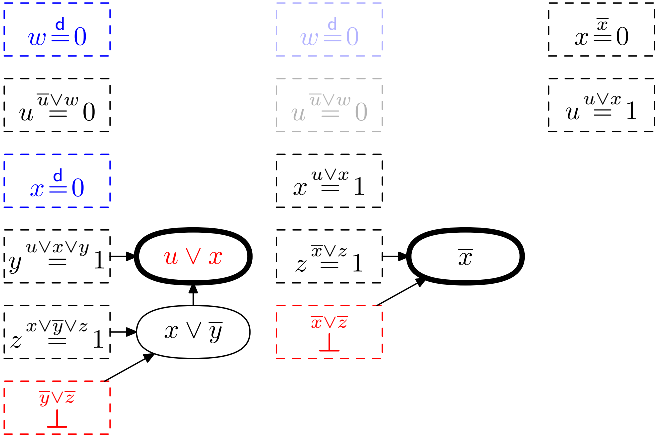
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max # assignments so that last variable still flips

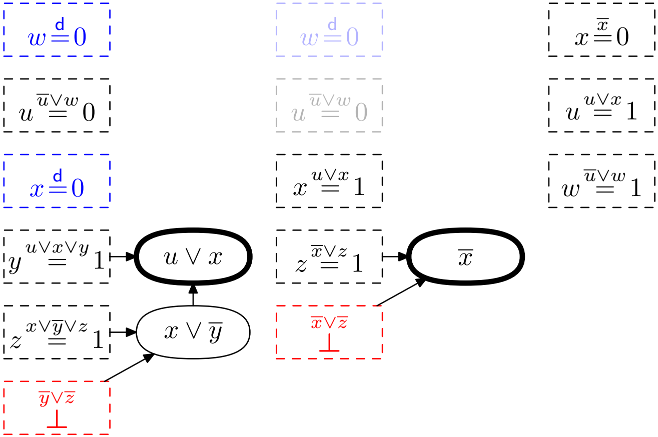
$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$



Complete Example of CDCL Execution

Backjump: roll back max # assignments so that last variable still flips

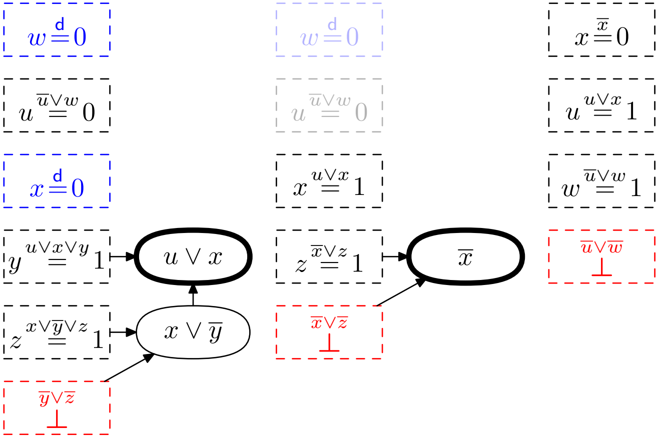
$(u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max # assignments so that last variable still flips

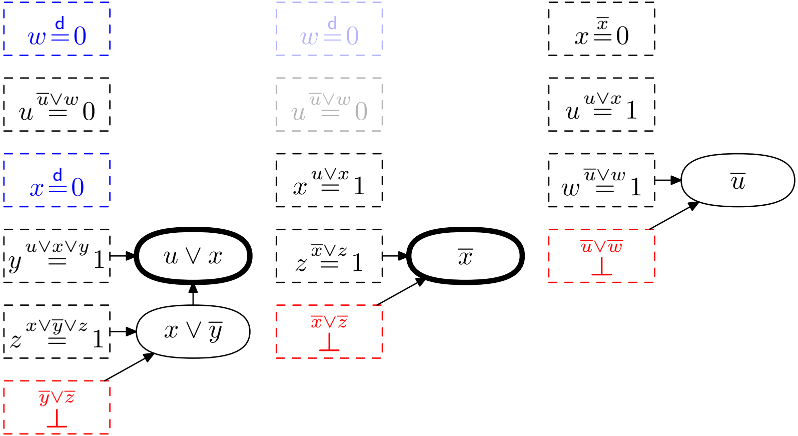
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max # assignments so that last variable still flips

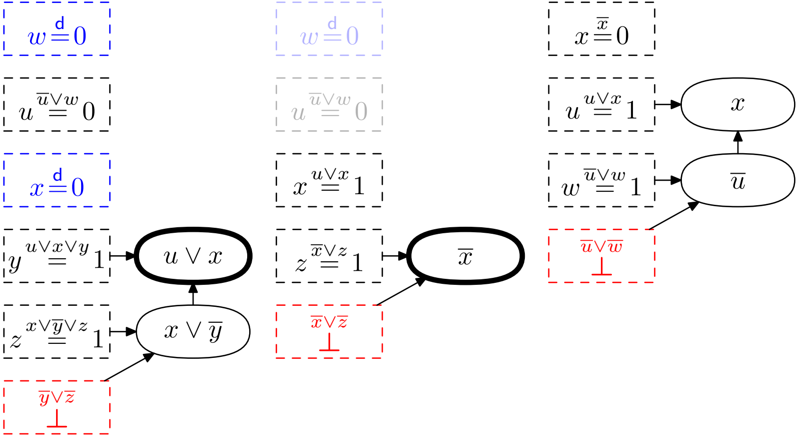
$(u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max # assignments so that last variable still flips

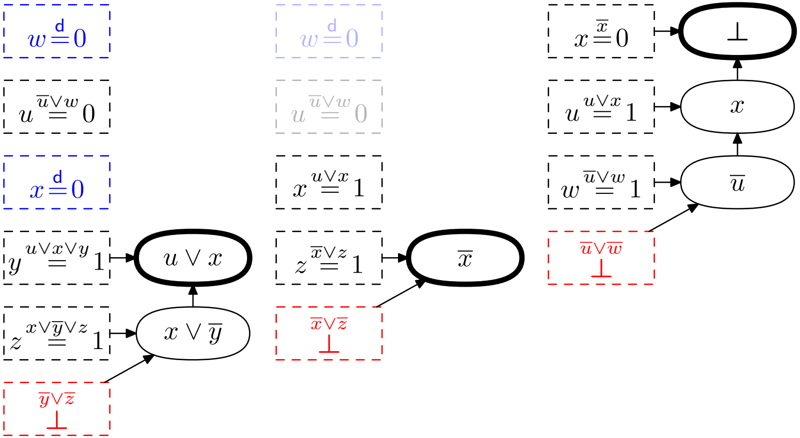
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max # assignments so that last variable still flips

$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



State-of-the-Art SAT Solving in One Slide

repeat
  if current assignment falsifies clause
    if no decisions made
      terminate with output `UNSATISFIABLE`
    apply learning scheme to add new clause & backjump
  else if all variables assigned
    terminate with output `SATISFIABLE`
  else if exists unit clause C propagating x to value $b \in \{0, 1\}$
    add propagated assignment $x \overset{C}{=} b$
  else if time to restart
    undo all variable assignments
  else
    if time for clause database reduction
      erase (roughly) half of learned clauses in memory
    use decision scheme to add assignment $x \overset{\mathsf{d}}{=} b$

CDCL Analysis and the Resolution Proof System

How to analyse CDCL performance?

Many intricate, hard-to-understand heuristics

Focus instead on underlying method of reasoning

CDCL Analysis and the Resolution Proof System

How to analyse CDCL performance?

Many intricate, hard-to-understand heuristics

Focus instead on underlying method of reasoning

Resolution proof system

- Start with clauses of formula
- Derive new clauses by resolution rule

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D}$$

- Done when contradiction $\perp$ in form of empty clause derived

CDCL Analysis and the Resolution Proof System

How to analyse CDCL performance?
Many intricate, hard-to-understand heuristics
Focus instead on underlying method of reasoning

Resolution proof system

- Start with clauses of formula
- Derive new clauses by resolution rule

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D}$$

- Done when contradiction $\perp$ in form of empty clause derived

When run on unsatisfiable formula, CDCL generates resolution proof*
So lower bounds on proof size $\Rightarrow$ lower bounds on running time

CDCL Analysis and the Resolution Proof System

How to analyse CDCL performance?
Many intricate, hard-to-understand heuristics
Focus instead on underlying method of reasoning

Resolution proof system

- Start with clauses of formula
- Derive new clauses by resolution rule

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D}$$

- Done when contradiction $\perp$ in form of empty clause derived

When run on unsatisfiable formula, CDCL generates resolution proof*
So lower bounds on proof size $\Rightarrow$ lower bounds on running time

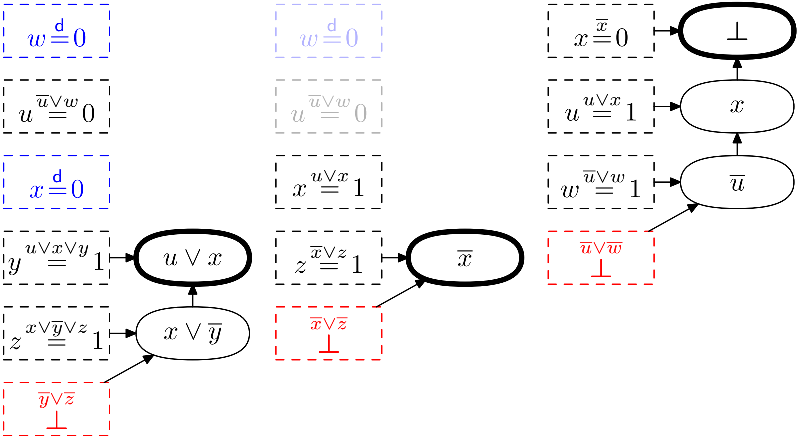
(*) Ignores preprocessing, but we don't have time to go into this

Resolution Proofs from CDCL Executions

Obtain resolution proof. . .

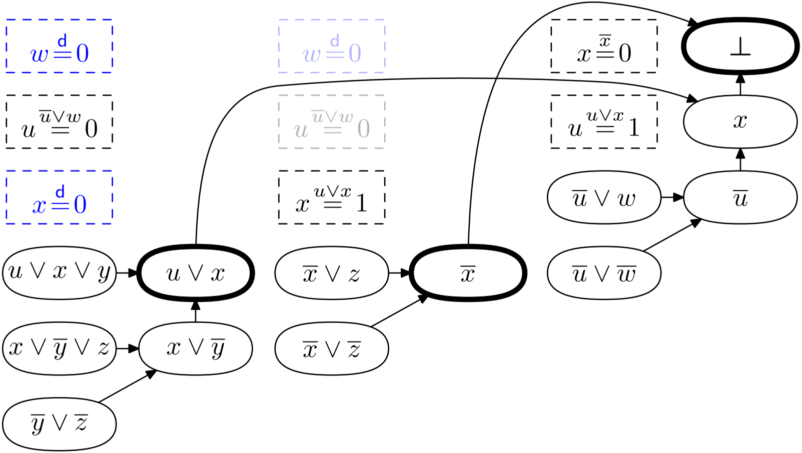
Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution. . .



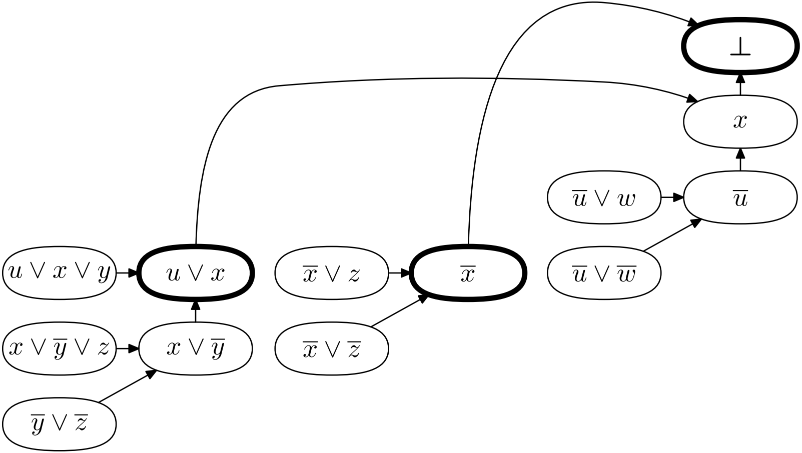
Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution by stringing together conflict analyses:



Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution by stringing together conflict analyses:



Conclusions and Open Problems

Current state of affairs

- Modern solvers perform amazingly well ("SAT is easy in practice")
- Very poor theoretical understanding:
  - Why do heuristics work?
  - Why are applied instances easy?
- Paradox: resolution quite weak proof system; many strong lower bounds for "obvious" formulas, e.g., [Hak85, Urq87, BW01, MN14]

Conclusions and Open Problems

Current state of affairs

- Modern solvers perform amazingly well ("SAT is easy in practice")
- Very poor theoretical understanding:
  - Why do heuristics work?
  - Why are applied instances easy?
- Paradox: resolution quite weak proof system; many strong lower bounds for "obvious" formulas, e.g., [Hak85, Urq87, BW01, MN14]

Directions for future work

- Develop better understanding of state-of-the-art solvers
- Improve heuristics (maybe thanks to better understanding)
- Explore stronger reasoning methods (potential exponential speed-up)
  - Algebra: Gröbner basis computations
  - Geometry: Integer linear programming

Conclusions and Open Problems

Current state of affairs

- Modern solvers perform amazingly well ("SAT is easy in practice")
- Very poor theoretical understanding:
  - Why do heuristics work?
  - Why are applied instances easy?
- Paradox: resolution quite weak proof system; many strong lower bounds for "obvious" formulas, e.g., [Hak85, Urq87, BW01, MN14]

Directions for future work

- Develop better understanding of state-of-the-art solvers
- Improve heuristics (maybe thanks to better understanding)
- Explore stronger reasoning methods (potential exponential speed-up)
  - Algebra: Gröbner basis computations
  - Geometry: Integer linear programming

Thank you for your attention!

References I

[BS97] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

[BW01] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, March 2001. Preliminary version in *STOC '99*.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[Hak85] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, August 1985.

[MMZ$^+$01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

References II

[MN14] Mladen Mikša and Jakob Nordström. Long proofs of (seemingly) simple formulas. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 121–137. Springer, July 2014.

[MS96] João P. Marques-Silva and Karem A. Sakallah. GRASP—a new search algorithm for satisfiability. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '96)*, pages 220–227, November 1996.

[Urq87] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, January 1987.