SAT Solving
Announcements

- **Office Hours Swap:**
  - Zavain has office hours from 4-6PM today in building 460, room 040A.
  - Rose has office hours tonight from 7-9PM in Gates B26B.
  - Keith has office hours Thursday from 2-4PM in Gates 178.

- OH schedule has been updated to reflect this.
A Note on the Checkpoints...
This Doesn't Work!

*Theorem:* If $R$ is transitive, then $R^{-1}$ is transitive.

*Proof:* Consider any $a$, $b$, and $c$ such that $aRb$ and $bRc$. Since $R$ is transitive, we have $aRc$. Since $aRb$ and $bRc$, we have $bR^{-1}a$ and $cR^{-1}b$. Since we have $aRc$, we have $cR^{-1}a$. Thus $cR^{-1}b$, $bR^{-1}a$, and $cR^{-1}a$. ■
This Doesn't Work!

**Theorem:** If $R$ is transitive, then $R^{-1}$ is transitive.

**Proof:** Consider any $a$, $b$, and $c$ such that $aRb$ and $bRc$. Since $R$ is transitive, we have $aRc$. Since $aRb$ and $bRc$, we have $bR^{-1}a$ and $cR^{-1}b$. Since we have $aRc$, we have $cR^{-1}a$. Thus $cR^{-1}b$, $bR^{-1}a$, and $cR^{-1}a$. ■

This proves

$$\forall a. \forall b. \forall c. (aRb \land bRc \rightarrow cR^{-1}b \land bR^{-1}a \land cR^{-1}a)$$
This Doesn't Work!

*Theorem:* If \( R \) is transitive, then \( R^{-1} \) is transitive.

*Proof:* Consider any \( a, b, \) and \( c \) such that \( aRb \) and \( bRc \). Since \( R \) is transitive, we have \( aRc \). Since \( aRb \) and \( bRc \), we have \( bR^{-1}a \) and \( cR^{-1}b \). Since we have \( aRc \), we have \( cR^{-1}a \). Thus \( cR^{-1}b, bR^{-1}a, \) and \( cR^{-1}a \). ■

This proves

\[
\forall a. \forall b. \forall c. (aRb \land bRc \rightarrow cR^{-1}b \land bR^{-1}a \land cR^{-1}a)
\]

You need to show

\[
\forall a. \forall b. \forall c. (aR^{-1}b \land bR^{-1}c \rightarrow aR^{-1}c)
\]
Theorem: If $R$ is transitive, then $R^{-1}$ is transitive.
Proof: Consider any $a$, $b$, and $c$ such that $aR^{-1}b$ and $bR^{-1}c$. We will prove $aR^{-1}c$. Since $aR^{-1}b$ and $bR^{-1}c$, we have that $bRa$ and $cRb$. Since $cRb$ and $bRa$, by transitivity we know $cRa$. Since $cRa$, we have $aR^{-1}c$, as required. ■
The Takeaway Point

• Don't get tripped up by definitions!
• If you want to directly prove that $P \rightarrow Q$, assume that $P$ is true and prove that $Q$ is true as well.
And now... SAT!
Is This Formula Ever True?

$p \lor \neg p$
Is This Formula Ever True?

$p \land \neg p$
Is This Formula Ever True?

\((r \rightarrow s \rightarrow t) \land (s \rightarrow t \rightarrow r) \land (t \rightarrow r \rightarrow s) \land t \land \neg s\)
Is This Formula Ever True?

\((x_0 \rightarrow (x_1 \leftrightarrow x_0)) \lor (x_2 \land x_1 \land \neg x_0) \lor (x_1 \rightarrow \neg x_1)\)
Satisfiability

- A propositional logic formula $\varphi$ is called **satisfiable** if there is some assignment to its variables that makes it evaluate to true.

- An assignment of true and false to the variables of $\varphi$ that makes it evaluate to true is called a **satisfying assignment**.

- Similar terms:
  - $\varphi$ is **tautological** if it is always true.
  - $\varphi$ is **satisfiable** if it *can* be made true.
  - $\varphi$ is **unsatisfiable** if it is always false.
The boolean satisfiability problem (SAT) is the following:

*Given a propositional logic formula $\varphi$, is $\varphi$ satisfiable?*

- Note: Goal is just to get a yes/no answer, not to actually find a satisfying assignment.
  - Though usually we can obtain an assignment easily when given a SAT solver.
- Extremely important problem both in theory and in practice.
Applications of SAT
Saturn

Precise and Scalable Software Analysis

Overview

The Saturn project is exploring techniques for highly scalable and precise analysis of software, with applications to both bug-finding and verification. Saturn is based on three main ideas:

- Saturn is *summary-based*: the analysis of a function $f$ is a summary of $f$'s behavior. At call sites for $f$, only $f$'s summary is used.

- Saturn is also *constraint-based*: analysis is expressed as a system of constraints describing how the state at one program point is related to the state at adjacent program points. The primary constraint language used in Saturn is boolean satisfiability, with each bit accessed by a procedure or loop represented by a distinct boolean variable.

- Program analyses in Saturn are expressed in a logic programming language with some extensions to support constraints and function summaries.
Saturn

Precise and Scalable Software Analysis

Overview

The Saturn project is exploring techniques for highly scalable and precise analysis of software, with applications to both bug-finding and verification. Saturn is based on three main ideas:

- Saturn is summary-based: the analysis of a function $f$ is a summary of $f$'s behavior. At call sites for $f$, only $f$'s summary is used.

- Saturn is also constraint-based: analysis is expressed as a system of constraints describing how the state at one program point is related to the state at adjacent program points. The primary constraint language used in Saturn is boolean satisfiability, with each bit accessed by a procedure or loop represented by a distinct boolean variable.

- Program analyses in Saturn are expressed in a logic programming language with some extensions to support constraints and function summaries.

http://saturn.stanford.edu/
Announcement: 2011 General Game Playing Competition

Introduction

General game players are systems able to accept declarative descriptions of arbitrary games at runtime and able to use such descriptions to play those games effectively. Unlike specialized game players, such as Deep Blue, general game players cannot rely on algorithms designed in advance for specific games. General game playing expertise must depend on intelligence on the part of the game player and not just intelligence of the programmer of the game player. In order to perform well, general game players must incorporate various Artificial Intelligence technologies, such as knowledge representation, reasoning, learning, and rational decision making; and these capabilities have to work together in integrated fashion.

While general game playing is a topic with inherent interest, work in this area has practical value as well. The underlying technology can be used in a variety of other application areas, such as business process management, electronic commerce, and military operations.
Solving SAT: Take One
A Simple Algorithm

• Given a formula $\phi$, we can just build a truth table for $\phi$ and check all of the rows.

• If any of them evaluate to true, then $\phi$ is satisfiable.

• If none of them evaluate to true, then $\phi$ is unsatisfiable.

• So what might this look like?
The Truth Table Algorithm

<table>
<thead>
<tr>
<th>$p$</th>
<th>$p \lor \neg p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
The Truth Table Algorithm

<table>
<thead>
<tr>
<th>$p$</th>
<th>$p \lor \neg p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
The Truth Table Algorithm

<table>
<thead>
<tr>
<th>$p$</th>
<th>$p \lor \neg p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>
The Truth Table Algorithm

<table>
<thead>
<tr>
<th>$p$</th>
<th>$p \lor \neg p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

where $\neg p$ is the negation of $p$. 

This table shows the truth values of $p \lor \neg p$ for all possible values of $p$. Since $\neg p$ is the negation of $p$, the disjunction $p \lor \neg p$ is always true, as it covers both cases where $p$ is true and where $p$ is false.
The Truth Table Algorithm

<table>
<thead>
<tr>
<th>$p$</th>
<th>$p \lor \neg p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>
The Truth Table Algorithm

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$(p \rightarrow q) \land q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
The Truth Table Algorithm

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$(p \rightarrow q) \land q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F F F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T T T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T T T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F F F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T T T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T T T</td>
</tr>
<tr>
<td>$p$</td>
<td>$q$</td>
<td>$r$</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>q</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
A Large Problem

• Truth tables can get very big very quickly!
• With $n$ variables, there will be $2^n$ rows.
• Many real-word SAT instances have hundreds of thousands of variables; this is completely infeasible!
Clause-Based Algorithms
Simplifying Our Formulas

- Arbitrary formulas in propositional logic can be complex.
  - Lots of different connectives.
  - Arbitrary nesting of formulas.
- Can be difficult to see how they all interrelate.
- Goal: Convert formulas into a simpler format.
Literals and Clauses

- A **literal** in propositional logic is a variable or its negation:
  - $x$
  - $\neg y$
  - But not $x \land y$.
- A **clause** is a many-way OR (disjunction) of literals.
  - $\neg x \lor y \lor \neg z$
  - $x$
  - But not $x \lor \neg(y \lor z)$
Conjunctive Normal Form

- A propositional logic formula $\phi$ is in **conjunctive normal form (CNF)** if it is the many-way AND (conjunction) of clauses.
  - $(x \lor y \lor z) \land (\neg x \lor \neg y) \land (x \lor y \lor z \lor \neg w)$
  - $x \lor z$
  - But not $(x \lor (y \land z)) \lor (x \lor y)$

- Only legal operators are $\neg$, $\lor$, $\land$.
- No nesting allowed.
The Structure of CNF

\[ (x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z) \]
The Structure of CNF

\((x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)\)
The Structure of CNF

\[(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)\]

Each clause must have \underline{at least one} true literal in it.
The Structure of CNF

\((x \lor y \lor z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)\)
We should pick at least one true literal from each clause
The Structure of CNF

\((x \lor y \lor \neg z) \land (\neg x \lor y \lor z) \land (\neg x \lor y \lor z)\)
The Structure of CNF

\((x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)\)
The Structure of CNF

\((x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)\)
The Structure of CNF

\[(x \lor y \lor z) \land (\neg x \lor y \lor z) \land (\neg x \lor y \lor z)\]
The Structure of CNF

\((x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)\)

... subject to the constraint that we never choose a literal and its negation
Getting to CNF

- There are excellent algorithms for solving SAT formulas in CNF.
- How do we convert an arbitrary propositional logic formula into a formula in CNF?
Negation Normal Form

- A formula \( \varphi \) in propositional logic is in **negation normal form (NNF)** iff
  - The only connectives are \( \neg \), \( \vee \), and \( \wedge \).
  - \( \neg \) is only applied directly to variables.
- **Examples:**
  - \((a \wedge (b \vee \neg c)) \vee (d \wedge \neg a)\)
  - \(a \vee b \vee \neg c\)
- **Non-Examples:**
  - \(\neg(a \wedge b)\)
  - \(a \rightarrow (\neg b \vee c)\)
  - \(\neg \neg a \vee \neg b \wedge \neg c\)
Getting to NNF

- NNF is a stepping stone toward CNF:
  - Only have $\lor$, $\land$, and $\neg$.
  - All negations pushed onto variables.
- Build an algorithm to get from arbitrary propositional logic down to NNF.
- Our conversion process will work as follows:
  - Eliminate complex connectives.
  - Simplify negations.
Eliminating Complex Connectives

- NNF only allows $\land$, $\lor$, $\neg$.
- First step: Replace other connectives with these three.
  - Replace $\phi \rightarrow \psi$ with $(\neg \phi \lor \psi)$.
  - Replace $\top$ with $(p \lor \neg p)$ for any variable $p$.
  - Replace $\bot$ with $(p \land \neg p)$ for any variable $p$.
  - For now, let's ignore $\leftrightarrow$; details are tricky.
- Result: A new formula that is logically equivalent to the original and not much bigger.
Eliminating Complex Connectives

\[ \neg(p \land q \land r) \rightarrow \neg(s \rightarrow \neg t \lor q) \]
\[ \neg\neg(p \land q \land r) \lor \neg(s \rightarrow \neg t \lor q) \]
\[ \neg\neg(p \land q \land r) \lor \neg(\neg s \lor \neg t \lor q) \]
Simplifying Negations

• NNF only allows negations in front of variables.

• Now that we have just $\lor$, $\land$, and $\neg$, repeatedly apply these rules to achieve this result:
  - Replace $\neg\neg\varphi$ with $\varphi$
  - Replace $\neg(\varphi \land \psi)$ with $\neg\varphi \lor \neg\psi$
  - Replace $\neg(\varphi \lor \psi)$ with $\neg\varphi \land \neg\psi$

• This process eventually terminates; the “height” of the negations keeps decreasing.
Simplifying Negations

\[ \neg \neg (p \land q \land r) \lor \neg (\neg s \lor \neg t \lor q) \]
\[ (p \land q \land r) \lor (\neg \neg s \lor \neg t \lor q) \]
\[ (p \land q \land r) \lor (s \land t \land \neg q) \]
\[ (p \land q \land r) \lor (s \land t \land \neg q) \]
From NNF to CNF

- Now that we can get to NNF, let's get down to CNF.
- Recall: CNF is the conjunction of clauses:
  \((x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_4) \land \ldots\)
- We'll use an inductive approach to convert NNF into CNF.
From NNF to CNF

• Every NNF formula is either
  • A literal,
  • The conjunction of two NNF formulas: \( \phi \land \psi \)
  • The disjunction of two NNF formulas: \( \phi \lor \psi \)
• Let's work through some examples and see if we can find a pattern.
Examples: NNF to CNF

\((x \lor y \lor z \lor \neg w) \land (y \lor x)\)
Examples: NNF to CNF

\[ x \lor (y \land z) \land (x \lor y) \land (x \lor z) \]
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>$x \lor (y \land z)$</th>
<th>$(x \lor y) \land (x \lor z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

The truth values for $x \lor (y \land z)$ and $(x \lor y) \land (x \lor z)$ are highlighted in yellow.
Examples: NNF to CNF

\[(w \land x) \lor (y \land z)\]
\[(((w \land x) \lor y) \land ((w \land x) \lor z)\]
\[(w \lor y) \land (x \lor y) \land (w \lor z) \land (x \lor z)\]
Examples: NNF to CNF

\[(v \land w \land x) \lor (y \land z)\]
\[(((v \land w \land x) \lor y) \land (((v \land w \land x) \land z)\]
\[(((v \land w) \lor y) \land (x \lor y) \land (((v \land w) \lor z) \land (x \lor z)\]
\[(v \lor y) \land (w \lor y) \land (x \lor y) \land (v \lor z) \land (w \lor z) \land (y \lor z)\]
Converting NNF to CNF

• Apply the following reasoning inductively:
  • If the formula is a literal, do nothing.
  • If the formula is $\varphi \land \psi$:
    - Convert $\varphi$ and $\psi$ to CNF, call it $\varphi'$ and $\psi'$.
    - Yield $\varphi' \land \psi'$
  • If the formula is $\varphi \lor \psi$:
    - Convert $\varphi$ and $\psi$ to CNF, call it $\varphi'$ and $\psi'$.
    - Repeatedly apply the distributive law
      $x \lor (y \land z) \equiv (x \lor y) \land (x \lor z)$ to $\varphi' \lor \psi'$ until simplified.
A Problem
A Problem

\[(a \land b) \lor (c \land d) \lor (e \land f) \lor (g \land h)\]
A Problem

\[(a \land b) \lor (c \land d) \lor (e \land f) \lor (g \land h)\]

\[((a \lor c) \land (a \lor d) \land (b \lor c) \land (b \lor d)) \lor (e \land f) \lor (g \land h)\]
A Problem

\[(a \land b) \lor (c \land d) \lor (e \land f) \lor (g \land h)\]

\[((a \lor c) \land (a \lor d) \land (b \lor c) \land (b \lor d)) \lor (e \land f) \lor (g \land h)\]

\[((a \lor c \lor e) \land (a \lor c \lor f) \land (a \lor d \lor e) \land (a \lor d \lor f) \land (b \lor c \lor e) \land (b \lor c \lor f) \land (b \lor d \lor e) \land (b \lor d \lor f)) \lor (g \land h)\]
A Problem

\[(a \land b) \lor (c \land d) \lor (e \land f) \lor (g \land h)\]

\[((a \lor c) \land (a \lor d) \land (b \lor c) \land (b \lor d))\]

\[\lor\]

\[(e \land f) \lor (g \land h)\]

\[((a \lor c \lor e) \land (a \lor c \lor f) \land (a \lor d \lor e) \land (a \lor d \lor f) \land\]

\[(b \lor c \lor e) \land (b \lor c \lor f) \land (b \lor d \lor e) \land (b \lor d \lor f))\]

\[\lor\]

\[(g \land h)\]

\[(a \lor c \lor e \lor v \land g) \land (a \lor c \lor f \lor v \land g) \land (a \lor d \lor e \lor v \land g) \land (a \lor d \lor f \lor v \land g) \land\]

\[(b \lor c \lor e \lor v \land g) \land (b \lor c \lor f \lor v \land g) \land (b \lor d \lor e \lor v \land g) \land (b \lor d \lor f \lor v \land g) \land\]

\[(a \lor c \lor e \lor v \land h) \land (a \lor c \lor f \lor v \land h) \land (a \lor d \lor e \lor v \land h) \land (a \lor d \lor f \lor v \land h) \land\]

\[(b \lor c \lor e \lor v \land h) \land (b \lor c \lor f \lor v \land h) \land (b \lor d \lor e \lor v \land h) \land (b \lor d \lor f \lor v \land h)\]
Exponential Blowup

- Our logic for eliminating $\vee$ can lead to exponential size increases.
- Not a problem with the algorithm; some formulas produce exponentially large CNF formulas.
- We will need to find another approach.
Eqivalence and Equisatisfiability

- Recall: Two logical formulas \( \phi \) and \( \psi \) are are **equivalent** (denoted \( \phi \equiv \psi \)) if they always take on the same truth values.

- Two logical formulas \( \phi \) and \( \psi \) are **equisatisfiable** (denoted \( \phi \cong \psi \)) if \( \phi \) is satisfiable iff \( \psi \) is satisfiable.

- To solve SAT for a formula \( \phi \), we can instead solve SAT for an equisatisfiable \( \psi \).
Equisatisfiably from NNF to CNF

\((a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\)
Equisatisfiably from NNF to CNF

\[(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\]
Equisatisfiably from NNF to CNF

\[(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\]

\[(a \land b \land \neg c)\]
Equisatisfiably from NNF to CNF

\[(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\]

\[(a) \land (b) \land (\neg c)\]
Equisatisfiably from NNF to CNF

$$(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)$$

$$(a \lor q) \land (b \lor q) \land (\neg c \lor q)$$
Equisatisfiably from NNF to CNF

\((a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\)

\((a \lor q) \land (b \lor q) \land (\neg c \lor q)\)

\((\neg a \land c)\)
Equisatisfiably from NNF to CNF

$$(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)$$

$$(a \lor q) \land (b \lor q) \land (\neg c \lor q)$$

$$(\neg a) \land (c)$$
Equisatisfiably from NNF to CNF

\[(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\]

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q)\]

\[(\neg a \lor \neg q) \land (c \lor \neg q)\]
Equisatisfiably from NNF to CNF

\[(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\]

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q)\]

\[\land \]

\[(\neg a \lor \neg q) \land (c \lor \neg q)\]
Equisatisfiably from NNF to CNF

\[(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\]

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q)\]

\[\land\]

\[(\neg a \lor \neg q) \land (c \lor \neg q)\]
Equisatisfiably from NNF to CNF

\((a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\)

\((a \lor q) \land (b \lor q) \land (\neg c \lor q)\)

\(\land\)

\((\neg a \lor \neg q) \land (c \lor \neg q)\)
Equisatisfiably from NNF to CNF

\((a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\)

\((a \lor q) \land (b \lor q) \land (\neg c \lor q) \land
\neg a \lor \neg q) \land (c \lor \neg q)\)
Equisatisfiably from NNF to CNF

\[(a \land b \land \neg c) \lor \neg a \land c \lor (a \land \neg b)\]

\[(a \lor q) \land (b \lor q) \land (c \lor q)\]

\[\land\]

\[(\neg a \lor q) \land (c \lor \neg q)\]
Equisatisfiably from NNF to CNF

\[(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\]

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q)\]
Equisatisfiability from NNF to CNF

\[(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\]

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q)\]

\[\land\]

\[(\neg a \lor \neg q) \land (c \lor \neg q)\]
Equisatisfiably from NNF to CNF

\[(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\]

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q)\]

\[\land\]

\[(\neg a \lor \neg q) \land (c \lor \neg q)\]
Equisatisfiably from NNF to CNF

\[(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\]

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q)\]

\[\land \]

\[(\neg a \lor \neg q) \land (c \lor \neg q)\]
Equisatisfiably from NNF to CNF

\[(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\]

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q)\]

\[\land\]

\[(\neg a \lor \neg q) \land (c \lor \neg q)\]
Equisatisfiably from NNF to CNF

\[(a \land b \land \neg c) \lor (\neg a \land c) \lor (a \land \neg b)\]

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q)\]
Equisatisfiably from NNF to CNF

\[( (a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) ) \lor (a \land \neg b) \]

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) \]
Equisatisfiably from NNF to CNF

$$( (a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) ) \lor (a \land \neg b)$$
Equisatisfiably from NNF to CNF

\[( (a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) ) \lor (a \land \neg b) \]
Equisatisfiably from NNF to CNF

\(( (a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) ) \lor (a \land \neg b) \)

\(( (a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) ) \)
Equisatisfiably from NNF to CNF

\[( (a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) ) \lor (a \land \neg b) \]
Equisatisfiably from NNF to CNF

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) \lor (a \land \neg b)\]

\[(a \lor q \lor r) \land (b \lor q \lor r) \land (\neg c \lor q \lor r) \land (\neg a \lor \neg q \lor r) \land (c \lor \neg q \lor r)\]
Equisatisfiably from NNF to CNF

\[
( (a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) ) \lor (a \land \neg b)
\]

\[
( (a \lor q \lor r) \land (b \lor q \lor r) \land (\neg c \lor q \lor r) \land (\neg a \lor \neg q \lor r) \land (c \lor \neg q \lor r))
\]

\[
a \land \neg b
\]
Equisatisfiably from NNF to CNF

\(( (a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) ) \lor (a \land \neg b) \)
Equisatisfiably from NNF to CNF

$$(a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) \lor (a \land \neg b)$$

$$(a \lor q \lor r) \land (b \lor q \lor r) \land (c \lor q \lor r) \land (\neg a \lor \neg q \lor r) \land (c \lor \neg q \lor r) \land (a \land \neg r) \land (\neg b \land \neg r)$$
Equisatisfiably from NNF to CNF

\[
\left( (a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) \right) \lor \left( a \land \neg b \right)
\]

\[
\left( (a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) \right) \land
\left( a \land \neg r \right) \land \left( \neg b \land \neg r \right)
\]
Equisatisfiability from NNF to CNF

\[ (a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) ) \lor (a \land \neg b) \]

\[ (a \lor q \lor r) \land (b \lor q \lor r) \land (\neg c \lor q \lor r) \land (\neg a \lor \neg q \lor r) \land (c \lor \neg q \lor r) \land (a \land \neg r) \land (b \land \neg r) \land (c \land \neg r) \]
Equisatisfiably from NNF to CNF

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) \lor (a \land \neg b)\]

\[(a \lor q \lor r) \land (b \lor q \lor r) \land (\neg c \lor q \lor r) \land (\neg a \lor \neg q \lor r) \land (c \lor \neg q \lor r) \land (a \land \neg r) \land (\neg b \land \neg r)\]
Equisatisfiably from NNF to CNF

\[
\begin{align*}
&(a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) \lor (a \land \neg b) \\
&(a \lor q \lor r) \land (b \lor q \lor r) \land (\neg c \lor q \lor r) \land (\neg a \lor \neg q \lor r) \land (c \lor \neg q \lor r) \\
&\land \\
&(a \land \neg r) \land (\neg b \land \neg r)
\end{align*}
\]
Equisatisfiably from NNF to CNF

\[( ( a \lor q ) \land ( b \lor q ) \land ( \neg c \lor q ) \land ( \neg a \lor \neg q ) \land ( c \lor \neg q ) ) \lor ( a \land \neg b ) \]

\[( ( a \lor q ) \lor r ) \land ( b \lor q ) \lor r ) \land ( \neg c \lor q ) \lor r ) \land ( \neg a \lor \neg q ) \lor r ) \land ( c \lor \neg q ) \lor r ) \land ( a \land \neg r ) \land ( b \land \neg r ) \land ( r \land \neg r ) \]
Equisatisfiably from NNF to CNF

\[(a \lor q) \land (b \lor q) \land (c \lor q) \land \neg a \lor \neg q \land c \lor \neg q) \lor (a \lor \neg b)\]

\[(a \lor q \lor r) \land (b \lor q \lor r) \land (c \lor q \lor r) \land \neg a \lor \neg q \lor r) \land (c \lor q \lor r) \land (a \lor r) \land (b \lor r) \land (c \lor r)\]
Equisatisfiably from NNF to CNF

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) \lor (a \land \neg b)\]

\[(a \lor q \lor r) \land (b \lor q \lor r) \land (\neg c \lor q \lor r) \land (\neg a \lor \neg q \lor r) \land (c \lor \neg q \lor r) \land (a \land \neg r) \land (\neg b \land r)\]
Equisatisfiably from NNF to CNF

\(( (a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) ) \lor (a \land \neg b) \)

\(( (a \lor q \lor r) \land (b \lor q \lor r) \land (\neg c \lor q \lor r) \land (\neg a \lor \neg q \lor r) \land (c \lor \neg q \lor r) \land (a \land \neg r) \land (b \land \neg r) ) \)
Equisatisfiably from NNF to CNF

\[(a \lor q) \land (b \lor q) \land (\neg c \lor q) \land (\neg a \lor \neg q) \land (c \lor \neg q) \lor (a \land \neg b)\]

\[(a \lor q \lor r) \land (b \lor q \lor r) \land (\neg c \lor q \lor r) \land (\neg a \lor \neg q \lor r) \land (c \lor \neg q \lor r) \land (a \land \neg r) \land (b \land \neg r)\]
Equisatisfiably from NNF to CNF

- If the formula is a literal, do nothing.
- If the formula is $\phi \land \psi$:
  - Convert $\phi$ and $\psi$ to CNF, call it $\phi'$ and $\psi'$.
  - Yield $\phi' \land \psi'$
- If the formula is $\phi \lor \psi$:
  - Convert $\phi$ and $\psi$ to CNF, call it $\phi'$ and $\psi'$.
  - Create a new variable $q$.
  - Add $q$ to each clause of $\phi'$.
  - Add $\neg q$ to each clause of $\psi'$.
  - Yield $\phi' \land \psi'$.
- Adds at most $n$ new variables to each clause, where $n$ is the number of clauses. Size increase at worst quadratic.
Where We Are Now

• We've found a way to convert arbitrary propositional logic formulas into equisatisfiable CNF formulas.

• The size may increase, but at most quadratically.

• Now, let's actually try to solve SAT!
A Simple Backtracking Algorithm
( ¬a ∨ b ∨ c )
( a ∨ c ∨ d )
( a ∨ c ∨ ¬d )
( a ∨ ¬c ∨ d )
( a ∨ ¬c ∨ ¬d )
( ¬b ∨ ¬c ∨ d )
( ¬a ∨ b ∨ ¬c )
( ¬a ∨ ¬b ∨ c )
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
(¬a ∨ b ∨ c)
(c ∨ d)
(c ∨ ¬d)
(¬c ∨ d)
(¬c ∨ ¬d)
(¬b ∨ ¬c ∨ d)
(¬a ∨ b ∨ ¬c)
(¬a ∨ ¬b ∨ c)
(\neg a \lor b \lor c )

( c \lor d )

(c \lor \neg d )

(\neg c \lor d )

(\neg c \lor \neg d )

(\neg b \lor \neg c \lor d )

(\neg a \lor b \lor \neg c )

(\neg a \lor \neg b \lor c )
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
( \neg a \lor b \lor c )

( c \lor d )

( c \lor \neg d )

( \neg c \lor d )

( c \lor \neg d )

( \neg b \lor \neg c \lor d )

( \neg a \lor b \lor \neg c )

( \neg a \lor \neg b \lor c )

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
(\neg a \lor b \lor c)

(d)

(\neg d)

(\neg c \lor d)

(\neg c \lor \neg d)

(\neg b \lor \neg c \lor d)

(\neg a \lor b \lor \neg c)

(\neg a \lor \neg b \lor c)

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
(¬a ∨ b ∨ c)

(  
  d  
)

(  
  ¬d  
)

(  
  ¬c ∨ d  
)

(  
  ¬c ∨ ¬d  
)

(  
  ¬b ∨ ¬c ∨ d  
)

(  
  ¬a ∨ b ∨ ¬c  
)

(  
  ¬a ∨ ¬b ∨ c  
)

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
( ¬a V b V c )
( ¬c V d )
( ¬c V ¬d )
( ¬b V ¬c V d )
( ¬a V b V ¬c )
( ¬a V ¬b V c )
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
\((\neg a \lor b \lor \neg c)\)

\((\neg d)\)

\((\neg c \lor \neg d)\)

\((\neg b \lor \neg c \lor d)\)

\((\neg a \lor \neg b \lor \neg c)\)

\((\neg a \lor \neg b \lor c)\)
(¬a ∨ b ∨ c)
(¬d)
(¬c ∨ d)
(¬c ∨ ¬d)
(¬b ∨ ¬c ∨ d)
(¬a ∨ ¬b ∨ ¬c)
(¬a ∨ b ∨ c)

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
( ¬a V b V c )
(       d   )
(         )
( ¬c V d )
( ¬c V ¬d )
( ¬b V ¬c V d )
( ¬a V b V ¬c )
( ¬a V ¬b V c )

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
(¬a V b V c)
(¬a V b V ¬c)
(¬a V b V ¬c V ¬d)
(¬a V ¬b V c)
(¬a V ¬b V c V d)
(¬b V ¬c V d)
(¬c V d)
(¬c V d)
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
\( (\neg a \lor b \lor c) \)
\( (c \lor d) \)
\( (c \lor \neg d) \)
\( (d) \)
\( (\neg d) \)
\( (\neg b \lor \neg c \lor d) \)
\( (\neg a \lor b \lor \neg c) \)
\( (\neg a \lor \neg b \lor c) \)

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
\((\neg a \lor b \lor c)\)
\((c \lor d)\)
\((c \lor \neg d)\)
\((d)\)
\((\neg d)\)
\((\neg b \lor \neg c \lor d)\)
\((\neg a \lor b \lor \neg c)\)
\((\neg a \lor \neg b \lor c)\)

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
\[-a \lor b \lor c \]

\[-c \lor b \lor -c \lor d \]

\[-a \lor -b \lor c \]

\[-a \lor b \lor -c \]

\[-c \lor c \lor -d \]

\[-d \]

\[-b \lor -c \lor d \]

\[-a \lor b \lor -c \]

\[-a \lor -b \lor c \]

\[-a \lor b \lor c \]
\((\neg a \lor b \lor c)\)

\((c \lor d)\)

\((c \lor \neg d)\)

\((d)\)

\((\neg d)\)

\((\neg b \lor \neg c \lor d)\)

\((\neg a \lor b \lor \neg c)\)

\((\neg a \lor \neg b \lor c)\)
(¬a ∨ b ∨ c)
(c ∨ d)
(c ∨ ¬d)
(d)
(¬d)
(¬b ∨ ¬c ∨ d)
(¬a ∨ b ∨ ¬c)
(¬a ∨ ¬b ∨ c)
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
(¬a ∨ b ∨ c)

(¬c ∨ d)

(¬c ∨ ¬d)

(¬c ∨ ¬d)

(¬c ∨ ¬d)

(¬a ∨ ¬b ∨ ¬c)

(¬a ∨ ¬b ∨ c)

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
(¬a V b V c)

(  c V d  )

(  c V ¬d  )

(¬c V d)

(¬c V ¬d)

(¬c V d)

(¬a V b V ¬c)

(¬a V ¬b V c)

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
(¬a ∨ b ∨ c)
( c ∨ d )
( c ∨ ¬d )
( ¬c ∨ d )
( ¬c ∨ ¬d )
( ¬b ∨ ¬c ∨ d )
( ¬a ∨ b ∨ ¬c )
( ¬a ∨ b ∨ c )

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
(¬a ∨ b ∨ c )
(a ∨ c ∨ d )
(a ∨ c ∨ ¬d )
(a ∨ ¬c ∨ d )
(a ∨ ¬c ∨ ¬d )
(¬b ∨ ¬c ∨ d )
(¬a ∨ b ∨ ¬c )
(¬a ∨ ¬b ∨ c )

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
( \neg a \lor b \lor c )
( a \lor c \lor d )
( a \lor c \lor \neg d )
( a \lor \neg c \lor d )
( a \lor \neg c \lor \neg d )
( \neg b \lor \neg c \lor d )
( \neg a \lor b \lor \neg c )
( \neg a \lor \neg b \lor c )
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
( ¬c )

( a ∨ c ∨ d )

( a ∨ c ∨ ¬d )

( a ∨ ¬c ∨ d )

( ¬b ∨ ¬c ∨ d )

( ¬b ∨ c )

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formulas from "The Quest for Efficient Boolean Satisfiability Solvers" by Sharad Malik of Princeton University
(a ∨ c)

(a ∨ c ∨ d)

(a ∨ c ∨ ¬d)

(a ∨ ¬c ∨ d)

(a ∨ ¬c ∨ ¬d)

(¬b ∨ ¬c ∨ d)

(¬b ∨ c)

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
( b ∨ c )
( a ∨ c ∨ d )
( a ∨ c ∨ ¬d )
( a ∨ ¬c ∨ d )
( a ∨ ¬c ∨ ¬d )
( ¬b ∨ ¬c ∨ d )
( b ∨ ¬c )
( ¬b ∨ c )

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
(b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬d)
(a ∨ ¬c ∨ d)
(a ∨ ¬c ∨ ¬d)
(¬c ∨ d)
(b ∨ ¬c)
(c)

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
( b ∨ c )
(a ∨ c ∨ d)
(a ∨ c ∨ ¬d)
(a ∨ ¬c ∨ d)
(a ∨ ¬c ∨ ¬d)
( ¬c ∨ d)
(b ∨ ¬c)
(c)

Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
( b \lor c )
(a \lor c \lor d)
(a \lor c \lor \neg d)
(a \lor \neg c \lor d)
(a \lor \neg c \lor \neg d)
(\neg c \lor d)
(b \lor \neg c)
(c)
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
(b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬d)
(a ∨ ¬c ∨ d)
(a ∨ ¬c ∨ ¬d)
(b ∨ ¬c)
(c)
(d)
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Formula from “The Quest for Efficient Boolean Satisfiability Solvers” by Sharad Malik of Princeton University
Backtracking SAT Solving

• If \( \varphi \) is an empty set of clauses, return \textbf{true}.
  • All clauses are satisfied.

• If \( \varphi \) contains \((\ )\), return \textbf{false},
  • Some clause is unsatisfiable.

• Otherwise:
  • Choose some variable \( x \).
  • Return whether \( \varphi_{\neg x} \) is satisfiable or \( \varphi_x \) is satisfiable,
    where \( \varphi_{\neg x} \) is \( \varphi \) with \( x \) set to false and \( \varphi_x \) is \( \varphi \) with \( x \) set to true.
Analyzing Backtracking

- The backtracking solver works reasonably well on most inputs.
  - Low memory usage – just need to remember one potential path along the tree.
  - For formulas with many satisfying assignments, typically finds one very quickly.

- But it has its weaknesses.
  - Completely blind searching – might miss “obvious” choices.
  - In the worst-case, must explore the entire tree, which has $2^n$ leaves.
Adding Heuristics

- A **heuristic** is an approach to solving a problem that may or may not work correctly.
  - Contrast with an **algorithm**, which has definitive guarantees on its behavior.
- The simplicity of CNF makes it possible to add heuristics to our backtracking solver.
- What sorts of heuristics might we add?
Pure Literal Elimination

\[
( \neg a \lor b \lor c ) \land ( a \lor c ) \land ( \neg a \lor \neg b ) \land ( \neg a \lor b \lor d ) \land ( \neg b \lor \neg d )
\]
Pure Literal Elimination

\[ (\neg a \lor b \lor c) \land (a \lor c) \land (\neg a \lor \neg b) \land (\neg a \lor b \lor d) \land (\neg b \lor \neg d) \]
Pure Literal Elimination

\( (\neg a \lor b \lor c) \land (a \lor c) \land (\neg a \lor \neg b) \land (\neg a \lor b \lor d) \land (\neg b \lor \neg d) \)

The variable c is never negated here. There is no reason not to set it to true.
Pure Literal Elimination

\[ (\neg a \lor \neg b) \land (\neg a \lor b \lor d) \lor (\neg b \lor \neg d) \]

The variable c is never negated here. There is no reason not to set it to true.
Pure Literal Elimination

(¬a ∨ ¬b) ∧ (¬a ∨ b ∨ d) ∧ (¬c ∨ ¬d)
Pure Literal Elimination

\[(\neg a \lor \neg b) \land (\neg a \lor b \lor d) \land (\neg b \lor \neg d)\]
Pure Literal Elimination

The variable \( a \) is always negated here. There is no reason not to set it to false.
Pure Literal Elimination

The variable a is always negated here. There is no reason not to set it to false.

( ¬a v ¬b )
Pure Literal Elimination

\((\neg b \lor \neg d)\)
Pure Literal Elimination

( \neg b \lor \neg d )
Pure Literal Elimination

The variable $b$ is always negated here. There is no reason not to set it to false.

$$(\neg b \lor \neg d)$$
Pure Literal Elimination

The variable b is always negated here. There is no reason not to set it to false.
Pure Literal Elimination
Pure Literal Elimination

All clauses have been satisfied, so the formula is satisfiable.
Pure Literal Elimination

• A literal is called **pure** if its negation appears nowhere in the formula.

• Setting that literal to true will satisfy some number of clauses automatically and simplify the formula.

• Many formulas can be satisfied by iteratively applying pure literal elimination.
Unit Propagation

$$
(\neg a \vee b \vee c) \land (a \lor d) \land (\neg a \lor \neg b) \land (\neg a \lor b \lor d) \land \neg d
$$
Unit Propagation

\((\neg a \lor b \lor c) \land (a \lor d) \land (\neg a \lor \neg b) \land (a \lor c) \land a \lor b \lor d) \land d\)
Unit Propagation

\[(\neg a \lor b \lor c) \land (a \lor d) \land (b \lor \neg a \lor \neg b \lor d) \land \neg d \land (a \lor b \lor d)\]

\(\neg d\) is all by itself. \(d\) has to be false for this formula to be true.
Unit Propagation

\((\neg a \lor b \lor c) \land (a) \land (\neg c \lor \neg a \lor b) \land (a \lor b)\)

\(\neg d\) is all by itself. \(d\) has to be false for this formula to be true.
Unit Propagation

\[(\neg a \lor b \lor c) \land (a) \land (\neg a \lor \neg b) \land (\neg a \lor b)\]
Unit Propagation

\[(\neg a \lor b \lor c) \land (a) \land (\neg a \lor \neg b) \land (\neg a \lor b) \land \neg a \lor b\]
Unit Propagation

\[(\neg a \lor b \lor c) \land (a \land (\neg a \lor \neg b) \land (\neg a \lor b) \land a \lor b)\]

a is all by itself. a has to be true for this formula to be true.
Unit Propagation

\((b \lor c) \land (\neg b) \land (b)\)

a is all by itself. a has to be true for this formula to be true.
Unit Propagation

\((b \lor c) \land (\neg b) \land (b)\)
Unit Propagation

\[(b \lor c) \land (\neg b) \land (b)\]
Unit Propagation

\[(b \lor c) \land (\neg b) \land (b)\]

b is all by itself. b has to be true for this formula to be true.
Unit Propagation

(b)

b is all by itself. b has to be true for this formula to be true.
Unit Propagation
Unit Propagation

( )
Unit Propagation

We are left with an empty clause. The formula is unsatisfiable.
Unit Propagation

• A **unit clause** is a clause containing just one literal.

• For the formula to be true, that literal must be set to true.

• This might expose other unit clauses.
DPLL

- The **DPLL algorithm** is a modification of the simple backtracking search algorithm.
  - Named for **Davis, Putnam, Logemann, and Loveland**, its inventors.
- Incorporates the two heuristics we just saw.
DPLL

- Simplify $\varphi$ with unit propagation.
- Simplify $\varphi$ with pure literal elimination.
- If $\varphi$ is empty, return true.
- If $\varphi$ contains ( ), return false.
- Otherwise:
  - Choose some variable $x$.
  - Return whether $\varphi_{\neg x}$ is satisfiable or $\varphi_{x}$ is satisfiable, where $\varphi_{\neg x}$ is $\varphi$ with $x$ set to false and $\varphi_{x}$ is $\varphi$ with $x$ set to true.
\[(a \lor b \lor c) \land \neg a \lor \neg b \lor c\]
\[(\neg a \lor \neg b \lor c) \land (\neg b \lor \neg c \lor \neg d)\]
\[(\neg b \lor \neg c \lor \neg d) \land (b \lor c \lor \neg d)\]
\[(\neg a \lor \neg b \lor \neg c) \land (\neg a \lor \neg c \lor \neg d)\]

START
(a ∨ b ∨ c)
(¬a ∨ ¬b ∨ c)
(¬b ∨ ¬c ∨ ¬d)
(¬b ∨ ¬c ∨ d)

( (a ∨ ¬d) )
( (b ∨ c ∨ d) )
( (¬a ∨ b ∨ ¬c) )
( (b ∨ c ∨ ¬d) )
( (¬a ∨ ¬b) )

( (a ∨ c ∨ d) )
( (¬a ∨ ¬b ∨ ¬d) )
( (a ∨ ¬b ∨ ¬c) )
( (a ∨ ¬c ∨ d) )

START

Guess ¬a
START

Guess ¬a
( a ∨ b ∨ c )
(b ∨ c ∨ d )
( ¬b ∨ ¬c ∨ ¬d )
( ¬b ∨ ¬c ∨ d )

( a ∨ ¬d )
(b ∨ c ∨ d )
( a ∨ ¬b ∨ ¬c )
( a ∨ ¬c ∨ d )

START

Guess ¬a
( b ∨ c )
( ¬d )
( b ∨ c ∨ d )
( c ∨ d )

( ¬b ∨ ¬c ∨ ¬d )
( b ∨ c ∨ ¬d )
( ¬b ∨ ¬c )
( ¬c ∨ d )

START

Guess ¬a
(b ∨ c)

(¬b ∨ ¬c ∨ d)

(b ∨ c ∨ ¬d)

(¬b ∨ ¬c ∨ ¬d)

(¬c ∨ d)

START

Guess ¬a
(b ∨ c)

(¬b ∨ ¬c ∨ ¬d)

(¬b ∨ ¬c ∨ d)

(¬d)

(c ∨ d)

(¬b ∨ ¬c)

(¬c ∨ d)

START

Guess ¬a

Propagate ¬d
( b ∨ c )
( b ∨ c ∨ d )
( ¬b ∨ ¬c ∨ ¬d )
( ¬b ∨ ¬c ∨ d )
( ¬b ∨ ¬c )
( ¬c ∨ d )

START

Guess ¬a
Propagate ¬d
(b ∨ c)

(b ∨ c ∨ d)

(c ∨ d)

(¬b ∨ ¬c)

(¬c ∨ d)

START

Guess ¬a

Propagate ¬d
( b ∨ c )

( b ∨ c )

( c )

( ¬b ∨ ¬c )

( ¬c )

START

Guess ¬a

Propagate ¬d
\[(b \lor c)\]  \[(b \lor c)\]  \[(c)\]  \[\neg b \lor \neg c\]  \[\neg c\]

START

Guess \(\neg a\)

Propagate \(\neg d\)
( b \lor c )

( b \lor c )

( \neg b \lor \neg c )

( \neg c )

START

Guess \neg a

Propagate \neg d

Propagate c
( b ∨ c )

( ¬b ∨ ¬c )

( ¬c )

( ¬b ∨ ¬c )

START

Guess ¬a

Propagate ¬d

Propagate c
\[(\neg b \lor \neg c) \land (\neg d)\]

START

- Guess \(\neg a\)
- Propagate \(\neg d\)
- Propagate \(c\)
START

- Guess \( \neg a \)
- Propagate \( \neg d \)
- Propagate \( c \)
\((\neg b)\)
( ¬b ) → Propagate ¬d → Propagate c → FAILURE

START → Guess ¬a → ( ¬b ) → ( )
\( (b \lor c) \quad (\neg b \lor \neg c) \quad (c) \quad (\neg b \lor \neg c) \quad (\neg c) \)

- **START**
  - Guess \( \neg a \)
  - Propagate \( \neg d \)
  - Propagate \( c \)

**FAILURE**
Guess \( \neg a \)

Propagate \( \neg d \)

Propagate \( c \)

FAILURE

\[
\begin{align*}
(b \lor c) & \quad (\neg d) \\
(b \lor c \lor \neg d) & \quad (c \lor d) \\
(\neg b \lor \neg c \lor \neg d) & \\
(\neg b \lor \neg c \lor \neg d & \lor \neg d) \\
\end{align*}
\]
( a ∨ b ∨ c )
( ¬a ∨ ¬b ∨ c )
( ¬b ∨ ¬c ∨ ¬d )
( ¬b ∨ ¬c ∨ d )

( a ∨ ¬d )
( b ∨ c ∨ d )
( ¬a ∨ b ∨ ¬c )
( b ∨ c ∨ ¬d )
( ¬a ∨ ¬b )

START

Guess ¬a
Propagate ¬d
Propagate c
FAILURE

( a ∨ c ∨ d )
( ¬a ∨ ¬b ∨ ¬d )
( a ∨ ¬b ∨ ¬c )
( a ∨ ¬c ∨ d )
Guess ¬a

Propagate ¬d

Propagate c

FAILURE

Forced: a
Guess ¬a

Propagate ¬d

Propagate c

FAILURE

Forced: a
( ¬a ∨ ¬b ∨ c )  
( ¬b ∨ ¬c ∨ ¬d )  
( ¬b ∨ ¬c ∨ d )  
( ¬a ∨ ¬b )

( b ∨ c ∨ d )  
( ¬a ∨ b ∨ ¬c )  
( b ∨ c ∨ ¬d )  
( ¬a ∨ ¬b ∨ ¬d )

START

Guess ¬a
Propagate ¬d
Propagate c
FAILURE

Forced: a
\[(\neg b \lor c)\]
\[(\neg b \lor \neg c \lor \neg d)\]
\[(\neg b \lor \neg c \land \neg d)\]
\[(b \lor c \lor \neg d)\]
\[(b \lor c \land \neg d)\]
\[(\neg b)\]

START

Guess \(\neg a\)
Propagate \(\neg d\)
Propagate \(c\)

Forced: \(a\)

FAILURE
Failure: 

\((\neg b \lor c)\) 
\((\neg b \lor \neg c \lor \neg d)\) 
\((\neg b \lor \neg c \lor d)\) 
\((b \lor c \lor \neg d)\) 
\((b \lor c)\) 
\((\neg b \lor \neg d)\) 

Forced: a
(¬b ∨ c)
(¬b ∨ ¬c ∨ ¬d)
(¬b ∨ ¬c ∨ d)

( b ∨ c ∨ d )
( b ∨ ¬c )
(b ∨ c ∨ ¬d)

¬b

START

Guess ¬a
Propagate ¬d
Propagate c
FAILURE

Forced: a
Propagate ¬b
(¬b ∨ c)
(¬b ∨ ¬c ∨ ¬d)
(¬b ∨ ¬c ∨ d)
(b ∨ c ∨ ¬d)
(b ∨ ¬c)
(¬b ∨ ¬d)

Forced: a
Guess ¬a
Propagate ¬d
Propagate c
FAILURE

Propagate ¬b
\(( b \lor c \lor d )\)

\(( b \lor \neg c )\)

\(( b \lor c \lor \neg d )\)

START

- Guess \( \neg a \)
  - Propagate \( \neg d \)
  - Propagate \( c \)
    - FAILURE
- Forced: \( a \)
  - Propagate \( \neg b \)
( c ∨ d )
( ¬c )
( c ∨ ¬d )

START

Guess ¬a
Propagate ¬d
Propagate c
FAILURE

Forced: a
Propagate ¬b
\[(c \lor d) \land (\neg c) \land (c \lor \neg d)\]

START

Guess \(\neg a\)
Propagate \(\neg d\)
Propagate \(c\)
FAILURE

Forced: \(a\)
Propagate \(\neg b\)
\[(c \lor d) \land (\neg c) \land (c \lor \neg d)\]

START

- Guess \(\neg a\)
- Propagate \(\neg d\)
- Propagate \(c\)

- Forced: \(a\)
- Propagate \(\neg b\)
- Propagate \(\neg c\)

FAILURE
\[(c \lor d)\]  
\[(\neg c)\]  
\[(c \lor \neg d)\]  

START  

Guess \(\neg a\)  
Propagate \(\neg d\)  
Propagate \(c\)  
FAILURE  

Forced: \(a\)  
Propagate \(\neg b\)  
Propagate \(\neg c\)
( (c ∨ d) )

( (c ∨ ¬d) )

START

- Guess ¬a
  - Propagate ¬d
  - Propagate c
  - FAILURE
- Forced: a
  - Propagate ¬b
  - Propagate ¬c
(d)
(¬d)

START

Guess ¬a
Propagate ¬d
Propagate c
FAILURE

Forced: a
Propagate ¬b
Propagate ¬c
(d)

Forced: a

(-b)

(¬d)
Guess ¬a
Propagate ¬d
Propagate c
FAILURE

Forced: a
Propagate ¬b
Propagate ¬c
Propagate d
START

- Guess ¬a
- Propagate ¬d
- Propagate c
- Propagate ¬c
- Propagate d

- Forced: a
- Propagate ¬b
- Propagate d

FAILURE

(... ¬d ...)

¬d
( )

START

Guess \( \neg a \)
Propagate \( \neg d \)
Propagate c
FAILURE

Forced: a
Propagate \( \neg b \)
Propagate \( \neg c \)
Propagate d
DPLL is Powerful

- DPLL was invented 50 years ago this July, but is still the basis for most SAT solvers.
- The two heuristics aggressively simplify many common cases.
- However, still has an exponential worst-case runtime.
A Randomized Approach
Randomized SAT Solving

(¬a ∨ b ∨ c )
( a ∨ c ∨ d )
( a ∨ c ∨ ¬d )
( a ∨ ¬c ∨ d )
( a ∨ ¬c ∨ ¬d )
( ¬b ∨ ¬c ∨ d )
( ¬a ∨ b ∨ ¬c )
( ¬a ∨ ¬b ∨ c )
Randomized SAT Solving

\[\neg a \lor b \lor c\]
\[a \lor c \lor d\]
\[a \lor c \lor \neg d\]
\[a \lor \neg c \lor d\]
\[\neg b \lor \neg c \lor d\]
\[\neg a \lor b \lor \neg c\]
\[\neg a \lor \neg b \lor c\]
Randomized SAT Solving

( ¬a ∨ b ∨ c )
( a ∨ c ∨ d )
( a ∨ c ∨ ¬d )
( a ∨ ¬c ∨ d )
( a ∨ ¬c ∨ ¬d )
( ¬b ∨ ¬c ∨ d )
( ¬a ∨ b ∨ ¬c )
( ¬a ∨ ¬b ∨ c )

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>false</td>
</tr>
<tr>
<td>b</td>
<td>false</td>
</tr>
<tr>
<td>c</td>
<td>false</td>
</tr>
<tr>
<td>d</td>
<td>false</td>
</tr>
</tbody>
</table>
Randomized SAT Solving

(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬d)
(a ∨ ¬c ∨ d)
(a ∨ ¬c ∨ ¬d)
(¬b ∨ ¬c ∨ d)
(¬a ∨ b ∨ ¬c)
(¬a ∨ ¬b ∨ c)

a false
b false
c false
d false
Randomized SAT Solving
Randomized SAT Solving

\[(\neg a \lor b \lor c)\]
\[(a \lor c \lor d)\]
\[(a \lor c \lor \neg d)\]
\[(a \lor \neg c \lor d)\]
\[(\neg b \lor \neg c \lor d)\]
\[(\neg a \lor b \lor \neg c)\]
\[(\neg a \lor \neg b \lor c)\]
Randomized SAT Solving

(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬d)
(a ∨ ¬c ∨ d)
(a ∨ ¬c ∨ ¬d)
(¬b ∨ ¬c ∨ d)
(¬a ∨ b ∨ ¬c)
(¬a ∨ ¬b ∨ c)

a  false
b  false
c  true
d  false
Randomized SAT Solving

\[
\begin{align*}
&\neg a \lor b \lor c \\
&a \lor c \lor d \\
&a \lor c \lor \neg d \\
&a \lor \neg c \lor d \\
&\neg b \lor \neg c \lor d \\
&\neg a \lor b \lor \neg c \\
&\neg a \lor \neg b \lor c
\end{align*}
\]
Randomized SAT Solving

(¬a ∨ b ∨ c)
( a ∨ c ∨ d )
( a ∨ c ∨ ¬d )
( a ∨ ¬c ∨ d )
( a ∨ ¬c ∨ ¬d )
( ¬b ∨ ¬c ∨ d )
( ¬a ∨ b ∨ ¬c )
( ¬a ∨ ¬b ∨ c )
Randomized SAT Solving

\[ (\neg a \lor b \lor c) \]
\[ (a \lor c \lor d) \]
\[ (a \lor c \lor \neg d) \]
\[ (a \lor \neg c \lor d) \]
\[ (\neg b \lor \neg c \lor d) \]
\[ (\neg a \lor b \lor \neg c) \]
\[ (\neg a \lor \neg b \lor c) \]
Randomized SAT Solving

( \neg a \lor b \lor c )
( a \lor c \lor d )
( a \lor c \lor \neg d )
( a \lor \neg c \lor d )
( \neg b \lor \neg c \lor d )
( \neg a \lor b \lor \neg c )
( \neg a \lor b \lor c )
Randomized SAT Solving

(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬d)
(a ∨ ¬c ∨ d)
(a ∨ ¬c ∨ ¬d)
(¬b ∨ ¬c ∨ d)
(¬a ∨ b ∨ ¬c)
(¬a ∨ ¬b ∨ c)

| a | false |
| b | true  |
| c | true  |
| d | true  |
Randomized SAT Solving

\[
\begin{align*}
\neg a \lor b \lor c \\
a \lor c \lor d \\
a \lor \neg c \lor d \\
a \lor \neg c \lor \neg d \\
\neg b \lor \neg c \lor d \\
\neg a \lor b \lor \neg c \\
\neg a \lor \neg b \lor c
\end{align*}
\]
Randomized SAT Solving

(¬a ∨ b ∨ c )
(a ∨ c ∨ d )
(a ∨ c ∨ ¬d )
(a ∨ ¬c ∨ d )
(a ∨ ¬c ∨ ¬d )
(¬b ∨ ¬c ∨ d )
(¬a ∨ b ∨ ¬c )
(¬a ∨ ¬b ∨ c )

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>
Randomized SAT Solving

( ¬a v b v c )
( a v c v d )
( a v c v ¬d )
( a v ¬c v d )
( a v ¬c v ¬d )
( ¬b v ¬c v d )
( ¬a v b v ¬c )
( ¬a v ¬b v c )

a true
b true
c true
d true
Randomized SAT Solving

(¬a ∨ b ∨ c )
( a ∨ c ∨ d )
( a ∨ c ∨ ¬d )
( a ∨ ¬c ∨ d )
( a ∨ ¬c ∨ ¬d )
( ¬b ∨ ¬c ∨ d )
( ¬a ∨ b ∨ ¬c )
( ¬a ∨ ¬b ∨ c )

a  true
b  true
c  true
d  true
Randomized SAT Solving

\( \neg a \lor b \lor c \)
\( a \lor c \lor d \)
\( a \lor c \lor \neg d \)
\( a \lor \neg c \lor d \)
\( \neg b \lor \neg c \lor d \)
\( \neg a \lor b \lor \neg c \)
\( \neg a \lor \neg b \lor c \)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Randomized SAT Solving

- We can attempt to solve SAT randomly:
  - Make a completely random guess at a satisfying assignment.
  - If the formula is satisfiable, output true.
  - For each one-variable change possible:
    - If that change improves the assignment, make the change and repeat.
  - If stuck, make another completely random guess at a satisfying assignment.
  - After some number of tries, output probably not.
- Many industrial SAT solving algorithms (such as GSAT and WalkSAT) are based on this approach.
How hard is SAT?
How hard is SAT?

We'll see more on this later on...
Next Time

• Introduction to Computability Theory
  • Strings and decision problems.
  • DFAs.