Mathematical Logic

Part Three

Friday Four Square!
Today at 4:15PM, Outside Gates

Announcements

- Problem Set 3 due right now.
- Problem Set 4 goes out today.
  - Checkpoint due **Monday, February 4**.
  - Remainder due **Friday, February 8**.
  - Play around with propositional and first-order logic!

What is First-Order Logic?

- **First-order logic** is a logical system for reasoning about properties of objects.
- Augments the logical connectives from propositional logic with
  - **predicates** that describe properties of objects, and
  - **functions** that map objects to one another,
  - **quantifiers** that allow us to reason about multiple objects simultaneously.

Quantifiers

- A statement of the form **$\forall x.\ \psi$** asserts that for **every** choice of x in our domain, ψ is true.
- A statement of the form **$\exists x.\ \psi$** asserts that for **some** choice of x in our domain, ψ is true.
- The syntax

$$\forall x \in S.\ \varphi$$

$$\exists x \in S.\ \varphi$$

  is allowed for quantifying over sets.

Translating into First-Order Logic

- First-order logic has great expressive power and is often used to formally encode mathematical definitions.
- Let's go provide rigorous definitions for the terms we've been using so far.

Set Theory

"The **union** of two sets is the set containing all elements of both sets."

$$\forall S.\ (Set(S) \rightarrow \forall T.\ (Set(T) \rightarrow \forall x.\ (x \in S \cup T \leftrightarrow \boldsymbol{x \in S \ \vee \ x \in T})))$$

Set Theory

"The **intersection** of two sets is the set containing all elements of both sets."

$$\forall S.\ (Set(S) \rightarrow \forall T.\ (Set(T) \rightarrow \forall x.\ (x \in S \cap T \leftrightarrow \boldsymbol{x \in S \ \wedge \ x \in T})))$$

Relations

“R is a reflexive relation over A.”

$$\forall a \in A.\ a \mathrel{R} a$$

Relations

"R is a symmetric relation over A."

$$\forall a \in A.\ \forall b \in A.\ (a \mathrm{R} b \to b \mathrm{R} a)$$

Relations

"R is an antisymmetric relation over A."

$$\forall a \in A.\ \forall b \in A.\ (aRb \land bRa \to a = b)$$

Relations

"R is a transitive relation over A."

$$\forall a \in A.\ \forall b \in A.\ \forall c \in A.\ (aRb \land bRc \to aRc)$$

Negating Quantifiers

- We spent much of Monday's lecture discussing how to negate propositional constructs.
- How do we negate quantifiers?

An Extremely Important Table

	When is this true?	When is this false?
$\forall x.\ P(x)$	For any choice of x, P(x)	For some choice of x, ¬P(x)
$\exists x.\ P(x)$	For some choice of x, P(x)	For any choice of x, ¬P(x)
$\forall x.\ \neg P(x)$	For any choice of x, ¬P(x)	For some choice of x, P(x)
$\exists x.\ \neg P(x)$	For some choice of x, ¬P(x)	For any choice of x, P(x)

An Extremely Important Table

	When is this true?	When is this false?
$\forall x.\ P(x)$	For any choice of x, P(x)	$\exists x.\ \neg P(x)$
$\exists x.\ P(x)$	For some choice of x, P(x)	$\forall x.\ \neg P(x)$
$\forall x.\ \neg P(x)$	For any choice of x, ¬P(x)	$\exists x.\ P(x)$
$\exists x.\ \neg P(x)$	For some choice of x, ¬P(x)	$\forall x.\ P(x)$

Negating First-Order Statements

- Use the equivalences

$$\neg \forall x.\ \varphi \equiv \exists x.\ \neg \varphi$$

$$\neg \exists x.\ \varphi \equiv \forall x.\ \neg \varphi$$

  to negate quantifiers.
- Mechanically:
  - Push the negation across the quantifier.
  - Change the quantifier from ∀ to ∃ or vice-versa.
- Use techniques from propositional logic to negate connectives.

Analyzing Relations

"R is a binary relation over set A that is not reflexive"

$$\neg \forall a \in A.\ aRa$$
$$\exists a \in A.\ \neg aRa$$

"Some $a \in A$ is not related to itself by R."

Analyzing Relations

"R is a binary relation over A that is not antisymmetric"

$$\neg \forall x \in A.\ \forall y \in A.\ (xRy \land yRx \to x = y)$$
$$\exists x \in A.\ \neg \forall y \in A.\ (xRy \land yRx \to x = y)$$
$$\exists x \in A.\ \exists y \in A.\ \neg (xRy \land yRx \to x = y)$$
$$\exists x \in A.\ \exists y \in A.\ (xRy \land yRx \land \neg (x = y))$$
$$\exists x \in A.\ \exists y \in A.\ (xRy \land yRx \land x \neq y)$$

"Some $x \in A$ and $y \in A$ are related to one another by R, but are not equal"

A Useful Equivalence

- The following equivalences are useful when negating statements in first-order logic:

$$\neg(p \wedge q) \equiv p \rightarrow \neg q$$

$$\neg(p \rightarrow q) \equiv p \wedge \neg q$$

- These identities are useful when negating statements involving quantifiers.
  - $\wedge$ is used in existentially-quantified statements.
  - $\rightarrow$ is used in universally-quantified statements.

Negating Quantifiers

- What is the negation of the following statement?

$$\boldsymbol{\exists x.\ \forall y.\ (P(x)\ \wedge\ Q(y))}$$

- We can obtain it as follows:

$$\neg\exists x.\ \forall y.\ (P(x)\ \wedge\ Q(y))$$

$$\forall x.\neg\forall y.\ (P(x)\ \wedge\ Q(y))$$

$$\forall x.\ \exists y.\ \neg(P(x)\ \wedge\ Q(y))$$

$$\forall x.\ \exists y.\ (P(x) \rightarrow \neg Q(y))$$

Using the predicates

- *Tourn*(*T*), which states that *T* is a tournament,
- $p \in T$, which states that *p* is a player in tournament *T*, and
- *Beats*(p_1, p_2), which states that p_1 beat p_2,

Write a sentence in first-order logic that means “Every tournament has a tournament winner.”

Every tournament has a tournament winner.

$$
\begin{aligned}
&\forall T.\ (\mathit{Tourn}(T) \rightarrow \\
&\quad (\exists w \in T. \\
&\quad\quad (\forall p \in T. \\
&\quad\quad\quad (p \neq w \rightarrow \\
&\quad\quad\quad\quad (\mathit{Beats}(w, p) \vee \\
&\quad\quad\quad\quad\quad (\exists p' \in T. \\
&\quad\quad\quad\quad\quad\quad (\mathit{Beats}(w, p') \wedge \mathit{Beats}(p', p)) \\
&\quad\quad\quad\quad\quad) \\
&\quad\quad\quad\quad) \\
&\quad\quad\quad) \\
&\quad\quad) \\
&\quad) \\
&)
\end{aligned}
$$

Every tournament has a tournament winner.

$$
\begin{aligned}
&\neg \forall T.\ (Tourn(T) \rightarrow \\
&\quad (\exists w \in T. \\
&\quad\quad (\forall p \in T. \\
&\quad\quad\quad (p \neq w \rightarrow \\
&\quad\quad\quad\quad (Beats(w, p) \vee \\
&\quad\quad\quad\quad\quad (\exists p' \in T. \\
&\quad\quad\quad\quad\quad\quad (Beats(w, p') \wedge Beats(p', p)) \\
&\quad\quad\quad\quad\quad) \\
&\quad\quad\quad\quad) \\
&\quad\quad\quad) \\
&\quad\quad) \\
&\quad) \\
&)
\end{aligned}
$$

Every tournament has a tournament winner.

$$
\begin{aligned}
&\exists T.\ (\mathit{Tourn}(T) \wedge \\
&\quad (\forall w \in T. \\
&\quad\quad (\exists p \in T. \\
&\quad\quad\quad (p \neq w \wedge \\
&\quad\quad\quad\quad (\neg \mathit{Beats}(w, p) \wedge \\
&\quad\quad\quad\quad\quad (\forall p' \in T. \\
&\quad\quad\quad\quad\quad\quad (\mathit{Beats}(w, p') \rightarrow \neg \mathit{Beats}(p', p)) \\
&\quad\quad\quad\quad\quad) \\
&\quad\quad\quad\quad) \\
&\quad\quad\quad) \\
&\quad\quad) \\
&\quad) \\
&)
\end{aligned}
$$

Every tournament has a tournament winner.

$$
\begin{aligned}
&\exists T.\ (\mathit{Tourn}(T) \wedge \\
&\quad (\forall w \in T. \\
&\quad\quad (\exists p \in T. \\
&\quad\quad\quad (p \neq w \wedge \\
&\quad\quad\quad\quad (\neg \mathit{Beats}(w, p) \wedge \\
&\quad\quad\quad\quad\quad (\forall p' \in T. \\
&\quad\quad\quad\quad\quad\quad (\mathit{Beats}(w, p') \rightarrow \neg \mathit{Beats}(p', p)) \\
&\quad\quad\quad\quad\quad) \\
&\quad\quad\quad\quad) \\
&\quad\quad\quad) \\
&\quad\quad) \\
&\quad) \\
&)
\end{aligned}
$$

There is some tournament where
for each player w,
there is some other player p who w didn't beat and
for each player p'
if w beat p', then p' did not beat p.

Uniqueness

Uniqueness

- Often, statements have the form “there is a *unique* x such that …”
- Some sources use a **uniqueness quantifier** to express this:

$$\exists! n.\ P(n)$$

- However, it's possible to encode uniqueness using just the two quantifiers we've seen.

$$\exists! n.\ P(n) \equiv \exists n.\ (P(n) \wedge \forall m.\ (P(m) \to m = n))$$

- In CS103, do not use the ∃! quantifier. Just use ∃ and ∀.

Summary of First-Order Logic

- Predicates allow us to reason about different properties of the same object.
- Functions allow us to transform objects into one another.
- Quantifiers allow us to reason about properties of some or all objects.
- There are many useful identities for negating first-order formulae.

SAT Solving

$P(x)$ $\forall$

Back to Propositional Logic...

$\exists$

Is This Formula Ever True?

$$p \vee \neg p$$

Is This Formula Ever True?

$$p \wedge \neg p$$

Is This Formula Ever True?

$$(r \rightarrow s \rightarrow t) \wedge (s \rightarrow t \rightarrow r) \wedge (t \rightarrow r \rightarrow s) \wedge t \wedge \neg s$$

Is This Formula Ever True?

$$(x_0 \rightarrow (x_1 \leftrightarrow x_0)) \vee (x_2 \wedge x_1 \wedge \neg x_0) \vee (x_1 \rightarrow \neg x_1)$$

Satisfiability

- A propositional logic formula φ 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 φ that makes it evaluate to true is called a **satisfying assignment**.
- Similar terms:
  - φ is a **tautology** if *every* variable assignment is a satisfying assignment.
  - φ is **satisfiable** if *some* variable assignment is a satisfying assignment.
  - φ is **unsatisfiable** if *no* variable assignment is a satisfying assignment.

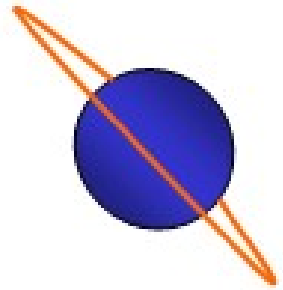
SAT

- The **boolean satisfiability problem** (**SAT**) is the following:

  Given a propositional logic formula φ, is φ satisfiable?

- Note: Goal is just to get a yes/no answer, not to actually find a satisfying assignment.
  - It's possible to extract a satisfying assignment if you know one exists; talk to me after class if you're curious.
- 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.

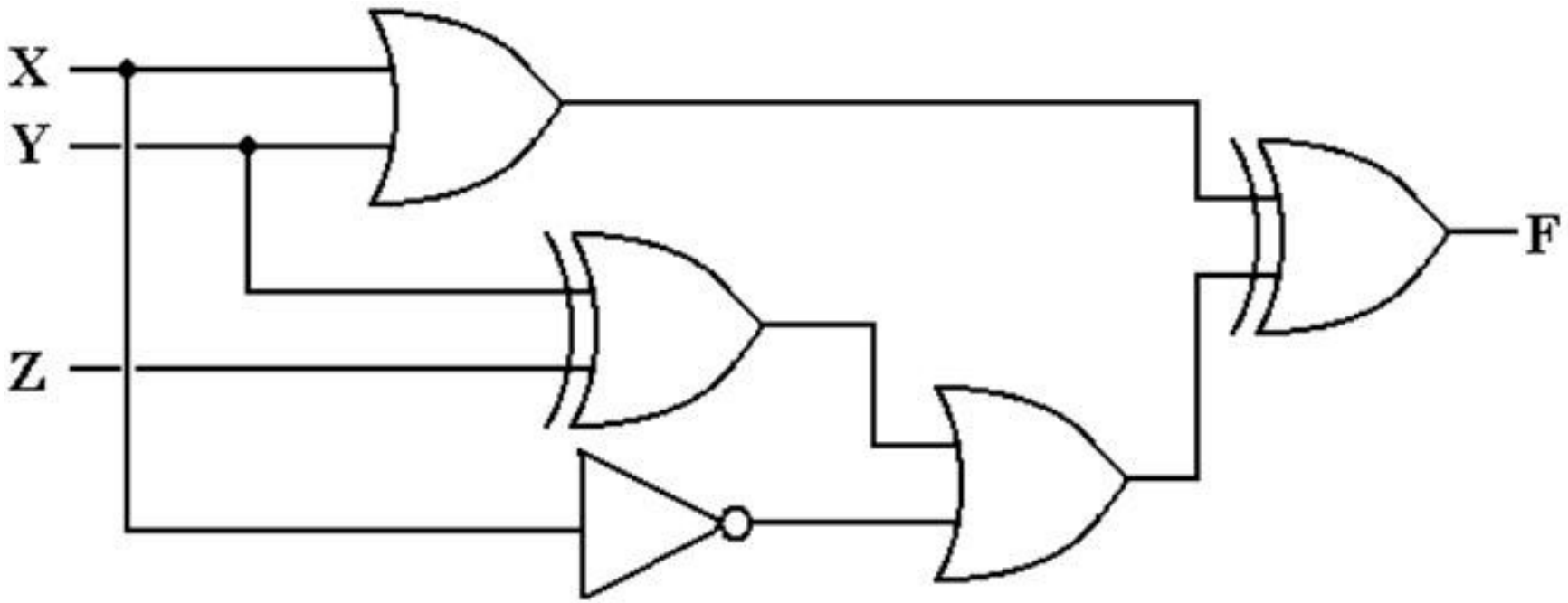
http://saturn.stanford.edu/

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 decsion 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.


X
Y
Z
F

Solving SAT: Take One

A Simple Algorithm

- Given a formula φ, we can just build a truth table for φ and check all of the rows.
- If any of them evaluate to true, then φ is satisfiable.
- If none of them evaluate to true, then φ is unsatisfiable.
- So what might this look like?

The Truth Table Algorithm

p	$p \vee \neg p$		
F	F	T	T
T	T	T	F

The Truth Table Algorithm

p	q	$(p \rightarrow q) \wedge q$		
F	F	T	F	F
F	T	T	T	T
T	F	F	F	F
T	T	T	T	T

p	q	r	$(r \leftrightarrow q)$	$\wedge$	r	$\wedge$	p	$\wedge$	$\neg q$
F	F	F	T	F	F	F	F		T
F	F	T	F	F	T	F	F		T
F	T	F	F	F	F	F	F		F
F	T	T	T	F	T	F	F		F
T	F	F	T	F	F	T	T		T
T	F	T	F	F	T	T	T		T
T	T	F	F	F	F	T	T		F
T	T	T	T	F	T	T	T		F

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.

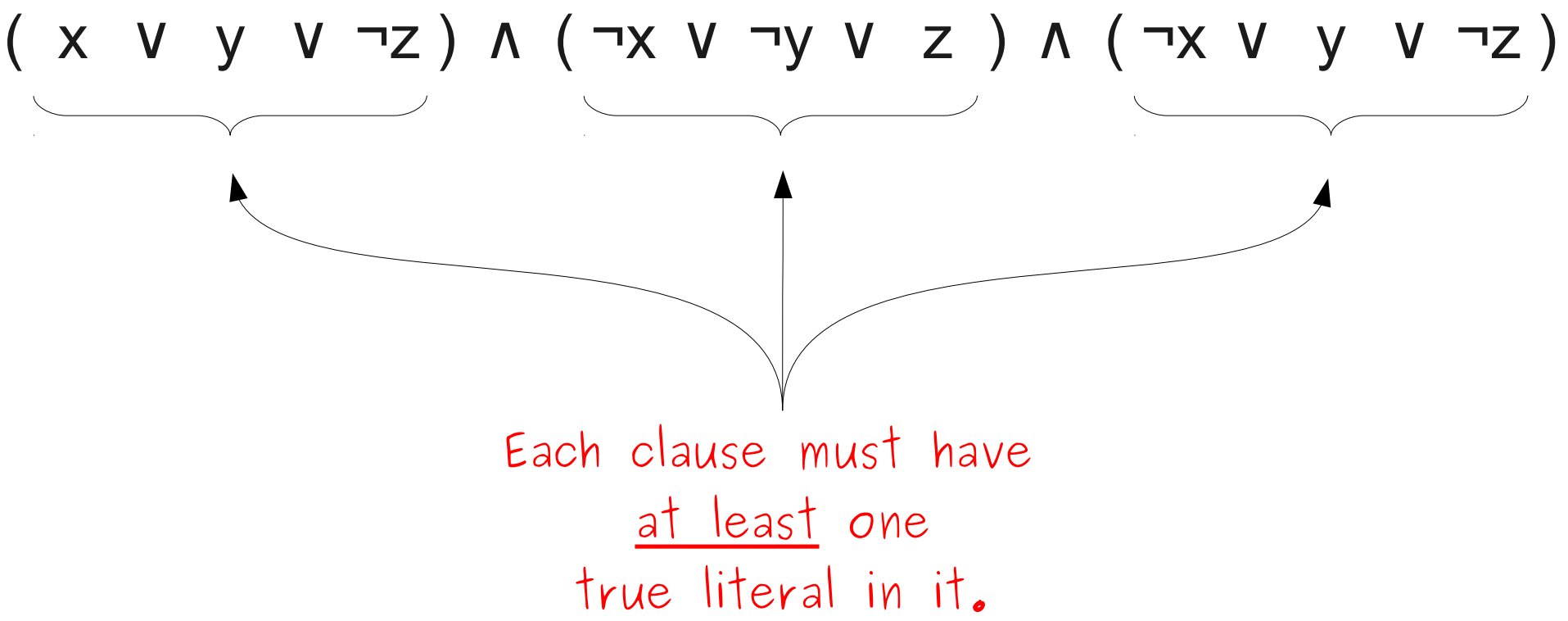
Literals and Clauses

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

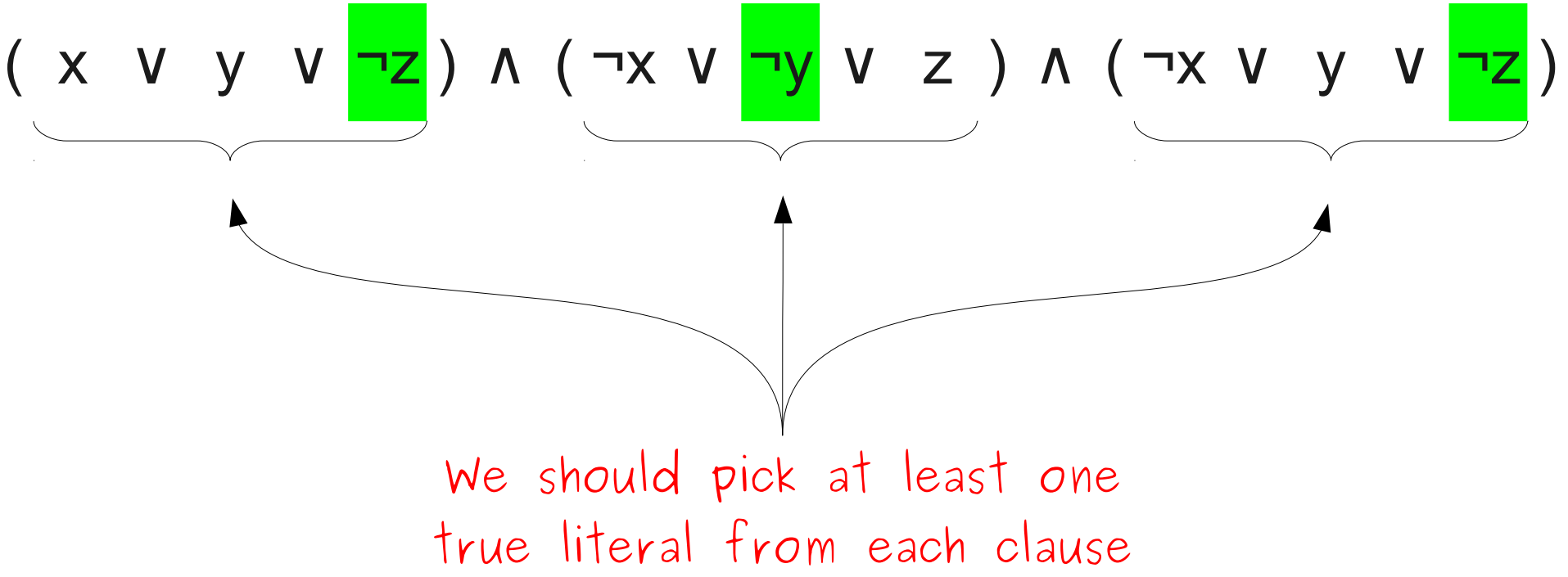
Conjunctive Normal Form

- A propositional logic formula φ is in **conjunctive normal form** (**CNF**) if it is the many-way AND (*conjunction*) of clauses.
  - $(x \vee y \vee z) \wedge (\neg x \vee \neg y) \wedge (x \vee y \vee z \vee \neg w)$
  - $x \vee z$
  - But not $(x \vee (y \wedge z)) \vee (x \vee y)$
- Only legal operators are $\neg$, $\vee$, $\wedge$.
- No nesting allowed.

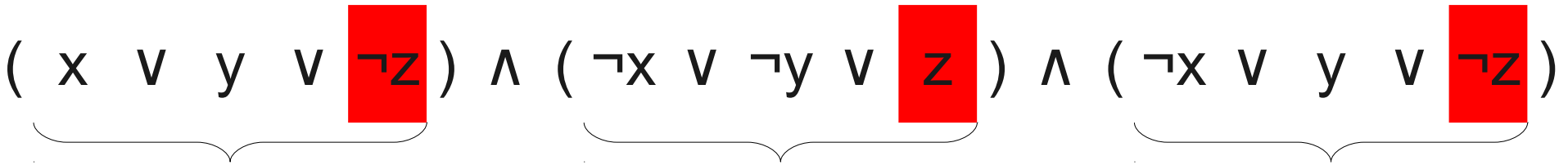
The Structure of CNF



The Structure of CNF



The Structure of CNF



… 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?
- Outline:
  - Turn any formula into a “mezzanine” format called NNF.
  - Convert NNF formulas into CNF formulas.
  - Details at the end of these slides.

SAT for CNF Formulas:
A Simple Backtracking Algorithm

(b v c)

(a v c v d)

(a v c v ¬d)

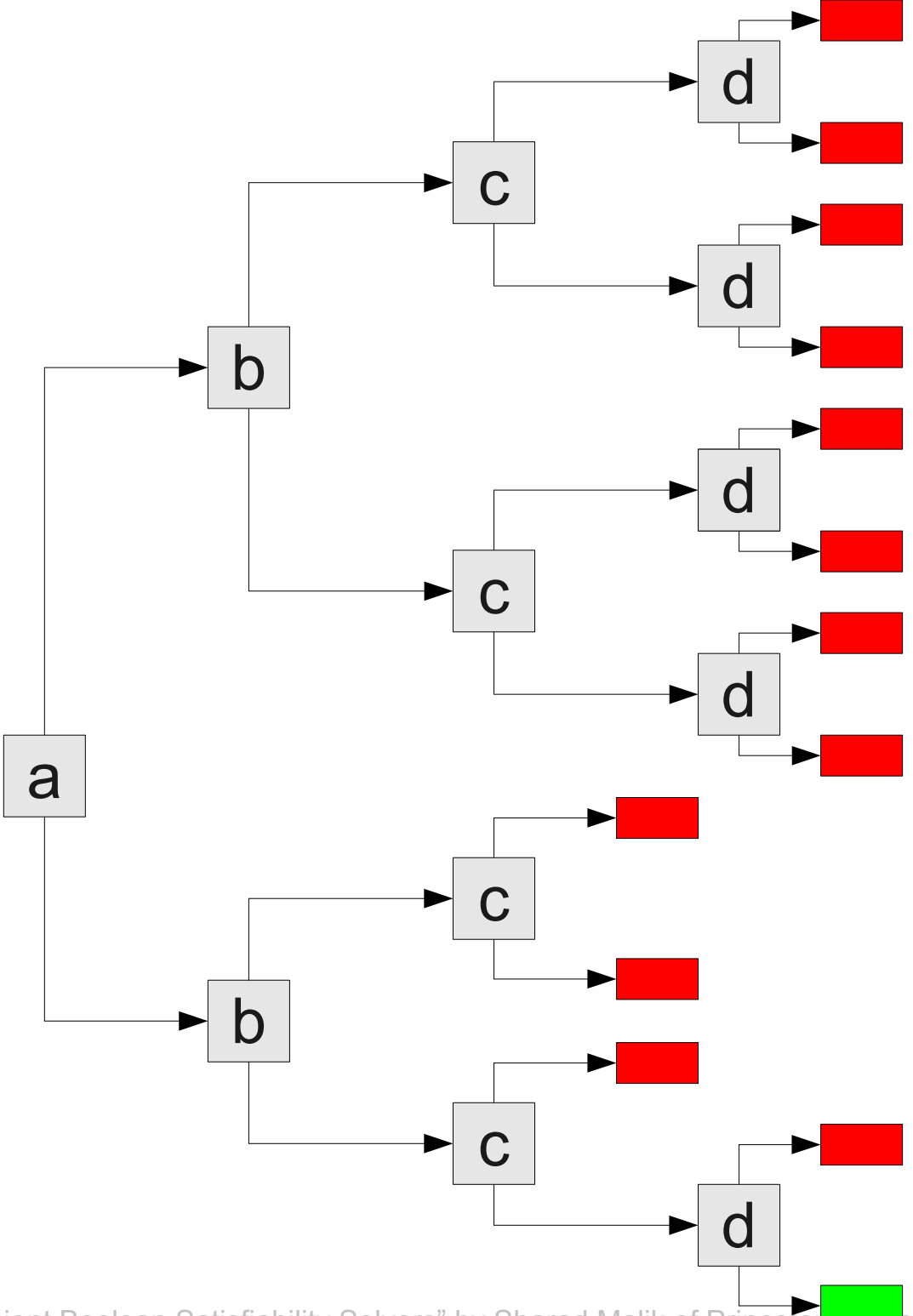
(a v ¬c v d)

(a v ¬c v ¬d)

(d)

(b v ¬c)

(c)



Backtracking SAT Solving

- If φ is an empty set of clauses, return **true**.
  - All clauses are satisfied.
- If φ contains ( ), return **false**.
  - Some clause is unsatisfiable.
- Otherwise:
  - Choose some variable *x*.
  - Return whether $\varphi_{\neg x}$ is satisfiable or φ_{x} is satisfiable, where $\varphi_{\neg x}$ is φ with *x* set to false and φ_{x} is φ 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 for n variables.

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

$$(\neg a \vee b \vee c) \wedge (a \vee c) \wedge (\neg a \vee \neg b) \wedge (\neg a \vee b \vee d) \wedge (\neg b \vee \neg d)$$

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

Pure Literal Elimination

$$(\neg a \vee \neg b) \wedge (\neg a \vee b \vee d) \wedge (\neg b \vee \neg d)$$

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

Pure Literal Elimination

(¬b ∨ ¬d)

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

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

$$(\neg a \vee b \vee c) \wedge (a \vee d) \wedge (\neg a \vee \neg b) \wedge (\neg a \vee b \vee d) \wedge \neg d$$

¬d is all by itself. d has to be false for this formula to be true.

Unit Propagation

$(\neg a \vee b \vee c) \wedge (\ a\ ) \wedge (\neg a \vee \neg b) \wedge (\neg a \vee b\ \ \ )$

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

Unit Propagation

(b ∨ c) ∧ (¬b) ∧ (b)

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 **D**avis, **P**utnam, **L**ogemann, and **L**oveland, its inventors.
- Incorporates the two heuristics we just saw.

DPLL

- Simplify φ with unit propagation.
- Simplify φ with pure literal elimination.
- If φ is empty, return **true**.
- If φ contains ( ), return **false**.
- Otherwise:
  - Choose some variable *x*.
  - Return whether $\varphi_{\neg x}$ is satisfiable or φ_{x} is satisfiable, where $\varphi_{\neg x}$ is φ with *x* set to false and φ_{x} is φ with *x* set to true.

DPLL is Powerful

- DPLL was invented 50 years and three months ago, 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.

How hard is SAT?

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

An Important Milestone

Recap: **Discrete Mathematics**

- The past four weeks have focused exclusively on discrete mathematics:

Induction	Functions
Graphs	The Pigeonhole Principle
Relations	Logic
Set Theory	Cardinality

- These are the building blocks we will use throughout the rest of the quarter.
- These are the building blocks you will use throughout the rest of your CS career.

Next Up: **Computability Theory**

- It's time to switch gears and address the limits of what can be computed.
- We'll explore
  - What is the formal definition of a computer?
  - What might computers look like with various resource constraints?
  - What problems can be solved by computers?
  - What problems ***can't*** be solved by computers?
- **Get ready to explore the boundaries of what computers could ever be made to do.**

Next Time

- **Formal Languages**
  - What is the mathematical definition of a problem?
- **Finite Automata**
  - What does a mathematical model of a computer look like?

Appendix: Getting to CNF

Negation Normal Form

- A formula φ in propositional logic is in **negation normal form** (**NNF**) iff
  - The only connectives are ¬, ∨, and ∧.
  - ¬ 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:
  - $\neg(a \wedge b)$
  - $a \rightarrow (\neg b \vee c)$
  - $\neg\neg a \vee \neg \mathrm{b} \wedge \neg c$

Getting to NNF

- NNF is a stepping stone toward CNF:
  - Only have ∨, ∧, and ¬.
  - 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 ∧, ∨, ¬.
- First step: Replace other connectives with these three.
  - Replace **φ → ψ** with **(¬φ ∨ ψ)**.
  - Can also replace ↔, ⊤, and ⊥ with formulas just using ∧, ∨, and ¬; you'll do this in the problem set. ☺
- Result: A new formula that is logically equivalent to the original and not much bigger.

Eliminating Complex Connectives

$$\neg(p \wedge q \wedge r) \rightarrow \neg(s \rightarrow \neg t \vee q)$$
$$\neg\neg(p \wedge q \wedge r) \vee \neg(s \rightarrow \neg t \vee q)$$
$$\neg\neg(p \wedge q \wedge r) \vee \neg(\neg s \vee \neg t \vee q)$$

Simplifying Negations

- NNF only allows negations in front of variables.
- Now that we have just ∨, ∧, and ¬, repeatedly apply these rules to achieve this result:
  - Replace **¬¬φ** with **φ**
  - Replace **¬(φ ∧ ψ)** with **¬φ ∨ ¬ψ**
  - Replace **¬(φ ∨ ψ)** with **¬φ ∧ ¬ψ**
- This process eventually terminates; the “height” of the negations keeps decreasing.

Simplifying Negations

$$\neg\neg(p \wedge q \wedge r) \vee \neg(\neg s \vee \neg t \vee q)$$
$$(p \wedge q \wedge r) \vee \neg(\neg s \vee \neg t \vee q)$$
$$(p \wedge q \wedge r) \vee (\neg\neg s \wedge \neg\neg t \wedge \neg q)$$
$$(p \wedge q \wedge r) \vee (s \wedge t \wedge \neg q)$$

From NNF to CNF

- Now that we can get to NNF, let's get down to CNF.
- Recall: CNF is the conjunction of clauses:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_4) \wedge \ldots$$

- 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: $\varphi \wedge \psi$
  - The disjunction of two NNF formulas: $\varphi \vee \psi$
- Let's work through some examples and see if we can find a pattern.

Examples: NNF to CNF

$$(x \vee y \vee z \vee \neg w) \wedge (\neg x \vee \neg y)$$

Examples: NNF to CNF

$$x \vee (y \wedge z)$$
$$(x \vee y) \wedge (x \vee z)$$

x	y	z	$x \vee (y \wedge z)$			$(x \vee y) \wedge (x \vee z)$		
F	F	F	F	F	F		F	F	F
F	F	T	F	F	F		F	F	T
F	T	F	F	F	F		T	F	F
F	T	T	F	T	T		T	T	T
T	F	F	T	T	F		T	T	T
T	F	T	T	T	F		T	T	T
T	T	F	T	T	F		T	T	T
T	T	T	T	T	T		T	T	T

Examples: NNF to CNF

$$(w \wedge x) \vee (y \wedge z)$$
$$((w \wedge x) \vee y) \wedge ((w \wedge x) \vee z)$$
$$(w \vee y) \wedge (x \vee y) \wedge (w \vee z) \wedge (x \vee z)$$

Examples: NNF to CNF

$$(v \wedge w \wedge x) \vee (y \wedge z)$$
$$((v \wedge w \wedge x) \vee y) \wedge ((v \wedge w \wedge x) \wedge z)$$
$$((v \wedge w) \vee y) \wedge (x \vee y) \wedge ((v \wedge w) \vee z) \wedge (x \vee z)$$
$$(v \vee y) \wedge (w \vee y) \wedge (x \vee y) \wedge (v \vee z) \wedge (w \vee z) \wedge (y \vee z)$$

Converting NNF to CNF

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

A Problem

$$(a \wedge b) \vee (c \wedge d) \vee (e \wedge f) \vee (g \wedge h)$$

$$((a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)) \vee (e \wedge f) \vee (g \wedge h)$$

$$((a \vee c \vee e) \wedge (a \vee c \vee f) \wedge (a \vee d \vee e) \wedge (a \vee d \vee f) \wedge (b \vee c \vee e) \wedge (b \vee c \vee f) \wedge (b \vee d \vee e) \wedge (b \vee d \vee f)) \vee (g \wedge h)$$

$$(a \vee c \vee e \vee g) \wedge (a \vee c \vee f \vee g) \wedge (a \vee d \vee e \vee g) \wedge (a \vee d \vee f \vee g) \wedge (b \vee c \vee e \vee g) \wedge (b \vee c \vee f \vee g) \wedge (b \vee d \vee e \vee g) \wedge (b \vee d \vee f \vee g) \wedge (a \vee c \vee e \vee h) \wedge (a \vee c \vee f \vee h) \wedge (a \vee d \vee e \vee h) \wedge (a \vee d \vee f \vee h) \wedge (b \vee c \vee e \vee h) \wedge (b \vee c \vee f \vee h) \wedge (b \vee d \vee e \vee h) \wedge (b \vee d \vee f \vee h)$$

Exponential Blowup

- Our logic for eliminating ∨ 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.

Equivalence and Equisatisfiability

- Recall: Two logical formulas φ and ψ are are **equivalent** (denoted **φ ≡ ψ**) if they always take on the same truth values.
- Two logical formulas φ and ψ are **equisatisfiable** (denoted **φ ≅ ψ**) if φ is satisfiable iff ψ is satisfiable.
  - Either φ and ψ are satisfiable, or φ and ψ are unsatisfiable.
- To solve SAT for a formula φ, we can instead solve SAT for an equisatisfiable ψ.

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \wedge b \wedge \neg c)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a) \wedge (b) \wedge (\neg c)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$(\neg a \wedge c)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$(\neg a) \wedge (c)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$(\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$\wedge$$

$$(\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$\wedge$$

$$(\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$\wedge$$

$$(\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$\wedge$$

$$(\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$\wedge$$

$$(\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$\wedge$$

$$(\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$\wedge$$

$$(\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$\wedge$$

$$(\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$\wedge$$

$$(\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q)$$

$$\wedge$$

$$(\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q))$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q \vee r) \wedge (b \vee q \vee r) \wedge (\neg c \vee q \vee r) \wedge (\neg a \vee \neg q \vee r) \wedge (c \vee \neg q \vee r)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q \vee r) \wedge (b \vee q \vee r) \wedge (\neg c \vee q \vee r) \wedge (\neg a \vee \neg q \vee r) \wedge (c \vee \neg q \vee r)$$

$$a \wedge \neg b$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q \vee r) \wedge (b \vee q \vee r) \wedge (\neg c \vee q \vee r) \wedge (\neg a \vee \neg q \vee r) \wedge (c \vee \neg q \vee r)$$

$$(a) \wedge (\neg b)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q \vee r) \wedge (b \vee q \vee r) \wedge (\neg c \vee q \vee r) \wedge (\neg a \vee \neg q \vee r) \wedge (c \vee \neg q \vee r)$$

$$(a \wedge \neg r) \wedge (\neg b \wedge \neg r)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q \vee r) \wedge (b \vee q \vee r) \wedge (\neg c \vee q \vee r) \wedge (\neg a \vee \neg q \vee r) \wedge (c \vee \neg q \vee r)$$

$$\wedge$$

$$(a \wedge \neg r) \wedge (\neg b \wedge \neg r)$$

Equisatisfiably from NNF to CNF

$$(\,(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)\,) \vee (\,a \wedge \neg b\,)$$

$$(\,a \vee q \vee r) \wedge (\,b \vee q \vee r) \wedge (\,\neg c \vee q \vee r) \wedge (\,\neg a \vee \neg q \vee r) \wedge (\,c \vee \neg q \vee r)$$

$$\wedge$$

$$(\,a \wedge \neg r\,) \wedge (\,\neg b \wedge \neg r\,)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q \vee r) \wedge (b \vee q \vee r) \wedge (\neg c \vee q \vee r) \wedge (\neg a \vee \neg q \vee r) \wedge (c \vee \neg q \vee r)$$

$$\wedge$$

$$(a \wedge \neg r) \wedge (\neg b \wedge \neg r)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q \vee r) \wedge (b \vee q \vee r) \wedge (\neg c \vee q \vee r) \wedge (\neg a \vee \neg q \vee r) \wedge (c \vee \neg q \vee r)$$

$$\wedge$$

$$(a \wedge \neg r) \wedge (\neg b \wedge \neg r)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q \vee r) \wedge (b \vee q \vee r) \wedge (\neg c \vee q \vee r) \wedge (\neg a \vee \neg q \vee r) \wedge (c \vee \neg q \vee r)$$

$$\wedge$$

$$(a \wedge \neg r) \wedge (\neg b \wedge \neg r)$$

Equisatisfiably from NNF to CNF

$$(\,(a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)\,) \vee (\,a \wedge \neg b\,)$$

$$(\,a \vee q \vee r) \wedge (\,b \vee q \vee r) \wedge (\,\neg c \vee q \vee r) \wedge (\,\neg a \vee \neg q \vee r) \wedge (\,c \vee \neg q \vee r)$$

$$\wedge$$

$$(\,a \wedge \neg r\,) \wedge (\,\neg b \wedge \neg r\,)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q \vee r) \wedge (b \vee q \vee r) \wedge (\neg c \vee q \vee r) \wedge (\neg a \vee \neg q \vee r) \wedge (c \vee \neg q \vee r)$$

$$\wedge$$

$$(a \wedge \neg r) \wedge (\neg b \wedge \neg r)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q \vee r) \wedge (b \vee q \vee r) \wedge (\neg c \vee q \vee r) \wedge (\neg a \vee \neg q \vee r) \wedge (c \vee \neg q \vee r)$$

$$\wedge$$

$$(a \wedge \neg r) \wedge (\neg b \wedge \neg r)$$

Equisatisfiably from NNF to CNF

$$((a \vee q) \wedge (b \vee q) \wedge (\neg c \vee q) \wedge (\neg a \vee \neg q) \wedge (c \vee \neg q)) \vee (a \wedge \neg b)$$

$$(a \vee q \vee r) \wedge (b \vee q \vee r) \wedge (\neg c \vee q \vee r) \wedge (\neg a \vee \neg q \vee r) \wedge (c \vee \neg q \vee r)$$

$$\wedge$$

$$(a \wedge \neg r) \wedge (\neg b \wedge \neg r)$$

Equisatisfiably from NNF to CNF

- If the formula is a literal, do nothing.
- If the formula is $\varphi \wedge \psi$:
  - Convert φ and ψ to CNF, call it φ' and ψ'.
  - Yield $\varphi' \wedge \psi'$
- If the formula is $\varphi \vee \psi$:
  - Convert φ and ψ to CNF, call it φ' and ψ'.
  - Create a new variable q.
  - Add q to each clause of φ'.
  - Add $\neg q$ to each clause of ψ'.
  - Yield $\varphi' \wedge \psi'$.
- Adds at most n new variables to each clause, where n is the number of clauses. Size increase at worst quadratic.