LICS 2021 Lecture 11: Boolean Satisfiability (SAT) Solving

Jakob Nordström

University of Copenhagen and Lund University

January 6, 2022

Three Simple Problems. . .

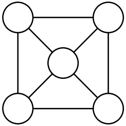
Colouring

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

Three Simple Problems...

Colouring

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

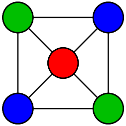


3-colouring?

Three Simple Problems...

Colouring

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

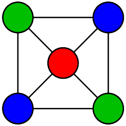


3-colouring? Yes

Three Simple Problems. . .

Colouring

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?



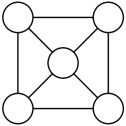
3-colouring? Yes, but no 2-colouring

Three Simple Problems. . .

Clique

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

Three Simple Problems...

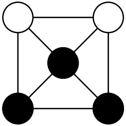


3-clique?

Clique

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

Three Simple Problems. . .

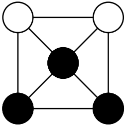


3-clique? Yes

CLIQUE

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

Three Simple Problems...



3-clique? Yes, but no 4-clique

Clique

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

Three Simple Problems...

COLOURING

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

CLIQUE

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

SAT

Given propositional logic formula, is there a satisfying assignment?

Three Simple Problems...

COLOURING

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

CLIQUE

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

SAT

Given propositional logic formula, is there a satisfying assignment?

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

Three Simple Problems. . .

COLOURING

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

CLIQUE

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

SAT

Given propositional logic formula, is there a satisfying assignment?

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

- Variables should be set to **true** or **false**
- Constraint $(x \lor \neg y \lor z)$: 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?

Three Simple Problems. . .

COLOURING

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

CLIQUE

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

SAT

Given propositional logic formula, is there a satisfying assignment?

COLOURING:	frequency allocation for mobile base stations
CLIQUE:	bioinformatics, computational chemistry
SAT:	easily models these and many other problems

. . . with Huge Practical Implications

- Some more examples of problems that can be encoded as propositional logic formulas:
  - computer hardware verification
  - computer software testing
  - artificial intelligence
  - cryptography
  - bioinformatics
  - et cetera. . .

. . . with Huge Practical Implications

- Some more examples of problems that can be encoded as propositional logic formulas:
  - computer hardware verification
  - computer software testing
  - artificial intelligence
  - cryptography
  - bioinformatics
  - et cetera. . .
- Leads to **humongous** formulas (100,000s or even 1,000,000s of variables)

. . . with Huge Practical Implications

- Some more examples of problems that can be encoded as propositional logic formulas:
  - computer hardware verification
  - computer software testing
  - artificial intelligence
  - cryptography
  - bioinformatics
  - et cetera. . .
- Leads to **humongous** formulas (100,000s or even 1,000,000s of variables)
- Can we use computers to solve these problems efficiently?

. . . with Huge Practical Implications

- Some more examples of problems that can be encoded as propositional logic formulas:
  - computer hardware verification
  - computer software testing
  - artificial intelligence
  - cryptography
  - bioinformatics
  - et cetera. . .
- Leads to **humongous** formulas (100,000s or even 1,000,000s of variables)
- Can we use computers to solve these problems efficiently?
- Question mentioned already in Gödel's famous letter in 1956 to von Neumann (the "father of computer science")

. . . with Huge Practical Implications

- Some more examples of problems that can be encoded as propositional logic formulas:
  - computer hardware verification
  - computer software testing
  - artificial intelligence
  - cryptography
  - bioinformatics
  - et cetera. . .
- Leads to **humongous** formulas (100,000s or even 1,000,000s of variables)
- Can we use computers to solve these problems efficiently?
- Question mentioned already in Gödel's famous letter in 1956 to von Neumann (the "father of computer science")
- Topic of intense research in computer science ever since 1960s

Solving Logic Formulas in Practice

- Dramatic progress last 20–25 years on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems

Solving Logic Formulas in Practice

- Dramatic progress last 20–25 years on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But... There are also small formulas (just $\sim$100 variables) that are completely beyond reach of even the very best SAT solvers

Solving Logic Formulas in Practice

- Dramatic progress last 20–25 years on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But... There are also small formulas (just $\sim$100 variables) that are completely beyond reach of even the very best SAT solvers
- Best known SAT solving algorithms based on Davis-Putnam-Logemann-Loveland or DPLL method from early 1960s (although with many clever optimizations)

Solving Logic Formulas in Practice

- Dramatic progress last 20–25 years on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But... There are also small formulas (just $\sim$100 variables) that are completely beyond reach of even the very best SAT solvers
- Best known SAT solving algorithms based on Davis-Putnam-Logemann-Loveland or DPLL method from early 1960s (although with many clever optimizations)
- Some natural questions:
  - How do these SAT solvers work?

Solving Logic Formulas in Practice

- Dramatic progress last 20–25 years on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But... There are also small formulas (just $\sim$100 variables) that are completely beyond reach of even the very best SAT solvers
- Best known SAT solving algorithms based on Davis-Putnam-Logemann-Loveland or DPLL method from early 1960s (although with many clever optimizations)
- Some natural questions:
  - How do these SAT solvers work?
  - How can they be so good in practice?

Solving Logic Formulas in Practice

- Dramatic progress last 20–25 years on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But... There are also small formulas (just $\sim$100 variables) that are completely beyond reach of even the very best SAT solvers
- Best known SAT solving algorithms based on Davis-Putnam-Logemann-Loveland or DPLL method from early 1960s (although with many clever optimizations)
- Some natural questions:
  - How do these SAT solvers work?
  - How can they be so good in practice?
  - When they fail to be efficient, can we understand why?

Solving Logic Formulas in Practice

- Dramatic progress last 20–25 years on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But... There are also small formulas (just $\sim$100 variables) that are completely beyond reach of even the very best SAT solvers
- Best known SAT solving algorithms based on Davis-Putnam-Logemann-Loveland or DPLL method from early 1960s (although with many clever optimizations)
- Some natural questions:
  - How do these SAT solvers work?
  - How can they be so good in practice?
  - When they fail to be efficient, can we understand why?
  - It's 2022 now — can we go beyond techniques from 1960s?

Plan for This Lecture

What we will cover in this lecture:

Plan for This Lecture

What we will cover in this lecture:

- Define more precisely the computational problem

Plan for This Lecture

What we will cover in this lecture:

- Define more precisely the computational problem
- Give (simplified) description of how modern SAT solvers work

Plan for This Lecture

What we will cover in this lecture:

- Define more precisely the computational problem
- Give (simplified) description of how modern SAT solvers work
- Present tools to analyze SAT solver performance

Plan for This Lecture

What we will cover in this lecture:

- Define more precisely the computational problem
- Give (simplified) description of how modern SAT solvers work
- Present tools to analyze SAT solver performance
- Discuss possible ways to go beyond current state of the art

Plan for This Lecture

What we will cover in this lecture:

- Define more precisely the computational problem
- Give (simplified) description of how modern SAT solvers work
- Present tools to analyze SAT solver performance
- Discuss possible ways to go beyond current state of the art

. . . And in the process also touch on some of the research being done in the Mathematical Insights into Algorithms for Optimization (MIAO) group

Formal Description of SAT Problem

- Variable x: takes value 1 (**true**) or 0 (**false**)
- Literal ℓ: variable x or its negation $\overline{x}$ (write $\overline{x}$ instead of $\neg x$)
- Clause $C = \ell_1 \vee \cdots \vee \ell_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- Conjunctive normal form (CNF) formula $F = C_1 \wedge \cdots \wedge C_m$: conjunction of clauses

Formal Description of SAT Problem

- Variable x: takes value 1 (**true**) or 0 (**false**)
- Literal ℓ: variable x or its negation $\overline{x}$ (write $\overline{x}$ instead of $\neg x$)
- Clause $C = \ell_1 \vee \cdots \vee \ell_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- Conjunctive normal form (CNF) formula $F = C_1 \wedge \cdots \wedge C_m$: conjunction of clauses

The SATISFIABILITY (or just SAT) Problem

Given a CNF formula F, is it satisfiable?

Formal Description of SAT Problem

- Variable x: takes value 1 (**true**) or 0 (**false**)
- Literal ℓ: variable x or its negation $\overline{x}$ (write $\overline{x}$ instead of $\neg x$)
- Clause $C = \ell_1 \lor \cdots \lor \ell_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- Conjunctive normal form (CNF) formula $F = C_1 \land \cdots \land C_m$: conjunction of clauses

The SATISFIABILITY (or just SAT) Problem

Given a CNF formula F, is it satisfiable?

For instance, what about our example formula?

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

How 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 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 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 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 either true or false, so all in all get 2^n different cases

How 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 either 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 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 either 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 that had been 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...

An Interesting Feature of the SAT Problem

- Deciding whether a satisfying assignment exists may take a long time

An Interesting Feature of the SAT Problem

- Deciding whether a satisfying assignment exists may take a long time
- But if you happen to know a satisfying assignment, easy to convince someone else that formula is satisfiable

An Interesting Feature of the SAT Problem

- Deciding whether a satisfying assignment exists may take a long time
- But if you happen to know a satisfying assignment, easy to convince someone else that formula is satisfiable
- How? Just give assignment — can be verified in linear time

An Interesting Feature of the SAT Problem

- Deciding whether a satisfying assignment exists may take a long time
- But if you happen to know a satisfying assignment, easy to convince someone else that formula is satisfiable
- How? Just give assignment — can be verified in linear time
- So SAT problem might seem hard to solve, but verifying a solution is easy (not all problems have this property — how do you verify a winning position in chess?)

An Interesting Feature of the SAT Problem

- Deciding whether a satisfying assignment exists may take a long time
- But if you happen to know a satisfying assignment, easy to convince someone else that formula is satisfiable
- How? Just give assignment — can be verified in linear time
- So SAT problem might seem hard to solve, but verifying a solution is easy (not all problems have this property — how do you verify a winning position in chess?)
- The family of problems for which solutions are easy to check have a name: NP

How to Solve the SAT Problem, Take 2

- SAT problem can be used to describe any problem in NP — it is NP-complete [Coo71, Lev73]

How to Solve the SAT Problem, Take 2

- SAT problem can be used to describe any problem in NP — it is NP-complete [Coo71, Lev73]
- If you can solve SAT efficiently, then you can solve any problem in NP efficiently (this is why SAT is so useful)

How to Solve the SAT Problem, Take 2

- SAT problem can be used to describe any problem in NP — it is NP-complete [Coo71, Lev73]
- If you can solve SAT efficiently, then you can solve any problem in NP efficiently (this is why SAT is so useful)
- So how hard is it to solve SAT? (Ok, brute force didn't work, but it usually doesn't — maybe can do something smarter?)

How to Solve the SAT Problem, Take 2

- SAT problem can be used to describe any problem in NP — it is NP-complete [Coo71, Lev73]
- If you can solve SAT efficiently, then you can solve any problem in NP efficiently (this is why SAT is so useful)
- So how hard is it to solve SAT? (Ok, brute force didn't work, but it usually doesn't — maybe can do something smarter?)
- We don't know

How to Solve the SAT Problem, Take 2

- SAT problem can be used to describe any problem in NP — it is NP-complete [Coo71, Lev73]
- If you can solve SAT efficiently, then you can solve any problem in NP efficiently (this is why SAT is so useful)
- So how hard is it to solve SAT? (Ok, brute force didn't work, but it usually doesn't — maybe can do something smarter?)
- We don't know
- This one of the million-dollar "Millennium Prize Problems" posed as the main challenges for mathematics in the new millennium

How to Solve the SAT Problem, Take 2

- SAT problem can be used to describe any problem in NP — it is NP-complete [Coo71, Lev73]
- If you can solve SAT efficiently, then you can solve any problem in NP efficiently (this is why SAT is so useful)
- So how hard is it to solve SAT? (Ok, brute force didn't work, but it usually doesn't — maybe can do something smarter?)
- We don't know
- This one of the million-dollar "Millennium Prize Problems" posed as the main challenges for mathematics in the new millennium
- Widely believe to be impossible to solve efficiently on computer in the worst case, but we really don't know

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

- If F contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

- If F contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
- If F contains no clauses, report "satisfiable" and terminate

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

- If F contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
- If F contains no clauses, report "satisfiable" and terminate
- Otherwise pick some variable x in F

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

- If F contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
- If F contains no clauses, report "satisfiable" and terminate
- Otherwise pick some variable x in F
- Set $x = 0$, simplify F and make recursive call

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

- If F contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
- If F contains no clauses, report "satisfiable" and terminate
- Otherwise pick some variable x in F
- Set $x = 0$, simplify F and make recursive call
- Set $x = 1$, simplify F and make recursive call

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

- If F contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
- If F contains no clauses, report "satisfiable" and terminate
- Otherwise pick some variable x in F
- Set $x = 0$, simplify F and make recursive call
- Set $x = 1$, simplify F and make recursive call
- If result in both cases "unsatisfiable", then report "unsatisfiable" and return

A DPLL Toy Example

$$\begin{aligned} F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ & \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w}) \end{aligned}$$

A DPLL Toy Example

$$\begin{aligned} F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ & \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals

A DPLL Toy Example

$$\begin{aligned} F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ & \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals

x

A DPLL Toy Example

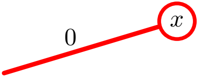
$$\begin{aligned} F = \quad & (\quad z) \land (y \lor \overline{z}) \land (\quad \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ & \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

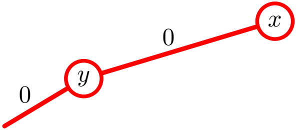
$$F = (\quad z) \land (\quad \overline{z}) \land (\quad \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

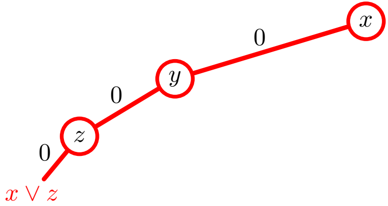
$$F = (x \lor z) \land (\quad \overline{z}) \land (\quad \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

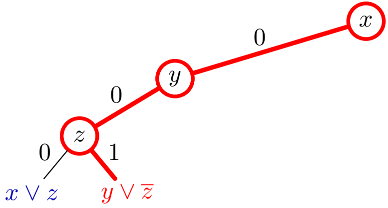
$$F = (\quad z) \land (y \lor \overline{z}) \land (\quad \overline{y} \lor u) \land (\overline{y} \lor \overline{u})$$
$$\land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

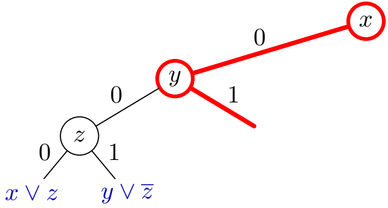
$$F = (\quad z) \land (y \lor \overline{z}) \land (\quad u) \land (\quad \overline{u}) \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

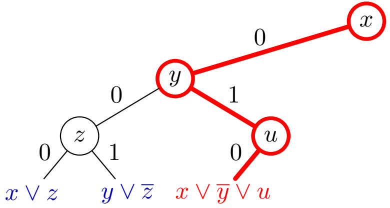
$$F = (\quad z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\quad \overline{u}) \land (\quad v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

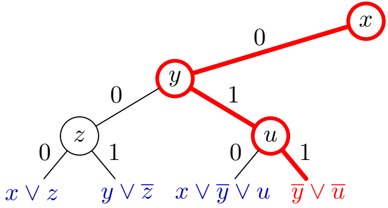
$$F = (\quad z) \land (y \lor \overline{z}) \land (\quad u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\quad w) \land (\overline{x} \lor \quad \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

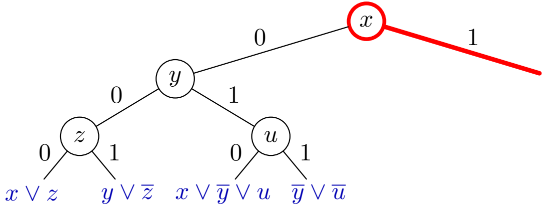
$$\begin{aligned} F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ & \land (u \lor v) \land (\overline{v}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

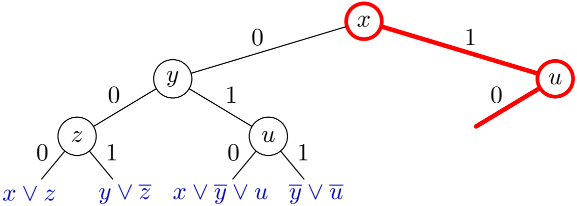
$$F = (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \quad) \land (\overline{y} \lor \overline{u}) \land (\quad v) \land (\quad \overline{v}) \land (\overline{u} \lor w) \land (\quad \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

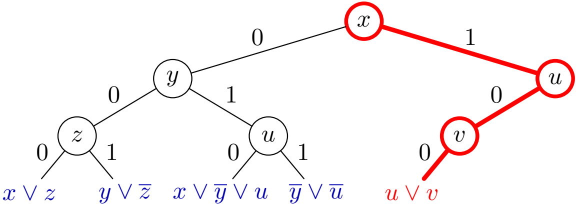
$$
\begin{aligned}
F = \quad & (x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \qquad) \wedge (\overline{y} \vee \overline{u}) \\
& \wedge (u \vee v) \wedge (\qquad \overline{v}) \wedge (\overline{u} \vee w) \wedge (\qquad \overline{u} \vee \overline{w})
\end{aligned}
$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

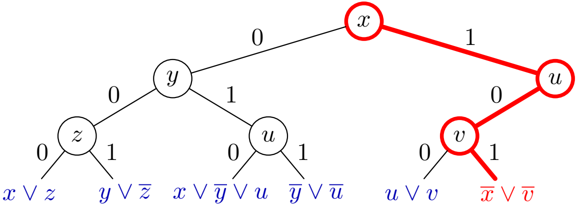
$$F = (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \quad) \land (\overline{y} \lor \overline{u})$$
$$\land \; (\quad v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\quad \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

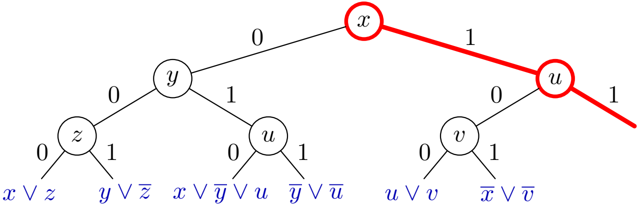
$$
\begin{aligned}
F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\
& \land (u \lor v) \land (\quad \overline{v}) \land (\quad w) \land (\quad \overline{w})
\end{aligned}
$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

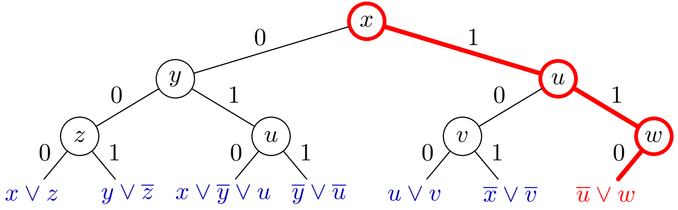
$$F = (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\overline{v}) \land (\overline{u} \lor w) \land (\overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

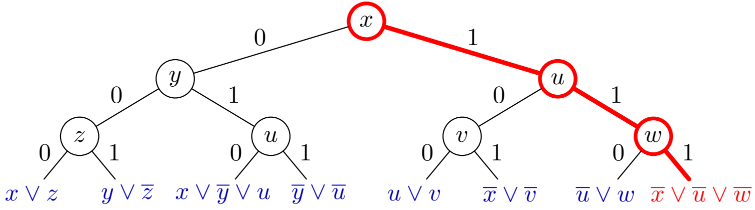
$$\begin{aligned} F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ & \land (u \lor v) \land (\qquad \overline{v}) \land (\qquad w) \land (\overline{x} \lor \overline{u} \lor \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

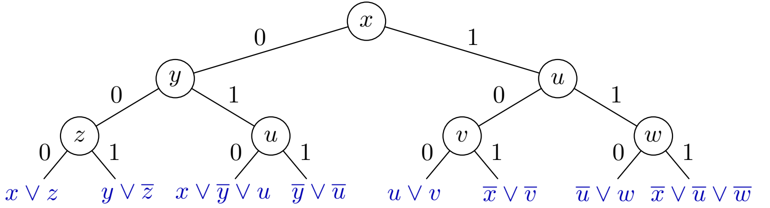
$$\begin{aligned} F = \quad & (x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u}) \\ & \wedge (u \vee v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



State-of-the-art SAT solvers: Ingredients

Many more ingredients in modern conflict-driven clause learning (CDCL) SAT solvers (as pioneered in [MS99, MMZ^{+}01]), e.g.:

- Branching or decision heuristic (choice of pivot variables crucial)
- When reaching leaf, compute explanation for conflict and add to formula as new clause (clause learning)
- Every once in a while, restart from beginning (but save computed info)

Let us discuss these ingredients

Variable Assignment Heuristics

Unit propagation

- Suppose current assignment ρ falsifies all literals in $C = \ell_1 \lor \ell_2 \lor \cdots \lor \ell_k$ except one (say ℓ_k) — C is unit under ρ
- Then ℓ_k has to be true, so set it to true
- Known as unit progagation or Boolean constraint progagation
- Always propagate if possible — in modern solvers aim for 99% of assignments being unit propagations

Variable Assignment Heuristics

Unit propagation

- Suppose current assignment ρ falsifies all literals in $C = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$ except one (say ℓ_k) — C is unit under ρ
- Then ℓ_k has to be true, so set it to true
- Known as unit progagation or Boolean constraint progagation
- Always propagate if possible — in modern solvers aim for 99% of assignments being unit propagations

VSIDS (Variable state independent decaying sum)

- When backtracking, score $+1$ for variables "causing conflict"
- Also multiply all scores with factor $\kappa < 1$ — exponential filter rewarding variables involved in recent conflicts
- When no propagations, decide on variable with highest score

Clause Learning

- At conflict, want to add clause avoiding same part of search tree being explored again

Clause Learning

- At conflict, want to add clause avoiding same part of search tree being explored again
- Suppose we can compute that decisions $x = 1$, $y = 0$, $z = 1$ responsible for conflict

Clause Learning

- At conflict, want to add clause avoiding same part of search tree being explored again
- Suppose we can compute that decisions $x = 1$, $y = 0$, $z = 1$ responsible for conflict
- Then can add $\overline{x} \lor y \lor \overline{z}$ to avoid these decisions being made again — decision learning scheme

Clause Learning

- At conflict, want to add clause avoiding same part of search tree being explored again
- Suppose we can compute that decisions $x = 1$, $y = 0$, $z = 1$ responsible for conflict
- Then can add $\overline{x} \lor y \lor \overline{z}$ to avoid these decisions being made again — decision learning scheme
- In practice, more advanced learning schemes

Clause Learning

- At conflict, want to add clause avoiding same part of search tree being explored again
- Suppose we can compute that decisions $x = 1$, $y = 0$, $z = 1$ responsible for conflict
- Then can add $\overline{x} \vee y \vee \overline{z}$ to avoid these decisions being made again — decision learning scheme
- In practice, more advanced learning schemes
- Derive new clause from clauses unit propagating on the way to conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

Decision

Free choice to assign value to variable

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

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

Decision

Free choice to assign value to variable

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

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

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

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

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

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \lor r}{=} 1$

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

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

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \vee r}{=} 1$

$w \stackrel{\overline{r} \vee w}{=} 1$

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

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

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \vee r}{=} 1$

$w \stackrel{\overline{r} \vee w}{=} 1$

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

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

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \lor r}{=} 1$

$w \stackrel{\overline{r} \lor w}{=} 1$

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

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

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

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

$r \stackrel{q \vee r}{=} 1$

$w \stackrel{\overline{r} \vee w}{=} 1$

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

$\boxed{p \overset{\mathsf{d}}{=} 0}$

$u \overset{p \lor \overline{u}}{=} 0$

$\boxed{q \overset{\mathsf{d}}{=} 0}$

$r \overset{q \lor r}{=} 1$

$w \overset{\overline{r} \lor w}{=} 1$

$\boxed{x \overset{\mathsf{d}}{=} 0}$

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

Notation $u \overset{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$
$u \stackrel{p \vee \overline{u}}{=} 0$
decision level 1

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$
$r \stackrel{q \vee r}{=} 1$
$w \stackrel{\overline{r} \vee w}{=} 1$
decision level 2

$\boxed{x \stackrel{\mathsf{d}}{=} 0}$
$y \stackrel{u \vee x \vee y}{=} 1$
$z \stackrel{x \vee \overline{y} \vee z}{=} 1$
$\stackrel{\overline{y} \vee \overline{z}}{\perp}$
decision level 3

Decision
Free choice to assign value to variable

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

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

Notation $u \stackrel{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, else decide
Add to assignment trail
Until satisfying assignment or conflict

Conflict Analysis

Time to analyse this conflict and learn from it!

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

$$\left.\begin{array}{l} \boxed{p \stackrel{\mathsf{d}}{=} 0} \\ u \stackrel{p \vee \overline{u}}{=} 0 \end{array}\right\} \text{decision level } 1$$

$$\left.\begin{array}{l} \boxed{q \stackrel{\mathsf{d}}{=} 0} \\ r \stackrel{q \vee r}{=} 1 \\ w \stackrel{\overline{r} \vee w}{=} 1 \end{array}\right\} \text{decision level } 2$$

$$\left.\begin{array}{l} \boxed{x \stackrel{\mathsf{d}}{=} 0} \\ y \stackrel{u \vee x \vee y}{=} 1 \\ z \stackrel{x \vee \overline{y} \vee z}{=} 1 \\ \stackrel{\overline{y} \vee \overline{z}}{\bot} \end{array}\right\} \text{decision level } 3$$

Conflict Analysis

Time to analyse this conflict and learn from it!

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

$p \overset{\mathsf{d}}{=} 0$

$u \overset{p \lor \overline{u}}{=} 0$

decision level 1

$q \overset{\mathsf{d}}{=} 0$

$r \overset{q \lor r}{=} 1$

$w \overset{\overline{r} \lor w}{=} 1$

decision level 2

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

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

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

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

decision level 3

Could backtrack by removing last decision level & flipping last decision

Conflict Analysis

Time to analyse this conflict and learn from it!

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

$p \stackrel{\mathsf{d}}{=} 0$
$u \stackrel{p \lor \overline{u}}{=} 0$
} decision level 1

$q \stackrel{\mathsf{d}}{=} 0$
$r \stackrel{q \lor r}{=} 1$
$w \stackrel{\overline{r} \lor w}{=} 1$
} decision level 2

$x \stackrel{\mathsf{d}}{=} 0$
$y \stackrel{u \lor x \lor y}{=} 1$
$z \stackrel{x \lor \overline{y} \lor z}{=} 1$
$\stackrel{\overline{y} \lor \overline{z}}{\bot}$
} decision level 3

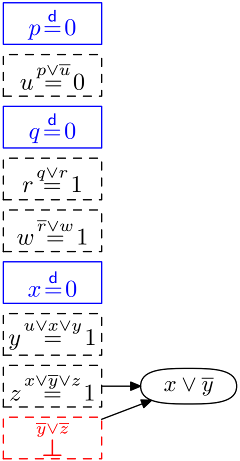
Could backtrack by removing last decision level & flipping last decision

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

Conflict Analysis

Time to analyse this conflict and learn from it!

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



Could backtrack by removing last decision level & flipping last decision

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

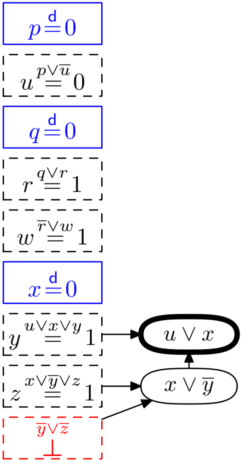
Case analysis over z for last two clauses:

- $x \vee \overline{y} \vee z$ wants $z = 1$
- $\overline{y} \vee \overline{z}$ wants $z = 0$
- Resolve clauses by merging them & removing z — must satisfy $x \vee \overline{y}$

Conflict Analysis

Time to analyse this conflict and learn from it!

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



Could backtrack by removing last decision level & 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:

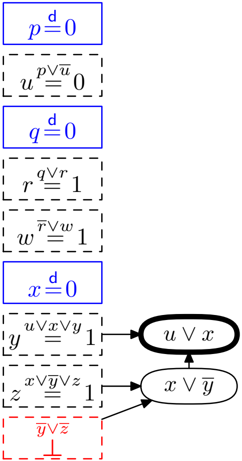
- $x \lor \overline{y} \lor z$ wants $z = 1$
- $\overline{y} \lor \overline{z}$ wants $z = 0$
- Resolve clauses by merging them & removing z — must satisfy $x \lor \overline{y}$

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

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

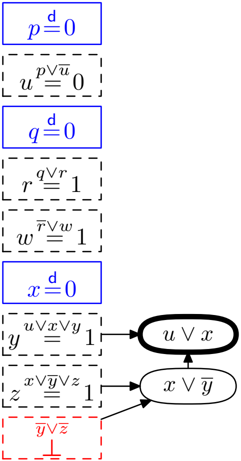
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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

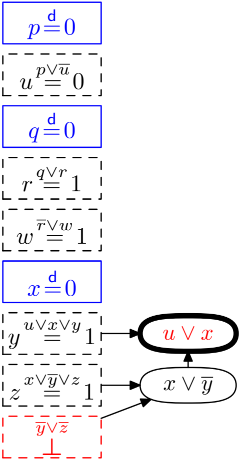


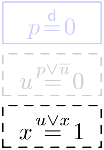
Assertion level 1 (max for non-UIP literal in learned clause) — keep trail to that level

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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





Assertion level 1 (max for non-UIP literal in learned clause) — keep trail to that level

Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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

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

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

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

$r \stackrel{q \lor r}{=} 1$

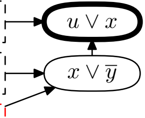
$w \stackrel{\overline{r} \lor w}{=} 1$

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

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

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

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



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

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

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

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

Assertion level 1 (max for non-UIP literal in learned clause) — keep trail to that level

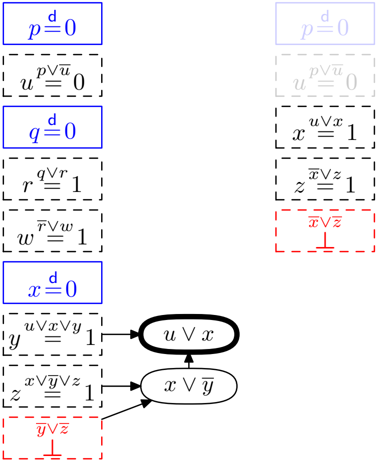
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Then continue as before...

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

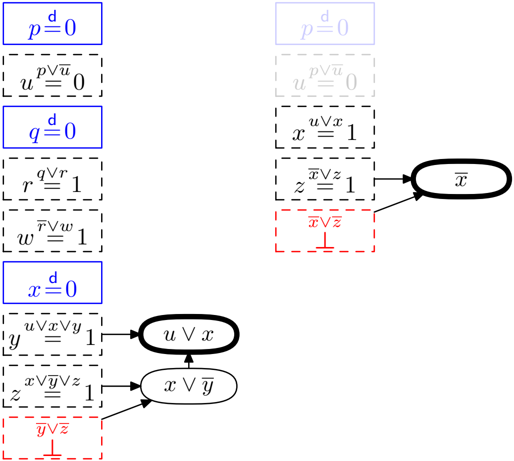
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

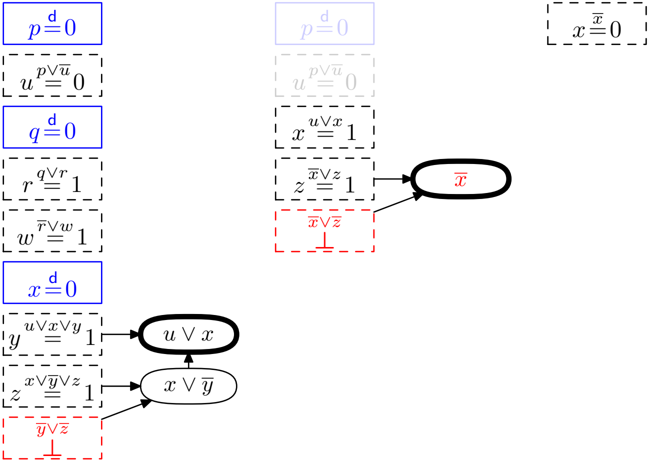
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

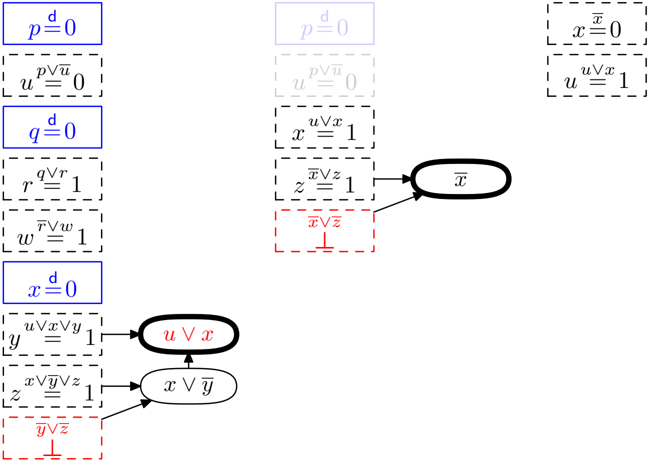
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

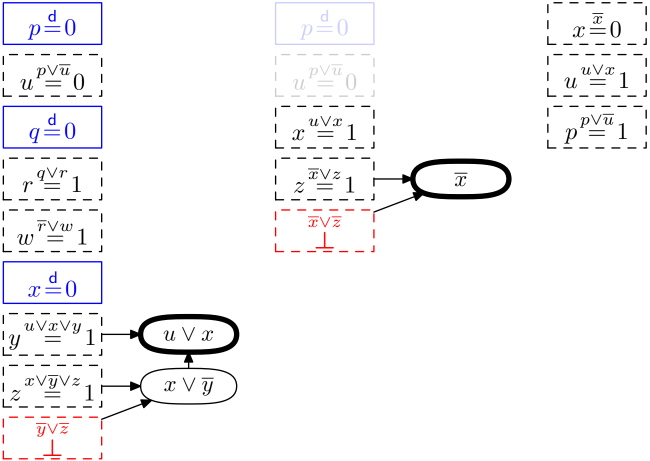
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

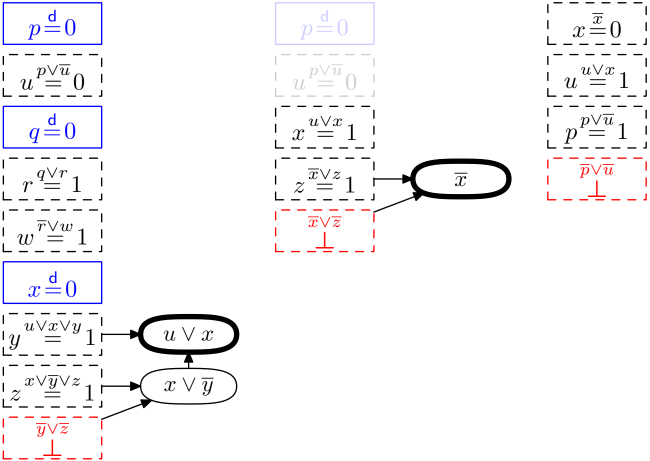
$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

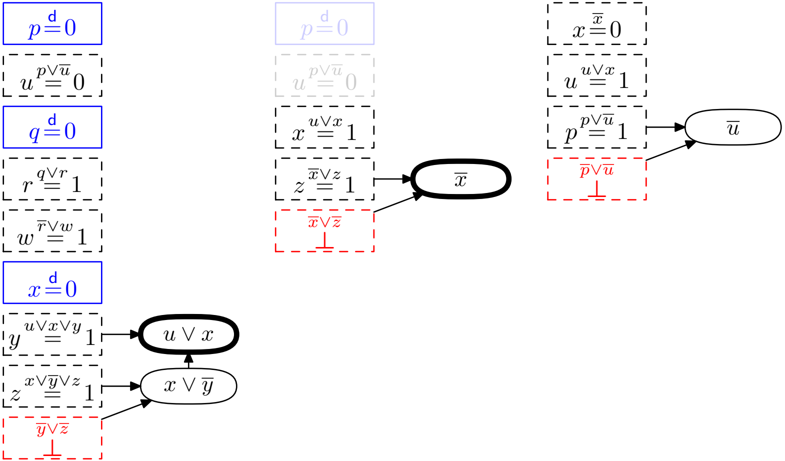
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

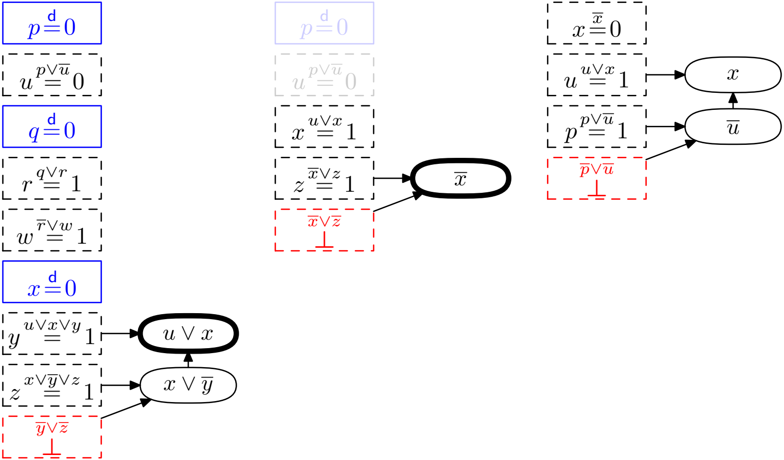
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

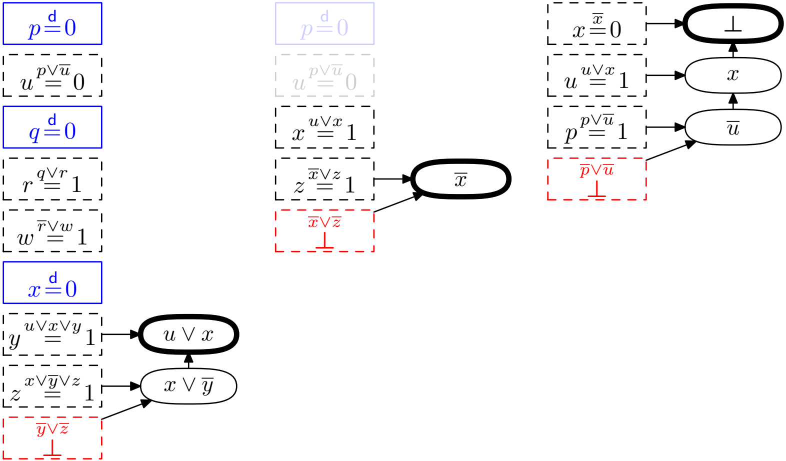
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



Clause Database Reduction

- In addition to learning clauses, also erase learned clauses that don't seem useful
- Modern solvers do this very aggressively
- Speeds up CDCL search (in particular, unit propagation, which dominates running time)
- But erasing too aggressively can throw away clauses that would have made solver terminate faster [EGG+18]
- So trade-off between search speed and search quality
- Except sometimes getting rid of clauses improves search quality too! [KN20]

Restarts

- Fairly frequently, start search all over (but keep learned clauses)
- Original intuition: stuck in bad part of search tree — go somewhere else
- Not the reason this is done now
- Popular variables with high VSIDS scores get set again [MMZ$^+$01]
- Are even set to same values (phase saving) [PD07]
- Current intution: improves the search by focusing on important variables
- Restart at fixed intervals or (better) make adaptive restarts depending on "quality" of learned clauses [AS09, AS12]

CDCL Main Loop Pseudocode

Algorithm 1: CDCL(F)

$\mathcal{D} \leftarrow F$; `// initialize clause database to contain formula`
$\rho \leftarrow \emptyset$; `// initialize assignment trail to empty`
forever do
  if *ρ falsifies some clause $C \in \mathcal{D}$* **then**
    $A \leftarrow$ ConflictAnalysis($\mathcal{D}, \rho, C$) ;
    if $A = \bot$ **then** output `UNSATISFIABLE` and exit;
    else
      add A to $\mathcal{D}$ and backjump by shrinking ρ ;
  else if *exists clause $C \in \mathcal{D}$ unit propagating x to $b \in \{0, 1\}$ under ρ* **then**
    add propagated assignment $x \stackrel{D}{=} b$ to ρ ;
  else if *time to restart* **then** $\rho \leftarrow \emptyset$;
  else if *time for clause database reduction* **then**
    erase (roughly) half of learned clauses in $\mathcal{D} \setminus F$ from $\mathcal{D}$
  else if *all variables assigned* **then** output `SATISFIABLE` and exit;
  else
    use decision scheme to choose assignment $x \stackrel{\mathsf{d}}{=} b$ to add to ρ ;

CDCL Main Loop Pseudocode

Algorithm 1: CDCL(F)

$\mathcal{D} \leftarrow F$; `// initialize clause database to contain formula`
$\rho \leftarrow \emptyset$; `// initialize assignment trail to empty`
forever do
  if *ρ falsifies some clause $C \in \mathcal{D}$* **then**
    $A \leftarrow$ ConflictAnalysis($\mathcal{D}, \rho, C$) ;
    if $A = \perp$ **then** output `UNSATISFIABLE` and exit;
    else
      add A to $\mathcal{D}$ and backjump by shrinking ρ ;
  else if *exists clause $C \in \mathcal{D}$ unit propagating x to $b \in \{0,1\}$ under ρ* **then**
    add propagated assignment $x \stackrel{D}{=} b$ to ρ ;
  else if *time to restart* **then** $\rho \leftarrow \emptyset$;
  else if *time for clause database reduction* **then**
    erase (roughly) half of learned clauses in $\mathcal{D} \setminus F$ from $\mathcal{D}$
  else if *all variables assigned* **then** output `SATISFIABLE` and exit;
  else
    use decision scheme to choose assignment $x \stackrel{\mathrm{d}}{=} b$ to add to ρ ;

Conflict Analysis Pseudocode

Algorithm 2: ConflictAnalysis$(\mathcal{D}, \rho, C_{\text{confl}})$

$C_{\text{learn}} \leftarrow C_{\text{confl}}$;
while C_{learn} *not UIP clause* **and** $C_{\text{learn}} \neq \bot$ **do**
    $\ell \leftarrow$ literal assigned last on trail ρ;
    if ℓ *propagated* **and** $\overline{\ell}$ *occurs in* C_{learn} **then**
        $C_{\text{reason}} \leftarrow$ reason$(\ell, \rho, \mathcal{D})$;
        $C_{\text{learn}} \leftarrow$ resolve$(C_{\text{learn}}, C_{\text{reason}})$;
    $\rho \leftarrow \rho \setminus \{\ell\}$;
return C_{learn};

State-of-the-art SAT solvers: What About the Recipe?

List of ingredients again (not exhaustive):

- Variable decisions & propagations
- Clause learning
- Restarts
- Clause database reduction

State-of-the-art SAT solvers: What About the Recipe?

List of ingredients again (not exhaustive):

- Variable decisions & propagations
- Clause learning
- Restarts
- Clause database reduction

Some natural questions:

- How best to combine these ingredients into a recipe?
- When and why does this recipe work?

State-of-the-art SAT solvers: What About the Recipe?

List of ingredients again (not exhaustive):

- Variable decisions & propagations
- Clause learning
- Restarts
- Clause database reduction

Some natural questions:

- How best to combine these ingredients into a recipe?
- When and why does this recipe work?

Why SAT solvers actually work so well
is a poorly understood question

Lots of research to comprehend this better
(Among other places in the MIAO group)

SAT Solver Analysis and the Resolution Proof System

How to make rigorous analysis of SAT solver performance?
Many intricate, hard-to-understand heuristics
So focus instead on underlying method of reasoning

SAT Solver Analysis and the Resolution Proof System

How to make rigorous analysis of SAT solver performance?

Many intricate, hard-to-understand heuristics

So focus instead on underlying method of reasoning

Resolution proof system

- Start with clauses of CNF formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C_1 \lor x \qquad C_2 \lor \overline{x}}{C_1 \lor C_2}$$

Resolution Proofs by Contradction

Resolution rule:

$$\frac{C_1 \lor x \qquad C_2 \lor \overline{x}}{C_1 \lor C_2}$$

Observation

If F is a satisfiable CNF formula and D is derived from clauses $D_1, D_2 \in F$ by the resolution rule, then $F \land D$ is satisfiable.

Resolution Proofs by Contradction

Resolution rule:

$$\frac{C_1 \lor x \qquad C_2 \lor \overline{x}}{C_1 \lor C_2}$$

Observation

If F is a satisfiable CNF formula and D is derived from clauses $D_1, D_2 \in F$ by the resolution rule, then $F \land D$ is satisfiable.

So can prove F unsatisfiable by deriving the unsatisfiable empty clause (denoted $\perp$) from F by resolution

Resolution Proofs by Contradction

Resolution rule:

$$\frac{C_1 \lor x \qquad C_2 \lor \overline{x}}{C_1 \lor C_2}$$

Observation

If F is a satisfiable CNF formula and D is derived from clauses $D_1, D_2 \in F$ by the resolution rule, then $F \land D$ is satisfiable.

So can prove F unsatisfiable by deriving the unsatisfiable empty clause (denoted $\perp$) from F by resolution

Such proof by contradiction also called resolution refutation

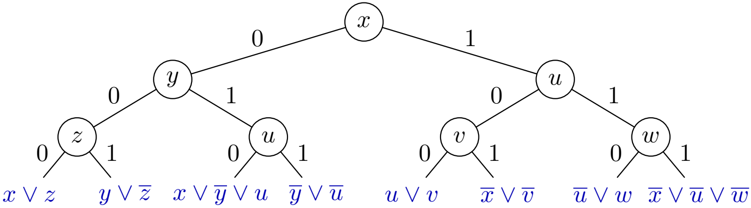
DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

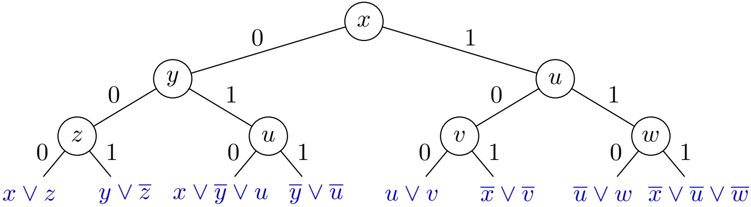
Look at our example again



DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

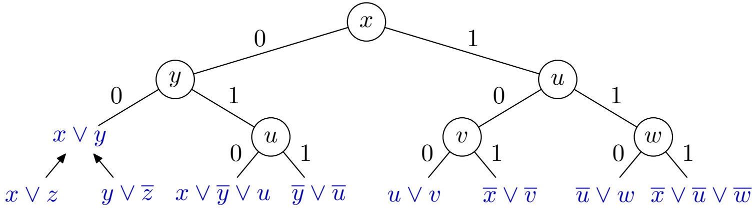


and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

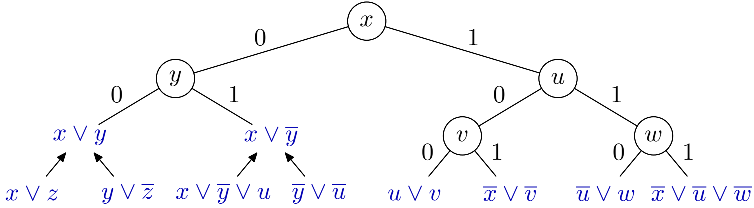


and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

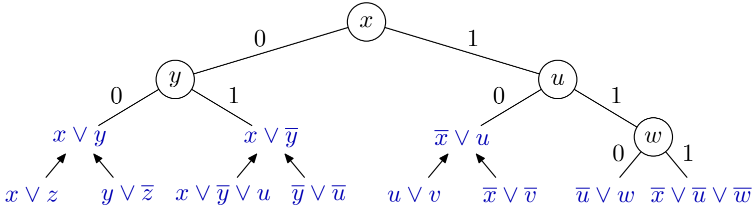


and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

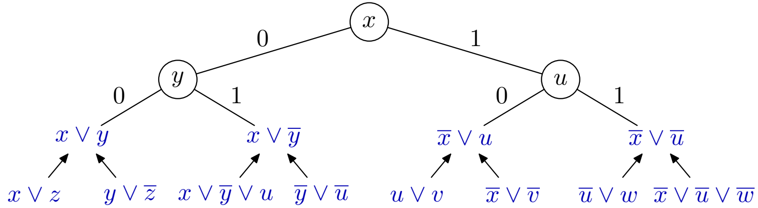


and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

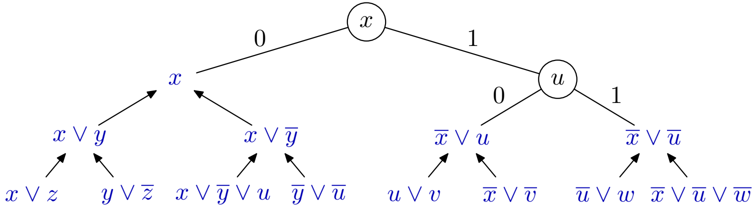


and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

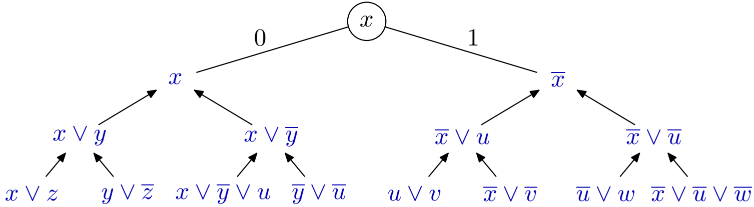


and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

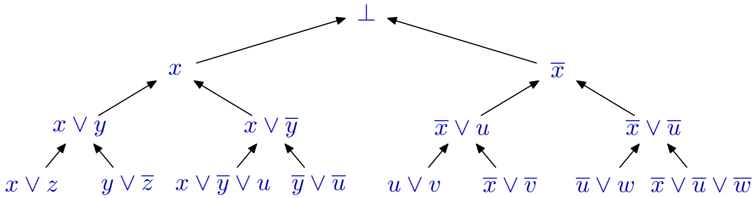


and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again



and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

DPLL Running Time and Tree-Like Resolution Proof Size

- Can extract resolution proof from any DPLL execution

DPLL Running Time and Tree-Like Resolution Proof Size

- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show

DPLL Running Time and Tree-Like Resolution Proof Size

- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show
- Such proof is tree-like — every derived clause used only once (to use a clause twice, we have to derive it twice from scratch)

DPLL Running Time and Tree-Like Resolution Proof Size

- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show
- Such proof is tree-like — every derived clause used only once (to use a clause twice, we have to derive it twice from scratch)
- Hence, lower bounds on tree-like proof size in resolution $\Rightarrow$ lower bounds on DPLL running time

DPLL Running Time and Tree-Like Resolution Proof Size

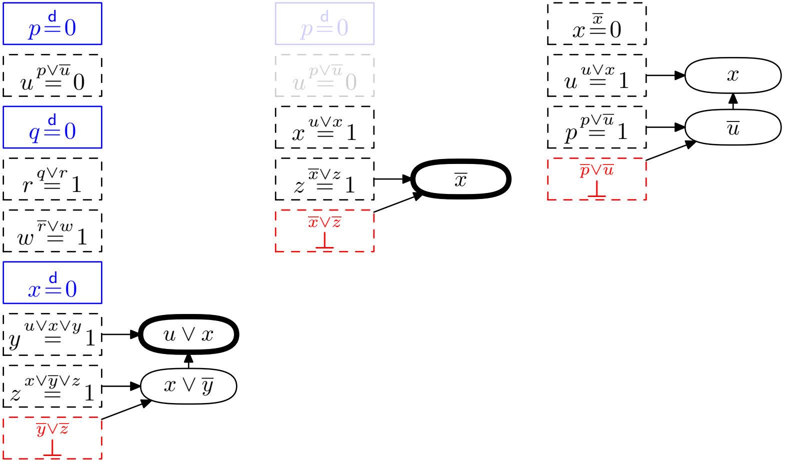
- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show
- Such proof is tree-like — every derived clause used only once (to use a clause twice, we have to derive it twice from scratch)
- Hence, lower bounds on tree-like proof size in resolution $\Rightarrow$ lower bounds on DPLL running time
- Conflict-driven clause learning adds "shortcut edges" in tree, but still yields resolution proof

CDCL and Resolution Proofs

Obtain resolution proof...

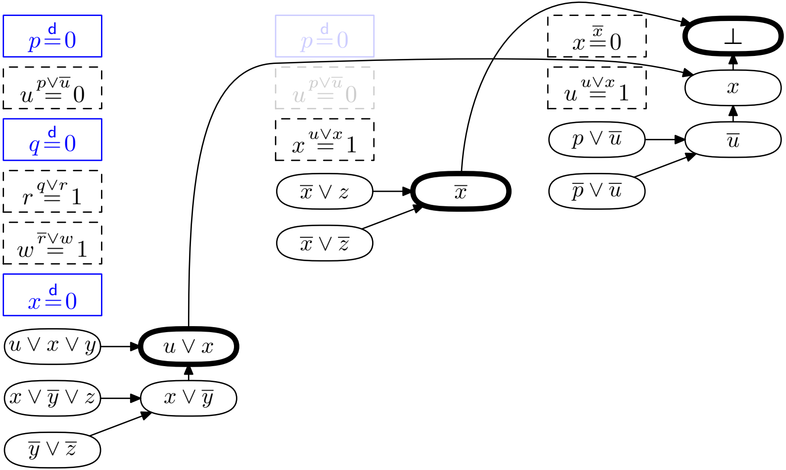
CDCL and Resolution Proofs

Obtain resolution proof from our example CDCL execution. . .



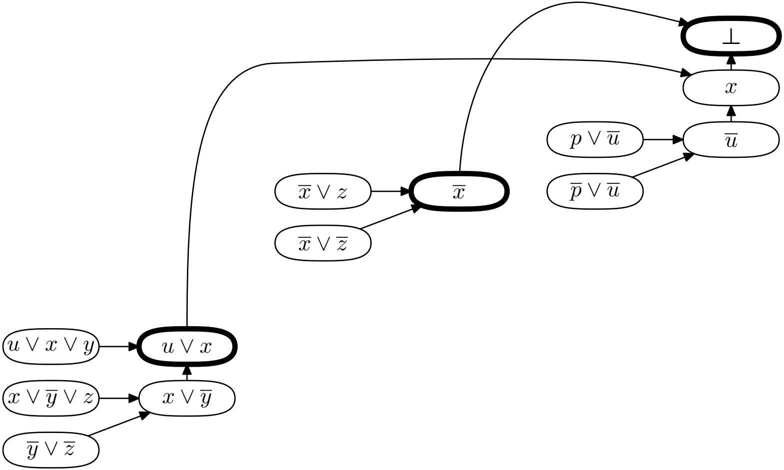
CDCL and Resolution Proofs

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



CDCL and Resolution Proofs

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



CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof (DAG-like, not tree-like) from CDCL execution

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, but you have seen enough in this lecture to be able to fill in the required details...

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, but you have seen enough in this lecture to be able to fill in the required details...
- This holds even for CDCL solvers with sophisticated heuristics and optimizations that we have not discussed*

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, but you have seen enough in this lecture to be able to fill in the required details...
- This holds even for CDCL solvers with sophisticated heuristics and optimizations that we have not discussed*
- Hence, lower bounds on resolution proof size $\Rightarrow$ lower bounds on CDCL running time

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, but you have seen enough in this lecture to be able to fill in the required details...
- This holds even for CDCL solvers with sophisticated heuristics and optimizations that we have not discussed*
- Hence, lower bounds on resolution proof size $\Rightarrow$ lower bounds on CDCL running time
- Lower (and upper) bounds for different methods of reasoning about propositional logic formulas studied in proof complexity

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, but you have seen enough in this lecture to be able to fill in the required details...
- This holds even for CDCL solvers with sophisticated heuristics and optimizations that we have not discussed*
- Hence, lower bounds on resolution proof size $\Rightarrow$ lower bounds on CDCL running time
- Lower (and upper) bounds for different methods of reasoning about propositional logic formulas studied in proof complexity

(*) Except for some preprocessing techniques, which is an important omission, but this gets complicated and we don't have time to go into details...

Current State of Affairs in SAT Solving

- State-of-the-art CDCL solvers often perform amazingly well ("SAT is easy in practice")

Current State of Affairs in SAT Solving

- State-of-the-art CDCL solvers often perform amazingly well ("SAT is easy in practice")
- Very poor theoretical understanding:
  - Why do heuristics work?
  - Why are applied instances easy?

Current State of Affairs in SAT Solving

- State-of-the-art CDCL solvers often 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 proof complexity lower bounds for (seemingly) "obvious" formulas

Examples of Hard Formulas For Resolution (1/3)

Pigeonhole principle (PHP) formulas [Hak85]
"$n + 1$ pigeons don't fit into n holes"

Examples of Hard Formulas For Resolution (1/3)

Pigeonhole principle (PHP) formulas [Hak85]
"$n+1$ pigeons don't fit into n holes"

Variables $p_{i,j}$ = "pigeon $i \to$ hole j"; $1 \le i \le n+1$; $1 \le j \le n$

$p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,n}$	every pigeon i gets a hole
$\overline{p}_{i,j} \vee \overline{p}_{i',j}$	no hole j gets two pigeons $i \neq i'$

Can also add "functionality" and "onto" axioms

$\overline{p}_{i,j} \vee \overline{p}_{i,j'}$	no pigeon i gets two holes $j \neq j'$
$p_{1,j} \vee p_{2,j} \vee \cdots \vee p_{n+1,j}$	every hole j gets a pigeon

Examples of Hard Formulas For Resolution (1/3)

Pigeonhole principle (PHP) formulas [Hak85]
"$n+1$ pigeons don't fit into n holes"

Variables $p_{i,j}$ = "pigeon $i \to$ hole j"; $1 \leq i \leq n+1$; $1 \leq j \leq n$

$p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,n}$	every pigeon i gets a hole
$\overline{p}_{i,j} \vee \overline{p}_{i',j}$	no hole j gets two pigeons $i \neq i'$

Can also add "functionality" and "onto" axioms

$\overline{p}_{i,j} \vee \overline{p}_{i,j'}$	no pigeon i gets two holes $j \neq j'$
$p_{1,j} \vee p_{2,j} \vee \cdots \vee p_{n+1,j}$	every hole j gets a pigeon

Even onto functional PHP hard — **"resolution cannot count"**

Resolution proof requires $\exp(\Omega(n)) = \exp\big(\Omega\big(\sqrt[3]{N}\big)\big)$ clauses
(measured in terms of formula size N)

Examples of Hard Formulas For Resolution (2/3)

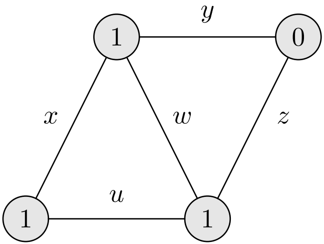
Tseitin formulas [Urq87]
"Sum of degrees of vertices in graph is even"

Examples of Hard Formulas For Resolution (2/3)

Tseitin formulas [Urq87]
"Sum of degrees of vertices in graph is even"

Variables = edges (in undirected graph of bounded degree)

- Label every vertex 0/1 so that sum of labels odd
- Write CNF requiring parity of # true incident edges = label



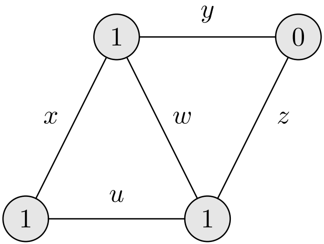
Examples of Hard Formulas For Resolution (2/3)

Tseitin formulas [Urq87]

"Sum of degrees of vertices in graph is even"

Variables = edges (in undirected graph of bounded degree)

- Label every vertex 0/1 so that sum of labels odd
- Write CNF requiring parity of # true incident edges = label



$$
\begin{array}{ll}
(u \lor x) & \land\ (y \lor \overline{z}) \\
\land\ (\overline{u} \lor \overline{x}) & \land\ (\overline{y} \lor z) \\
\land\ (w \lor x \lor y) & \land\ (u \lor w \lor z) \\
\land\ (w \lor \overline{x} \lor \overline{y}) & \land\ (u \lor \overline{w} \lor \overline{z}) \\
\land\ (\overline{w} \lor x \lor \overline{y}) & \land\ (\overline{u} \lor w \lor \overline{z}) \\
\land\ (\overline{w} \lor \overline{x} \lor y) & \land\ (\overline{u} \lor \overline{w} \lor z)
\end{array}
$$

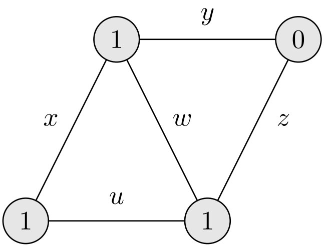
Examples of Hard Formulas For Resolution (2/3)

Tseitin formulas [Urq87]

"Sum of degrees of vertices in graph is even"

Variables = edges (in undirected graph of bounded degree)

- Label every vertex 0/1 so that sum of labels odd
- Write CNF requiring parity of # true incident edges = label



$$
\begin{array}{ll}
(u \lor x) & \land\ (y \lor \overline{z}) \\
\land\ (\overline{u} \lor \overline{x}) & \land\ (\overline{y} \lor z) \\
\land\ (w \lor x \lor y) & \land\ (u \lor w \lor z) \\
\land\ (w \lor \overline{x} \lor \overline{y}) & \land\ (u \lor \overline{w} \lor \overline{z}) \\
\land\ (\overline{w} \lor x \lor \overline{y}) & \land\ (\overline{u} \lor w \lor \overline{z}) \\
\land\ (\overline{w} \lor \overline{x} \lor y) & \land\ (\overline{u} \lor \overline{w} \lor z)
\end{array}
$$

Requires proof size $\exp(\Omega(N))$ on well-connected so-called expander graphs — **"resolution cannot count** $\bmod 2$**"**

Examples of Hard Formulas for Resolution (3/3)

Random k-CNF formulas [CS88]
Δn randomly sampled k-clauses over n variables

($\Delta \gtrsim 4.5$ sufficient to get unsatisfiable 3-CNF almost surely)

Again lower bound $\exp(\Omega(N))$

Examples of Hard Formulas for Resolution (3/3)

Random k-CNF formulas [CS88]
Δn randomly sampled k-clauses over n variables

($\Delta \gtrsim 4.5$ sufficient to get unsatisfiable 3-CNF almost surely)

Again lower bound $\exp(\Omega(N))$

And more...

- Colouring [BCMM05]
- Clique and VertexCover [BIS07]
- Zero-one designs [Spe10, VS10, MN14]
- Et cetera... (See, e.g., [BN21] for overview)

Theoretical Lower Bounds and Practical Reality

- If resolution so weak, how can CDCL SAT solvers be so good?
- One answer: this kind of "tricky" formulas don't show up too often in practice
- Another area of intense research: Try to describe what properties of "real-life" formulas make them easy or hard

Theoretical Lower Bounds and Practical Reality

- If resolution so weak, how can CDCL SAT solvers be so good?
- One answer: this kind of “tricky” formulas don’t show up too often in practice
- Another area of intense research: Try to describe what properties of “real-life” formulas make them easy or hard
- But sometimes we would like to be able to solve also “tricky” formulas (and, frankly, some seemed pretty easy, no?)
- Can we go beyond resolution?

Theoretical Lower Bounds and Practical Reality

- If resolution so weak, how can CDCL SAT solvers be so good?
- One answer: this kind of "tricky" formulas don't show up too often in practice
- Another area of intense research: Try to describe what properties of "real-life" formulas make them easy or hard
- But sometimes we would like to be able to solve also "tricky" formulas (and, frankly, some seemed pretty easy, no?)
- Can we go beyond resolution?
- Explore stronger methods of reasoning!
- Algorithms based on such methods could potentially lead to exponential speed-ups

Cutting Planes Proof System

Introduced in [CCT87] to model integer linear programming algorithm in [Gom63, Chv73]

Clauses translated to linear inequalities over the reals with integer coefficients

Cutting Planes Proof System

Introduced in [CCT87] to model integer linear programming algorithm in [Gom63, Chv73]

Clauses translated to linear inequalities over the reals with integer coefficients

Example: $x \vee y \vee \overline{z}$ gets translated to $x + y + (1 - z) \geq 1$ or equivalently $x + y - z \geq 0$

Cutting Planes Proof System

Introduced in [CCT87] to model integer linear programming algorithm in [Gom63, Chv73]

Clauses translated to linear inequalities over the reals with integer coefficients

Example: $x \vee y \vee \overline{z}$ gets translated to $x + y + (1 - z) \geq 1$ or equivalently $x + y - z \geq 0$

Derivation rules

Variable axioms $\dfrac{}{0 \leq x \leq 1}$

Multiplication $\dfrac{\sum a_i x_i \geq A}{\sum c a_i x_i \geq cA}$

Addition $\dfrac{\sum a_i x_i \geq A \quad \sum b_i x_i \geq B}{\sum (a_i + b_i) x_i \geq A + B}$

Division $\dfrac{\sum c a_i x_i \geq A}{\sum a_i x_i \geq \lceil A/c \rceil}$

Cutting Planes Refutation of CNF Formula

- Translate CNF formula to set of 0-1 linear inequalities
- Apply derivation rules
- Derive $0 \geq 1 \Leftrightarrow$ formula unsatisfiable
- Also makes sense for more general 0-1 linear inequalities (not just translations of CNF formulas)

Cutting Planes vs. Resolution

- Cutting planes can simulate resolution reasoning efficiently and is sometimes exponentially stronger
  (e.g., for PHP, just count to see #pigeons > #holes)

Cutting Planes vs. Resolution

- Cutting planes can simulate resolution reasoning efficiently and is sometimes exponentially stronger (e.g., for PHP, just count to see #pigeons $>$ #holes)
- And 0-1 linear inequalities much more concise than CNF

Compare

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3$$

and

$$\begin{aligned}
&(x_1 \lor x_2 \lor x_3 \lor x_4) \land (x_1 \lor x_2 \lor x_3 \lor x_5) \land (x_1 \lor x_2 \lor x_3 \lor x_6) \\
\land&(x_1 \lor x_2 \lor x_4 \lor x_5) \land (x_1 \lor x_2 \lor x_4 \lor x_6) \land (x_1 \lor x_2 \lor x_5 \lor x_6) \\
\land&(x_1 \lor x_3 \lor x_4 \lor x_5) \land (x_1 \lor x_3 \lor x_4 \lor x_6) \land (x_1 \lor x_3 \lor x_4 \lor x_6) \\
\land&(x_1 \lor x_4 \lor x_5 \lor x_6) \land (x_2 \lor x_3 \lor x_4 \lor x_5) \land (x_2 \lor x_3 \lor x_4 \lor x_6) \\
\land&(x_2 \lor x_3 \lor x_5 \lor x_6) \land (x_2 \lor x_4 \lor x_5 \lor x_6) \land (x_3 \lor x_4 \lor x_5 \lor x_6)
\end{aligned}$$

Cutting Planes vs. Resolution

- Cutting planes can simulate resolution reasoning efficiently and is sometimes exponentially stronger (e.g., for PHP, just count to see #pigeons > #holes)
- And 0-1 linear inequalities much more concise than CNF

Compare

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3$$

and

$$\begin{aligned}
&(x_1 \lor x_2 \lor x_3 \lor x_4) \land (x_1 \lor x_2 \lor x_3 \lor x_5) \land (x_1 \lor x_2 \lor x_3 \lor x_6) \\
\land&(x_1 \lor x_2 \lor x_4 \lor x_5) \land (x_1 \lor x_2 \lor x_4 \lor x_6) \land (x_1 \lor x_2 \lor x_5 \lor x_6) \\
\land&(x_1 \lor x_3 \lor x_4 \lor x_5) \land (x_1 \lor x_3 \lor x_4 \lor x_6) \land (x_1 \lor x_3 \lor x_4 \lor x_6) \\
\land&(x_1 \lor x_4 \lor x_5 \lor x_6) \land (x_2 \lor x_3 \lor x_4 \lor x_5) \land (x_2 \lor x_3 \lor x_4 \lor x_6) \\
\land&(x_2 \lor x_3 \lor x_5 \lor x_6) \land (x_2 \lor x_4 \lor x_5 \lor x_6) \land (x_3 \lor x_4 \lor x_5 \lor x_6)
\end{aligned}$$

- Construct more efficient SAT solvers using cutting planes?

SAT Solvers Based on Cutting Planes?

So-called pseudo-Boolean solvers using (subset of) cutting planes reasoning developed in, e.g., [CK05, LP10, EN18]

Counter-intuitively, hard to make competitive with CDCL

SAT Solvers Based on Cutting Planes?

So-called pseudo-Boolean solvers using (subset of) cutting planes reasoning developed in, e.g., [CK05, LP10, EN18]

Counter-intuitively, hard to make competitive with CDCL

Challenge 1: Conjunctive normal form

- Pseudo-Boolean solvers terrible for CNF input
- Solvers can rewrite CNF to more helpful 0-1 linear inequalities [BLLM14, EN20], but this doesn't work so well in practice
- Better to encode problem with 0-1 inequalities from the start

SAT Solvers Based on Cutting Planes?

So-called pseudo-Boolean solvers using (subset of) cutting planes reasoning developed in, e.g., [CK05, LP10, EN18]

Counter-intuitively, hard to make competitive with CDCL

Challenge 1: Conjunctive normal form

- Pseudo-Boolean solvers terrible for CNF input
- Solvers can rewrite CNF to more helpful 0-1 linear inequalities [BLLM14, EN20], but this doesn't work so well in practice
- Better to encode problem with 0-1 inequalities from the start

Challenge 2: Increased degrees of freedom(!?)

- Cutting planes much smarter method of reasoning
- But this makes it trickier to design smart search algorithms

SAT Solvers Based on Cutting Planes?

So-called pseudo-Boolean solvers using (subset of) cutting planes reasoning developed in, e.g., [CK05, LP10, EN18]

Counter-intuitively, hard to make competitive with CDCL

Challenge 1: Conjunctive normal form

- Pseudo-Boolean solvers terrible for CNF input
- Solvers can rewrite CNF to more helpful 0-1 linear inequalities [BLLM14, EN20], but this doesn't work so well in practice
- Better to encode problem with 0-1 inequalities from the start

Challenge 2: Increased degrees of freedom(!?)

- Cutting planes much smarter method of reasoning
- But this makes it trickier to design smart search algorithms

Is it truly harder to build good pseudo-Boolean solvers?
Or has just so much more work has been put into CDCL solvers?

So... Is There a Smarter Way Than Brute-Force?

In theory, probably no...

- COLOURING, CLIQUE, SAT, and 1000s other problems are "all the same" — efficient algorithm for one can solve all (the problems are all NP-complete)
- Widely believed impossible to construct algorithms that are always (a) efficient and (b) correct (even in worst case)
- Settling this question is one of Millennium Prize Problems: Are there efficient algorithms for NP-complete problems?

So. . . Is There a Smarter Way Than Brute-Force?

In theory, probably no. . .

- COLOURING, CLIQUE, SAT, and 1000s other problems are "all the same" — efficient algorithm for one can solve all (the problems are all NP-complete)
- Widely believed impossible to construct algorithms that are always (a) efficient and (b) correct (even in worst case)
- Settling this question is one of Millennium Prize Problems: Are there efficient algorithms for NP-complete problems?

In practice, definitely yes!

- Real-world problems are usually not "worst-case" but highly structured
- Fairly simple (but clever) methods work amazingly well amazingly often (though we don't really understand why)

So. . . Is There a Smarter Way Than Brute-Force?

In theory, probably no. . .

- COLOURING, CLIQUE, SAT, and 1000s other problems are "all the same" — efficient algorithm for one can solve all (the problems are all NP-complete)
- Widely believed impossible to construct algorithms that are always (a) efficient and (b) correct (even in worst case)
- Settling this question is one of Millennium Prize Problems: Are there efficient algorithms for NP-complete problems?

In practice, definitely yes!

- Real-world problems are usually not "worst-case" but highly structured
- Fairly simple (but clever) methods work amazingly well amazingly often (though we don't really understand why)

Stark disconnect between theory and practice. . .

Research Goals in the MIAO Group (1/2)

Strengthen the mathematical analysis of algorithmic methods

- Study methods of reasoning powerful enough to capture state-of-the-art algorithms used in practice
- Prove theorems about their power and limitations
- E.g., resolution proof system captures CDCL reasoning

Research Goals in the MIAO Group (1/2)

Strengthen the mathematical analysis of algorithmic methods

- Study methods of reasoning powerful enough to capture state-of-the-art algorithms used in practice
- Prove theorems about their power and limitations
- E.g., resolution proof system captures CDCL reasoning

Construct stronger algorithms for combinatorial problems

- Use insights into stronger mathematical methods of reasoning to build algorithms for SAT and other combinatorial problems
- Aiming for exponential speed-ups over state of the art
- E.g., use cutting planes to build pseudo-Boolean solvers

Research Goals in the MIAO Group (2/2)

Improve understanding of efficient computation in practice

- Use computational complexity theory to study "real-world" (not worst-case) problems
- Combine theoretical study and empirical experiments
- E.g., take "crafted formulas" with provable theoretical properties and investigate correlation with practical solver performance

Research Goals in the MIAO Group (2/2)

Improve understanding of efficient computation in practice

- Use computational complexity theory to study "real-world" (not worst-case) problems
- Combine theoretical study and empirical experiments
- E.g., take "crafted formulas" with provable theoretical properties and investigate correlation with practical solver performance

Certify correctness for modern combinatorial solvers

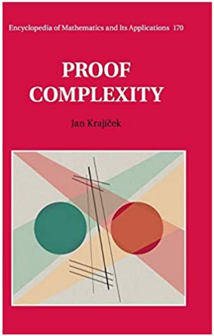
- In many combinatorial optimization paradigms, state-of-the-art solvers are known to be buggy
- Develop methods to make solvers output not just answer but machine-verifiable proof of correctness of this answer

Some References for Further Reading

Handbook of Satisfiability
(Especially chapter 7 ☺)

[BHvMW21]

Proof Complexity
by Jan Krajíček



[Kra19]

And survey papers, slides, and videos at `www.jakobnordstrom.se`

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice
- SAT solving more of an art form than a science — theoretical understanding lagging far behind

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice
- SAT solving more of an art form than a science — theoretical understanding lagging far behind
- Can use proof complexity to analyze potential and limitations of SAT solvers

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice
- SAT solving more of an art form than a science — theoretical understanding lagging far behind
- Can use proof complexity to analyze potential and limitations of SAT solvers
- And to get inspirations for algorithms based on stronger methods of reasoning

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice
- SAT solving more of an art form than a science — theoretical understanding lagging far behind
- Can use proof complexity to analyze potential and limitations of SAT solvers
- And to get inspirations for algorithms based on stronger methods of reasoning
- Lots of challenging work for PhD students (we're hiring!)

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice
- SAT solving more of an art form than a science — theoretical understanding lagging far behind
- Can use proof complexity to analyze potential and limitations of SAT solvers
- And to get inspirations for algorithms based on stronger methods of reasoning
- Lots of challenging work for PhD students (we're hiring!)
- Also room for interesting BSc or MSc projects

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice
- SAT solving more of an art form than a science — theoretical understanding lagging far behind
- Can use proof complexity to analyze potential and limitations of SAT solvers
- And to get inspirations for algorithms based on stronger methods of reasoning
- Lots of challenging work for PhD students (we're hiring!)
- Also room for interesting BSc or MSc projects

Thanks for listening! See you again Tuesday Jan 18!

References I

[AS09] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI '09)*, pages 399–404, July 2009.

[AS12] Gilles Audemard and Laurent Simon. Refining restarts strategies for SAT and UNSAT. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP '12)*, volume 7514 of *Lecture Notes in Computer Science*, pages 118–126. Springer, October 2012.

[BCMM05] Paul Beame, Joseph C. Culberson, David G. Mitchell, and Cristopher Moore. The resolution complexity of random graph k-colorability. *Discrete Applied Mathematics*, 153(1-3):25–47, December 2005.

[BHvMW21] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition, February 2021.

References II

[BIS07] Paul Beame, Russell Impagliazzo, and Ashish Sabharwal. The resolution complexity of independent sets and vertex covers in random graphs. *Computational Complexity*, 16(3):245–297, October 2007. Preliminary version in *CCC '01*.

[BLLM14] Armin Biere, Daniel Le Berre, Emmanuel Lonca, and Norbert Manthey. Detecting cardinality constraints in CNF. 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 285–301. Springer, July 2014.

[BN21] Samuel R. Buss and Jakob Nordström. Proof complexity and SAT solving. In Biere et al. [BHvMW21], chapter 7, pages 233–350.

[CCT87] William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.

[Chv73] Vašek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(1):305–337, 1973.

References III

[CK05] Donald Chai and Andreas Kuehlmann. A fast pseudo-Boolean constraint solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(3):305–317, March 2005. Preliminary version in *DAC '03*.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC '71)*, pages 151–158, May 1971.

[CS88] Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, October 1988.

[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.

References IV

[EGG+18] Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, Jakob Nordström, and Laurent Simon. Seeking practical CDCL insights from theoretical SAT benchmarks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1300–1308, July 2018.

[EN18] Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1291–1299, July 2018.

[EN20] Jan Elffers and Jakob Nordström. A cardinal improvement to pseudo-Boolean solving. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1495–1503, February 2020.

[Gom63] Ralph E. Gomory. An algorithm for integer solutions of linear programs. In R.L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.

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

References V

[KN20] Janne I. Kokkala and Jakob Nordström. Using resolution proofs to analyse CDCL solvers. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 427–444. Springer, September 2020.

[Kra19] Jan Krajíček. *Proof Complexity*, volume 170 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, March 2019.

[Lev73] Leonid A. Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973. In Russian. Available at `http://mi.mathnet.ru/ppi914`.

[LP10] Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, July 2010.

References VI

[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.

[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.

[MS99] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999. Preliminary version in *ICCAD '96*.

[PD07] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT '07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer, May 2007.

References VII

[Spe10] Ivor Spence. sgen1: A generator of small but difficult satisfiability benchmarks. *Journal of Experimental Algorithmics*, 15:1.2:1–1.2:15, March 2010.

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

[VS10] Allen Van Gelder and Ivor Spence. Zero-one designs produce small hard SAT instances. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 388–397. Springer, July 2010.