ECE750T-28: Computer-aided Reasoning for Software Engineering

Lecture 2: Normal Forms and DPLL

Vijay Ganesh
(Original notes from Isil Dillig)
Overview

► Last lecture:
  ► Two simple techniques for proving satisfiability and validity in propositional logic: truth tables and semantic argument
Overview

▶ Last lecture:
  ▶ Two simple techniques for proving satisfiability and validity in propositional logic: truth tables and semantic argument
  ▶ Neither very useful for practical automated reasoning
Overview

- **Last lecture:**
  - Two simple techniques for proving satisfiability and validity in propositional logic: truth tables and semantic argument
  - Neither very useful for practical automated reasoning

- **This Lecture:**
  - An algorithm called DPLL for determining satisfiability
Overview

▶ Last lecture:
  ▶ Two simple techniques for proving satisfiability and validity in propositional logic: truth tables and semantic argument
  ▶ Neither very useful for practical automated reasoning

▶ This Lecture:
  ▶ An algorithm called DPLL for determining satisfiability
  ▶ Many SAT solvers used today based on DPLL (more precisely, conflict-driven clause-learning)
Overview

- **Last lecture:**
  - Two simple techniques for proving satisfiability and validity in propositional logic: truth tables and semantic argument
  - Neither very useful for practical automated reasoning

- **This Lecture:**
  - An algorithm called DPLL for determining satisfiability
  - Many SAT solvers used today based on DPLL (more precisely, conflict-driven clause-learning)
  - However, requires converting formulas to a representation called normal forms
Overview

- **Last lecture:**
  - Two simple techniques for proving satisfiability and validity in propositional logic: truth tables and semantic argument
  - Neither very useful for practical automated reasoning

- **This Lecture:**
  - An algorithm called DPLL for determining satisfiability
  - Many SAT solvers used today based on DPLL (more precisely, conflict-driven clause-learning)
  - However, requires converting formulas to a representation called normal forms

- **The plan:** First talk about normal forms, then discuss DPLL
A normal form of a formula $F$ is another formula $F'$ such that $F$ is equivalent to $F'$, but $F'$ obeys certain syntactic restrictions.
Normal Forms

- A **normal form** of a formula $F$ is another formula $F'$ such that $F$ is equivalent to $F'$, but $F'$ obeys certain syntactic restrictions.

- There are three kinds of normal forms that are interesting in propositional logic:
Normal Forms

- A normal form of a formula $F$ is another formula $F'$ such that $F$ is equivalent to $F'$, but $F'$ obeys certain syntactic restrictions.

- There are three kinds of normal forms that are interesting in propositional logic:
Normal Forms

- A normal form of a formula $F$ is another formula $F'$ such that $F$ is equivalent to $F'$, but $F'$ obeys certain syntactic restrictions.

- There are three kinds of normal forms that are interesting in propositional logic:
  - Negation Normal Form (NNF)
Normal Forms

- A normal form of a formula $F$ is another formula $F'$ such that $F$ is equivalent to $F'$, but $F'$ obeys certain syntactic restrictions.

- There are three kinds of normal forms that are interesting in propositional logic:
  - Negation Normal Form (NNF)
  - Disjunctive Normal Form (DNF)
Normal Forms

- A normal form of a formula $F$ is another formula $F'$ such that $F$ is equivalent to $F'$, but $F'$ obeys certain syntactic restrictions.

- There are three kinds of normal forms that are interesting in propositional logic:
  - Negation Normal Form (NNF)
  - Disjunctive Normal Form (DNF)
  - Conjunctive Normal Form (CNF)
Negation Normal Form (NNF)

Negation Normal Form requires two syntactic restrictions:

- The only logical connectives are $\neg$, $\land$, $\lor$ (i.e., no $\rightarrow$, $\leftrightarrow$)
Negation Normal Form (NNF)

Negation Normal Form requires two syntactic restrictions:

- The only logical connectives are ¬, ∧, ∨ (i.e., no →, ↔)
- Negations appear only in literals
Negation Normal Form (NNF)

Negation Normal Form requires two syntactic restrictions:

- The only logical connectives are $\neg$, $\land$, $\lor$ (i.e., no $\rightarrow$, $\leftrightarrow$)
- Negations appear only in literals
- i.e., negations not allowed inside $\land$, $\lor$, or any other $\neg$
Negation Normal Form (NNF)

Negation Normal Form requires two syntactic restrictions:

- The only logical connectives are $\neg$, $\land$, $\lor$ (i.e., no $\rightarrow$, $\leftrightarrow$)
- Negations appear only in literals
- i.e., negations not allowed inside $\land$, $\lor$, or any other $\neg$
- i.e., negations can only appear in front of variables
Negation Normal Form (NNF)

Negation Normal Form requires two syntactic restrictions:

- The only logical connectives are ¬, ∧, ∨ (i.e., no →, ↔)
- Negations appear only in literals
- i.e., negations not allowed inside ∧, ∨, or any other ¬
- i.e., negations can only appear in front of variables
- Is formula \( p \lor (¬q \land (r \lor ¬s)) \) in NNF?
Negation Normal Form (NNF)

Negation Normal Form requires two syntactic restrictions:

- The only logical connectives are $\neg, \land, \lor$ (i.e., no $\rightarrow, \leftrightarrow$)
- Negations appear only in literals
- i.e., negations not allowed inside $\land, \lor$, or any other $\neg$
- i.e., negations can only appear in front of variables
- Is formula $p \lor (\neg q \land (r \lor \neg s))$ in NNF? Yes!
Negation Normal Form (NNF)

**Negation Normal Form** requires two syntactic restrictions:

- The only logical connectives are \( \neg, \land, \lor \) (i.e., no \( \rightarrow, \leftrightarrow \))
- Negations appear only in literals
  - i.e., negations not allowed inside \( \land, \lor \), or any other \( \neg \)
  - i.e., negations can only appear in front of variables
- Is formula \( p \lor (\neg q \land (r \lor \neg s)) \) in NNF? Yes!
- What about \( p \lor (\neg q \land \neg (\neg r \land s)) \)?
Negation Normal Form (NNF)

Negation Normal Form requires two syntactic restrictions:

- The only logical connectives are $\neg$, $\land$, $\lor$ (i.e., no $\rightarrow$, $\leftrightarrow$)
- Negations appear only in literals
- i.e., negations not allowed inside $\land$, $\lor$, or any other $\neg$
- i.e., negations can only appear in front of variables

- Is formula $p \lor (\neg q \land (r \lor \neg s))$ in NNF? Yes!
- What about $p \lor (\neg q \land \neg(\neg r \land s))$? No!
Negation Normal Form (NNF)

Negation Normal Form requires two syntactic restrictions:

- The only logical connectives are $\neg$, $\land$, $\lor$ (i.e., no $\rightarrow$, $\leftrightarrow$)
- Negations appear only in literals
  - i.e., negations not allowed inside $\land$, $\lor$, or any other $\neg$
  - i.e., negations can only appear in front of variables
- Is formula $p \lor (\neg q \land (r \lor \neg s))$ in NNF? Yes!
- What about $p \lor (\neg q \land \neg(\neg r \land s))$? No!
- What about $p \lor (\neg q \land (\neg\neg r \lor \neg s))$?
Negation Normal Form (NNF)

Negation Normal Form requires two syntactic restrictions:

- The only logical connectives are $\neg$, $\land$, $\lor$ (i.e., no $\rightarrow$, $\leftrightarrow$)
- Negations appear only in literals
  - i.e., negations not allowed inside $\land$, $\lor$, or any other $\neg$
  - i.e., negations can only appear in front of variables
- Is formula $p \lor (\neg q \land (r \lor \neg s))$ in NNF? Yes!
- What about $p \lor (\neg q \land \neg r \land s)$? No!
- What about $p \lor (\neg q \land (\neg \neg r \lor \neg s))$? No!
Conversion to NNF I

- To make sure the only logical connectives are $\neg, \wedge, \vee$, need to eliminate $\rightarrow$ and $\leftrightarrow$
Conversion to NNF I

- To make sure the only logical connectives are $\neg$, $\land$, $\lor$, need to eliminate $\rightarrow$ and $\leftrightarrow$.

- How do we express $F_1 \rightarrow F_2$ using $\lor$, $\land$, $\neg$?
Conversion to NNF I

- To make sure the only logical connectives are $\neg$, $\land$, $\lor$, need to eliminate $\to$ and $\leftrightarrow$

- How do we express $F_1 \to F_2$ using $\lor$, $\land$, $\neg$?

$$F_1 \to F_2 \iff \neg F_1 \lor F_2$$
Conversion to NNF I

- To make sure the only logical connectives are ¬, ∧, ∨, need to eliminate → and ⇔

- How do we express $F_1 \rightarrow F_2$ using ∨, ∧, ¬?

\[
F_1 \rightarrow F_2 \iff \neg F_1 \lor F_2
\]

- How do we express $F_1 \leftrightarrow F_2$ using only ¬, ∧, ∨?
Conversion to NNF I

▶ To make sure the only logical connectives are \(\neg, \land, \lor\), need to eliminate \(\to\) and \(\leftrightarrow\)

▶ How do we express \(F_1 \to F_2\) using \(\lor, \land, \neg\)?

\[
F_1 \to F_2 \iff \neg F_1 \lor F_2
\]

▶ How do we express \(F_1 \leftrightarrow F_2\) using only \(\neg, \land, \lor\)?

\[
F_1 \leftrightarrow F_2 \iff (\neg F_1 \lor F_2) \land (\neg F_2 \lor F_1)
\]
Conversion to NNF II

- Also need to ensure negations appear only in literals: push negations in

\[
\neg (F_1 \land F_2) \iff \neg F_1 \lor \neg F_2
\]

\[
\neg (F_1 \lor F_2) \iff \neg F_1 \land \neg F_2
\]

We also disallow double negations:

\[
\neg \neg F \iff F
\]
Conversion to NNF II

- Also need to ensure negations appear only in literals: push negations in

- Use DeMorgan’s laws to distribute $\neg$ over $\land$ and $\lor$:
Conversion to NNF II

- Also need to ensure negations appear only in literals: push negations in

- Use DeMorgan’s laws to distribute ¬ over ∧ and ∨:

\[ ¬(F_1 ∧ F_2) ⇔ ¬F_1 ∨ ¬F_2 \]
Conversion to NNF II

- Also need to ensure negations appear only in literals: push negations in

- Use DeMorgan’s laws to distribute $\neg$ over $\land$ and $\lor$:

\[ \neg(F_1 \land F_2) \iff \neg F_1 \lor \neg F_2 \]

\[ \neg(F_1 \lor F_2) \iff \neg F_1 \land \neg F_2 \]
Conversion to NNF II

- Also need to ensure negations appear only in literals: push negations in

- Use DeMorgan’s laws to distribute $\neg$ over $\land$ and $\lor$:

  $$\neg(F_1 \land F_2) \iff \neg F_1 \lor \neg F_2$$

  $$\neg(F_1 \lor F_2) \iff \neg F_1 \land \neg F_2$$

- We also disallow double negations:
Conversion to NNF II

- Also need to ensure negations appear only in literals: push negations in

- Use DeMorgan’s laws to distribute $\neg$ over $\land$ and $\lor$:
  
  $$\neg(F_1 \land F_2) \iff \neg F_1 \lor \neg F_2$$
  
  $$\neg(F_1 \lor F_2) \iff \neg F_1 \land \neg F_2$$

- We also disallow double negations:
  
  $$\neg\neg F \iff F$$
NNF Example

Convert $F : \neg(p \rightarrow (p \land q))$ to NNF
NNF Example

Convert $F : \neg(p \rightarrow (p \land q))$ to NNF

$$F_1 : \neg(\neg p \lor (p \land q))$$
NNF Example

Convert $F : \neg(p \rightarrow (p \land q))$ to NNF

$F_1 : \neg(\neg p \lor (p \land q))$

$F_2 : \neg\neg p \land \neg(p \land q)$
NNF Example

Convert $F : \neg(p \rightarrow (p \land q))$ to NNF

$F_1 : \neg(\neg p \lor (p \land q))$
$F_2 : \neg\neg p \land \neg(p \land q)$
$F_3 : \neg\neg p \land (\neg p \lor \neg q)$
NNF Example

Convert $F : \neg(p \to (p \land q))$ to NNF

$F_1 : \neg(\neg p \lor (p \land q))$
$F_2 : \neg\neg p \land \neg(p \land q)$
$F_3 : \neg\neg p \land (\neg p \lor \neg q)$
$F_4 : p \land (\neg p \lor \neg q)$
NNF Example

Convert \( F : \neg(p \rightarrow (p \land q)) \) to NNF

\[
\begin{align*}
F_1 : & \quad \neg(\neg p \lor (p \land q)) \\
F_2 : & \quad \neg \neg p \land \neg(p \land q) \\
F_3 : & \quad \neg \neg p \land (\neg p \lor \neg q) \\
F_4 : & \quad p \land (\neg p \lor \neg q)
\end{align*}
\]

\( F_4 \) is equivalent to \( F \) and is in NNF
Disjunctive Normal Form (DNF)

- A formula in **disjunctive normal form** is a disjunction of conjunction of literals.

\[ \bigvee_{i} \bigwedge_{j} \ell_{i,j} \quad \text{for literals } \ell_{i,j} \]
Disjunctive Normal Form (DNF)

- A formula in disjunctive normal form is a disjunction of conjunction of literals.

\[ \bigvee_{i} \bigwedge_{j} \ell_{i,j} \text{ for literals } \ell_{i,j} \]

- i.e., \( \lor \) can never appear inside \( \land \) or \( \neg \)
Disjunctive Normal Form (DNF)

- A formula in **disjunctive normal form** is a disjunction of conjunction of literals.

\[ \bigvee_i \bigwedge_j \ell_{i,j} \text{ for literals } \ell_{i,j} \]

- i.e., \( \lor \) can never appear inside \( \land \) or \( \neg \)

- Called disjunctive normal form because disjuncts are at the outer level
Disjunctive Normal Form (DNF)

- A formula in disjunctive normal form is a disjunction of conjunction of literals.
  \[ \bigvee_i \bigwedge_j \ell_{i,j} \text{ for literals } \ell_{i,j} \]

- i.e., \( \lor \) can never appear inside \( \land \) or \( \neg \)

- Called disjunctive normal form because disjuncts are at the outer level

- Each inner conjunction is called a clause
Disjunctive Normal Form (DNF)

- A formula in disjunctive normal form is a disjunction of conjunction of literals.

\[ \bigvee \bigwedge_{i,j} \ell_{i,j} \text{ for literals } \ell_{i,j} \]

- i.e., \( \lor \) can never appear inside \( \land \) or \( \neg \)

- Called disjunctive normal form because disjuncts are at the outer level

- Each inner conjunction is called a clause

- **Question:** If a formula is in DNF, is it also in NNF?
To convert formula to DNF, first convert it to NNF.
Conversion to DNF

- To convert formula to DNF, first convert it to NNF.
- Then, distribute $\land$ over $\lor$:
Conversion to DNF

- To convert formula to DNF, first convert it to NNF.
- Then, distribute $\land$ over $\lor$:

\[(F_1 \lor F_2) \land F_3\]
Conversion to DNF

- To convert formula to DNF, first convert it to NNF.

- Then, distribute $\land$ over $\lor$:

$$ (F_1 \lor F_2) \land F_3 \iff (F_1 \land F_3) \lor (F_2 \land F_3) $$
Conversion to DNF

- To convert formula to DNF, first convert it to NNF.

- Then, distribute $\wedge$ over $\vee$:

\[
(F_1 \lor F_2) \land F_3 \iff (F_1 \land F_3) \lor (F_2 \land F_3)
\]

\[
F_1 \land (F_2 \lor F_3)
\]
Conversion to DNF

- To convert formula to DNF, first convert it to NNF.
- Then, distribute $\land$ over $\lor$:

\[
(F_1 \lor F_2) \land F_3 \iff (F_1 \land F_3) \lor (F_2 \land F_3)
\]
\[
F_1 \land (F_2 \lor F_3) \iff (F_1 \land F_2) \lor (F_1 \land F_3)
\]
Example

Convert $F : (q_1 \lor \neg q_2) \land (\neg r_1 \rightarrow r_2)$ into DNF
Example

Convert $F : (q_1 \lor \neg \neg q_2) \land (\neg r_1 \to r_2)$ into DNF

$F_1 : (q_1 \lor \neg q_2) \land (\neg \neg r_1 \lor r_2)$

remove $\to$
Example

Convert $F : (q_1 \lor \neg q_2) \land (\neg r_1 \rightarrow r_2)$ into DNF

$F_1 : (q_1 \lor \neg q_2) \land (\neg
r_1 \lor r_2)$

$F_2 : (q_1 \lor q_2) \land (r_1 \lor r_2)$

remove $\rightarrow$

in NNF
Example

Convert $F : (q_1 \lor \neg q_2) \land (\neg r_1 \rightarrow r_2)$ into DNF

$F_1 : (q_1 \lor \neg q_2) \land (\neg \neg r_1 \lor r_2)$
$F_2 : (q_1 \lor q_2) \land (r_1 \lor r_2)$
$F_3 : (q_1 \land (r_1 \lor r_2)) \lor (q_2 \land (r_1 \lor r_2))$

remove $\rightarrow$

in NNF

dist

equivalent to $F$ and is in DNF
Example

Convert $F : (q_1 \lor \neg \neg q_2) \land (\neg r_1 \rightarrow r_2)$ into DNF

$F_1 : (q_1 \lor \neg \neg q_2) \land (\neg \neg r_1 \lor r_2)$

remove $\rightarrow$

$F_2 : (q_1 \lor q_2) \land (r_1 \lor r_2)$

in NNF

$F_3 : (q_1 \land (r_1 \lor r_2)) \lor (q_2 \land (r_1 \lor r_2))$

dist

$F_4 : (q_1 \land r_1) \lor (q_1 \land r_2) \lor (q_2 \land r_1) \lor (q_2 \land r_2)$

dist
Example

Convert $F : (q_1 \lor \neg
\neg q_2) \land (\neg r_1 \rightarrow r_2)$ into DNF

\[
F_1 : (q_1 \lor \neg
\neg q_2) \land (\neg
\neg r_1 \lor r_2) \\
F_2 : (q_1 \lor q_2) \land (r_1 \lor r_2) \\
F_3 : (q_1 \land (r_1 \lor r_2)) \lor (q_2 \land (r_1 \lor r_2)) \\
F_4 : (q_1 \land r_1) \lor (q_1 \land r_2) \lor (q_2 \land r_1) \lor (q_2 \land r_2)
\]

remove $\rightarrow$
in NNF
dist
dist

$F_4$ equivalent to $F$ and is in DNF
Claim: If formula is in DNF, trivial to determine satisfiability. How?

Since disjunction of clauses, formula is satisfied if any clause is satisfied.
If there is any clause that neither contains $\bot$ nor a literal and its negation, then the formula is satisfiable.
Idea: To determine satisfiability, convert formula to DNF and just do a syntactic check.
Claim: If formula is in DNF, trivial to determine satisfiability. How?

Since disjunction of clauses, formula is satisfied if any clause is satisfied.
Claim: If formula is in DNF, trivial to determine satisfiability. How?

Since disjunction of clauses, formula is satisfied if any clause is satisfied.

If there is any clause that neither contains ⊥ nor a literal and is and its negation, then the formula is satisfiable.
Claim: If formula is in DNF, trivial to determine satisfiability. How?

Since disjunction of clauses, formula is satisfied if any clause is satisfied.

If there is any clause that neither contains ⊥ nor a literal and is and its negation, then the formula is satisfiable.

Idea: To determine satisfiability, convert formula to DNF and just do a syntactic check.
DNF and Blow-up in formula size

- This idea is completely impractical. Why?
DNF and Blow-up in formula size

- This idea is completely impractical. Why?
- Consider formula: \((F_1 \lor F_2) \land (F_3 \lor F_4)\)
This idea is completely impractical. Why?

Consider formula: \((F_1 \lor F_2) \land (F_3 \lor F_4)\)

In DNF:
\[
(F_1 \land F_3) \lor (F_1 \land F_4) \lor (F_2 \land F_3) \lor (F_2 \land F_4)
\]
This idea is completely impractical. Why?

Consider formula: \((F_1 \lor F_2) \land (F_3 \lor F_4)\)

In DNF:

\[(F_1 \land F_3) \lor (F_1 \land F_4) \lor (F_2 \land F_3) \lor (F_2 \land F_4)\]
DNF and Blow-up in formula size

- This idea is completely impractical. Why?

- Consider formula: \((F_1 \lor F_2) \land (F_3 \lor F_4)\)

- In DNF:

\[
(F_1 \land F_3) \lor (F_1 \land F_4) \lor (F_2 \land F_3) \lor (F_2 \land F_4)
\]

- Every time we distribute, formula size doubles!
This idea is completely impractical. Why?

Consider formula: \((F_1 \lor F_2) \land (F_3 \lor F_4)\)

In DNF:

\[
(F_1 \land F_3) \lor (F_1 \land F_4) \lor (F_2 \land F_3) \lor (F_2 \land F_4)
\]

Every time we distribute, formula size doubles!

Moral: DNF conversion causes exponential blow-up in size!
This idea is completely impractical. Why?

Consider formula: \((F_1 \lor F_2) \land (F_3 \lor F_4)\)

In DNF:

\[(F_1 \land F_3) \lor (F_1 \land F_4) \lor (F_2 \land F_3) \lor (F_2 \land F_4)\]

Every time we distribute, formula size doubles!

Moral: DNF conversion causes exponential blow-up in size!

Checking satisfiability by converting to DNF is almost as bad as truth tables!
Conjunctive Normal Form (CNF)

- A formula in **conjuctive normal form** is a conjunction of disjunction of literals.

\[ \bigwedge_i \bigvee_j \ell_{i,j} \text{ for literals } \ell_{i,j} \]
Conjunctive Normal Form (CNF)

- A formula in **conjunctive normal form** is a conjunction of disjunction of literals.

\[ \bigwedge_i \bigvee_j \ell_{i,j} \text{ for literals } \ell_{i,j} \]

- i.e., \( \land \) not allowed inside \( \lor, \neg \).
A formula in **conjunctive normal form** is a conjunction of disjunction of literals.

\[ \bigwedge_i \bigvee_j \ell_{i,j} \quad \text{for literals } \ell_{i,j} \]

i.e., \( \wedge \) not allowed inside \( \vee, \neg \).

Called conjunctive normal form because conjuncts are at the outer level.
Conjunctive Normal Form (CNF)

- A formula in conjunctive normal form is a conjunction of disjunction of literals.

\[ \bigwedge \bigvee_{i,j} \ell_{i,j} \text{ for literals } \ell_{i,j} \]

- i.e., \( \wedge \) not allowed inside \( \vee, \neg \).

- Called conjunctive normal form because conjuncts are at the outer level

- Each inner disjunction is called a clause
Conjunctive Normal Form (CNF)

- A formula in **conjunctive normal form** is a conjunction of disjunction of literals.

  \[ \bigwedge \bigvee_{i,j} \ell_{i,j} \text{ for literals } \ell_{i,j} \]

- i.e., \( \land \) not allowed inside \( \lor, \neg \).

- Called conjunctive normal form because conjuncts are at the outer level.

- Each inner disjunction is called a **clause**.

- Is formula in CNF also in NNF?
Conversion to CNF

- To convert formula to CNF, first convert it to NNF.
Conversion to CNF

- To convert formula to CNF, first convert it to NNF.

- Then, distribute $\lor$ over $\land$:
Conversion to CNF

- To convert formula to CNF, first convert it to NNF.

- Then, distribute $\lor$ over $\land$:

$$(F_1 \land F_2) \lor F_3$$
Conversion to CNF

- To convert formula to CNF, first convert it to NNF.

- Then, distribute \( \lor \) over \( \land \):

\[
(F_1 \land F_2) \lor F_3 \iff (F_1 \lor F_3) \land (F_2 \lor F_3)
\]
Conversion to CNF

- To convert formula to CNF, first convert it to NNF.

- Then, distribute \( \lor \) over \( \land \):

\[
(F_1 \land F_2) \lor F_3 \iff (F_1 \lor F_3) \land (F_2 \lor F_3)
\]

\[
F_1 \lor (F_2 \land F_3)
\]
Conversion to CNF

- To convert formula to CNF, first convert it to NNF.
- Then, distribute $\lor$ over $\land$:

\[
(F_1 \land F_2) \lor F_3 \iff (F_1 \lor F_3) \land (F_2 \lor F_3)
\]
\[
F_1 \lor (F_2 \land F_3) \iff (F_1 \lor F_2) \land (F_1 \lor F_3)
\]
CNF Conversion Example

Convert $F : (p \leftrightarrow (q \rightarrow r))$ into CNF
CNF Conversion Example

Convert $F : (p \leftrightarrow (q \rightarrow r))$ into CNF

$$F_1 : (p \rightarrow (q \rightarrow r)) \land ((q \rightarrow r) \rightarrow p) \quad \text{remove } \leftrightarrow$$
Convert $F : (p \leftrightarrow (q \rightarrow r))$ into CNF

\[ F_1 : (p \rightarrow (q \rightarrow r)) \land ((q \rightarrow r) \rightarrow p) \quad \text{remove } \leftrightarrow \]
\[ F_2 : (\neg p \lor (q \rightarrow r)) \land (\neg (q \rightarrow r) \lor p) \quad \text{remove } \rightarrow \]
CNF Conversion Example

Convert $F : (p \leftrightarrow (q \rightarrow r))$ into CNF

$F_1 : (p \rightarrow (q \rightarrow r)) \land ((q \rightarrow r) \rightarrow p)$  remove $\leftrightarrow$

$F_2 : (\neg p \lor (q \rightarrow r)) \land (\neg (q \rightarrow r) \lor p)$  remove $\rightarrow$

$F_3 : (\neg p \lor (\neg q \lor r)) \land (\neg (\neg q \lor r) \lor p)$  remove $\rightarrow$
Convert $F : (p \leftrightarrow (q \rightarrow r))$ into CNF

$F_1 : (p \rightarrow (q \rightarrow r)) \land ((q \rightarrow r) \rightarrow p)$ \hspace{1cm} \text{remove } \leftrightarrow

$F_2 : (\neg p \lor (q \rightarrow r)) \land (\neg (q \rightarrow r) \lor p)$ \hspace{1cm} \text{remove } \rightarrow

$F_3 : (\neg p \lor (\neg q \lor r)) \land (\neg (\neg q \lor r) \lor p)$ \hspace{1cm} \text{remove } \rightarrow

$F_4 : (\neg p \lor \neg q \lor r) \land ((q \land \neg r) \lor p)$ \hspace{1cm} \text{De Morgan}
CNF Conversion Example

Convert $F : (p \leftrightarrow (q \rightarrow r))$ into CNF

$F_1 : (p \rightarrow (q \rightarrow r)) \land ((q \rightarrow r) \rightarrow p)$  \hspace{1cm} \text{remove $\leftrightarrow$}

$F_2 : (\neg p \lor (q \rightarrow r)) \land (\neg (q \rightarrow r) \lor p)$  \hspace{1cm} \text{remove $\rightarrow$}

$F_3 : (\neg p \lor (\neg q \lor r)) \land (\neg (\neg q \lor r) \lor p)$  \hspace{1cm} \text{remove $\rightarrow$}

$F_4 : (\neg p \lor \neg q \lor r) \land ((q \land \neg r) \lor p)$ \hspace{1cm} \text{De Morgan}

$F_5 : (\neg p \lor \neg q \lor r) \land (q \lor p) \land (\neg r \lor p)$ \hspace{1cm} \text{Distribute $\lor$ over $\land$}
CNF Conversion Example

Convert $F : (p \leftrightarrow (q \rightarrow r))$ into CNF

$F_1 : (p \rightarrow (q \rightarrow r)) \land ((q \rightarrow r) \rightarrow p)$  \hspace{1cm} \text{remove} \leftrightarrow

$F_2 : (\neg p \lor (q \rightarrow r)) \land (\neg (q \rightarrow r) \lor p)$  \hspace{1cm} \text{remove} \rightarrow

$F_3 : (\neg p \lor (\neg q \lor r)) \land (\neg (\neg q \lor r) \lor p)$  \hspace{1cm} \text{remove} \rightarrow

$F_4 : (\neg p \lor \neg q \lor r) \land ((q \land \neg r) \lor p)$  \hspace{1cm} \text{De Morgan}

$F_5 : (\neg p \lor \neg q \lor r) \land (q \lor p) \land (\neg r \lor p)$  \hspace{1cm} \text{Distribute} \lor \text{over} \land

$F_5$ is equivalent to $F$ and is in CNF
DNF vs. CNF

- **Fact:** Unlike DNF, it is not trivial to determine satisfiability of formula in CNF.
DNF vs. CNF

- **Fact:** Unlike DNF, it is not trivial to determine satisfiability of formula in CNF.

- Does CNF conversion cause exponential blow-up in size?
DNF vs. CNF

▶ **Fact:** Unlike DNF, it is not trivial to determine satisfiability of formula in CNF.

▶ Does CNF conversion cause exponential blow-up in size? Yes
DNF vs. CNF

- **Fact**: Unlike DNF, it is not trivial to determine satisfiability of formula in CNF.

- Does CNF conversion cause exponential blow-up in size? **Yes**

- **News**: But almost all SAT solvers first convert formula to CNF before solving!
Why CNF?

- **Interesting Question**: If it is just as expensive to convert formula to CNF as to DNF, why do solvers convert to CNF although it is much easier to determine satisfiability in DNF?
Why CNF?

- **Interesting Question**: If it is just as expensive to convert formula to CNF as to DNF, why do solvers convert to CNF although it is much easier to determine satisfiability in DNF?

- **Two reasons**:
Why CNF?

- **Interesting Question:** If it is just as expensive to convert formula to CNF as to DNF, why do solvers convert to CNF although it is much easier to determine satisfiability in DNF?

- **Two reasons:**
  
  1. Possible to convert to equisatisfiable (not equivalent) CNF formula with only linear increase in size!
Why CNF?

▶ **Interesting Question:** If it is just as expensive to convert formula to CNF as to DNF, why do solvers convert to CNF although it is much easier to determine satisfiability in DNF?

▶ **Two reasons:**

1. Possible to convert to **equisatisfiable** (not equivalent) CNF formula with only linear increase in size!

2. CNF makes it possible to perform interesting deductions (resolution)
Equisatisfiability

- Two formulas $F$ and $F'$ are equisatisfiable iff:

\[
F \text{ is satisfiable if and only if } F' \text{ is satisfiable}
\]
Equisatisfiability

- Two formulas $F$ and $F'$ are equisatisfiable iff:

  $F$ is satisfiable if and only if $F'$ is satisfiable

- If two formulas are equisatisfiable, are they equivalent?
Equisatisfiability

- Two formulas $F$ and $F'$ are equisatisfiable iff:

  $F$ is satisfiable if and only if $F'$ is satisfiable

- If two formulas are equisatisfiable, are they equivalent? No!

- Example:
Equisatisfiability

- Two formulas $F$ and $F'$ are **equisatisfiable** iff:
  
  $F$ is satisfiable if and only if $F'$ is satisfiable

- If two formulas are equisatisfiable, are they equivalent? **No!**

- **Example:** Any satisfiable formula (e.g., $p$) is equisat as $\top$
Equisatisfiability

- Two formulas $F$ and $F'$ are **equisatisfiable** iff:
  
  \[
  \text{\textit{F} is satisfiable if and only if \textit{F'} is satisfiable}
  \]

- If two formulas are equisatisfiable, are they equivalent? **No!**

- **Example:** Any satisfiable formula (e.g., $p$) is equisat as $\top$

- But clearly, $p$ is not equivalent to $\top$! Why?
Equisatisfiability

- Two formulas $F$ and $F'$ are **equisatisfiable** iff:

  \[
  F \text{ is satisfiable if and only if } F' \text{ is satisfiable}
  \]

- If two formulas are equisatisfiable, are they equivalent? **No!**

- **Example:** Any satisfiable formula (e.g., $p$) is equisat as $\top$

- But clearly, $p$ is not equivalent to $\top$! Why?

- Equisatisfiability is a much weaker notion than equivalence.
Equisatisfiability

- Two formulas $F$ and $F'$ are equisatisfiable iff:

$$F \text{ is satisfiable if and only if } F' \text{ is satisfiable}$$

- If two formulas are equisatisfiable, are they equivalent? **No!**

- **Example:** Any satisfiable formula (e.g., $p$) is equisat as $\top$

- But clearly, $p$ is not equivalent to $\top$! Why?

- Equisatisfiability is a much weaker notion than equivalence.

- But useful if all we want to do is determine satisfiability.
The Plan

- To determine satisfiability of $F$, convert formula to equisatisfiable formula $F'$ in CNF.
The Plan

- To determine satisfiability of $F$, convert formula to equisatisfiable formula $F'$ in CNF
- Use an algorithm (DPLL) to decide satisfiability of $F'$
The Plan

- To determine satisfiability of $F$, convert formula to equisatisfiable formula $F'$ in CNF

- Use an algorithm (DPLL) to decide satisfiability of $F'$

- Since $F'$ is equisatisfiable to $F$, $F$ is satifiable iff algorithm decides $F'$ is satisfiable
The Plan

- To determine satisfiability of $F$, convert formula to equisatisfiable formula $F'$ in CNF

- Use an algorithm (DPLL) to decide satisfiability of $F'$

- Since $F'$ is equisatisfiable to $F$, $F$ is satisfiable iff algorithm decides $F'$ is satisfiable

- Big question: How do we convert formula to equisatisfiable formula without causing exponential blow-up in size?
Tseitin’s transformation converts formula $F$ to equisatisfiable formula $F'$ in CNF with only a linear increase in size.
Tseitin’s Transformation I

▶ **Step 1:** Introduce a new variable $p_G$ for every subformula $G$ of $F$ (unless $G$ is already an atom).
Step 1: Introduce a new variable $p_G$ for every subformula $G$ of $F$ (unless $G$ is already an atom).

For instance, if $F = G_1 \land G_2$, introduce two variables $p_{G_1}$ and $p_{G_2}$ representing $G_1$ and $G_2$ respectively.
Step 1: Introduce a new variable $p_G$ for every subformula $G$ of $F$ (unless $G$ is already an atom).

For instance, if $F = G_1 \land G_2$, introduce two variables $p_{G_1}$ and $p_{G_2}$ representing $G_1$ and $G_2$ respectively.

$p_{G_1}$ is said to be representative of $G_1$ and $p_{G_2}$ is representative of $G_2$. 
Tseitin’s Transformation II

- **Step 2**: Consider each subformula

\[ G : G_1 \circ G_2 \quad (\circ \text{ arbitrary boolean connective}) \]
Tseitin’s Transformation II

▶ Step 2: Consider each subformula

\[ G : G_1 \circ G_2 \quad (\circ \text{ arbitrary boolean connective}) \]

▶ Stipulate representative of \( G \) is equivalent to representative of \( G_1 \circ G_2 \)

\[ p_G \iff p_{G_1} \circ p_{G_2} \]
Tseitin’s Transformation II

- **Step 2**: Consider each subformula

  \[ G : G_1 \circ G_2 \quad (\circ \text{arbitrary boolean connective}) \]

- Stipulate representative of \( G \) is equivalent to representative of \( G_1 \circ G_2 \)

  \[ p_G \leftrightarrow p_{G_1} \circ p_{G_2} \]
Tseitin's Transformation II

- **Step 2:** Consider each subformula

  \[ G : G_1 \circ G_2 \quad (\circ \text{ arbitrary boolean connective}) \]

  - Stipulate representative of \( G \) is equivalent to representative of \( G_1 \circ G_2 \)

    \[ p_G \leftrightarrow p_{G_1} \circ p_{G_2} \]

- **Step 3:** Convert \( p_G \leftrightarrow p_{G_1} \circ p_{G_2} \) to equivalent CNF (by converting to NNF and distributing ∨’s over ∧’s).
Step 2: Consider each subformula

\[ G : G_1 \circ G_2 \quad (\circ \text{ arbitrary boolean connective}) \]

Stipulate representative of \( G \) is equivalent to representative of \( G_1 \circ G_2 \)

\[ p_G \iff p_{G_1} \circ p_{G_2} \]

Step 3: Convert \( p_G \iff p_{G_1} \circ p_{G_2} \) to equivalent CNF (by converting to NNF and distributing \( \lor \)’s over \( \land \)’s).

Observe: Since \( p_G \iff p_{G_1} \circ p_{G_2} \) contains at most three propositional variables and exactly two connectives, size of this formula in CNF is bound by a constant.
Tseitin’s Transformation II

Given original formula $F$, let $p_F$ be its representative and let $S_F$ be the set of all subformulas of $F$ (including $F$ itself).
Given original formula $F$, let $p_F$ be its representative and let $S_F$ be the set of all subformulas of $F$ (including $F$ itself).

Then, introduce the formula

$$p_F \land \bigwedge_{G=(G_1 \circ G_2) \in S_F} \text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$$

Claim: This formula is equisatisfiable to $F$. The proof is by structural induction. Formula is also in CNF because conjunction of CNF formulas is in CNF.
Tseitin’s Transformation II

- Given original formula $F$, let $p_F$ be its representative and let $S_F$ be the set of all subformulas of $F$ (including $F$ itself).

- Then, introduce the formula

$$p_F \land \bigwedge_{G = (G_1 \circ G_2) \in S_F} \text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$$

- **Claim:** This formula is equisatisfiable to $F$. 
Given original formula $F$, let $p_F$ be its representative and let $S_F$ be the set of all subformulas of $F$ (including $F$ itself).

Then, introduce the formula

$$p_F \land \bigwedge_{G=(G_1 \circ G_2) \in S_F} \text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$$

Claim: This formula is equisatisfiable to $F$.

The proof is by structural induction.
Tseitin’s Transformation II

Given original formula $F$, let $p_F$ be its representative and let $S_F$ be the set of all subformulas of $F$ (including $F$ itself).

Then, introduce the formula

$$p_F \land \bigwedge_{G = (G_1 \circ G_2) \in S_F} \text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$$

Claim: This formula is equisatisfiable to $F$.

The proof is by structural induction.

Formula is also in CNF because conjunction of CNF formulas is in CNF.
Tseitin’s Transformation and Size

- Using this transformation, we converted $F$ to an equisatisfiable CNF formula $F'$.

$\exists G \in S F_{\text{CNF}}(p g \leftrightarrow p g_1 \circ p g_2) \in S F_{\text{CNF}}$

- $|S F|$ is bound by the number of connectives in $F$.

- Each formula $S F_{\text{CNF}}(p g \leftrightarrow p g_1 \circ p g_2)$ has constant size.

- Thus, transformation causes only linear increase in formula size.

- More precisely, the size of resulting formula is bound by $30n + 2$ where $n$ is size of original formula.
Tseitin’s Transformation and Size

- Using this transformation, we converted $F$ to an equisatisfiable CNF formula $F'$. 

- What about the size of $F'$?

\[
F' = p_F \land \bigwedge_{G=(G_1 \circ G_2) \in S_F} \text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2})
\]

- Thus, transformation causes only linear increase in formula size.

- More precisely, the size of resulting formula is bound by $30n + 2$ where $n$ is size of original formula.
Tseitin’s Transformation and Size

- Using this transformation, we converted $F$ to an equisatisfiable CNF formula $F'$.  

- What about the size of $F'$?

$$p_F \land \bigwedge_{G=(G_1 \circ G_2) \in S_F} \text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$$

- $|S_F|$ is bound by the number of connectives in $F$.  

Each formula $\text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$ has constant size.  

Thus, transformation causes only linear increase in formula size.  

More precisely, the size of resulting formula is bound by $30n + 2$ where $n$ is size of original formula.$\blacksquare$
Tseitin’s Transformation and Size

- Using this transformation, we converted $F$ to an equisatisfiable CNF formula $F'$.

- What about the size of $F'$?

\[ p_F \land \bigwedge_{G=(G_1 \circ G_2) \in S_F} \text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2}) \]

- $|S_F|$ is bound by the number of connectives in $F$.

- Each formula $\text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$ has constant size.
Tseitin’s Transformation and Size

▶ Using this transformation, we converted $F$ to an equisatisfiable CNF formula $F'$.

▶ What about the size of $F'$?

$$p_F \land \bigwedge_{G=(G_1 \circ G_2) \in S_F} \text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$$

▶ $|S_F|$ is bound by the number of connectives in $F$.

▶ Each formula $\text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$ has constant size.

▶ Thus, transformation causes only linear increase in formula size.
Using this transformation, we converted $F$ to an equisatisfiable CNF formula $F'$.

What about the size of $F'$?

$$p_F \land \bigwedge_{G = (G_1 \circ G_2) \in S_F} \text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$$

$|S_F|$ is bound by the number of connectives in $F$.

Each formula $\text{CNF}(p_g \leftrightarrow p_{g_1} \circ p_{g_2})$ has constant size.

Thus, transformation causes only linear increase in formula size.

More precisely, the size of resulting formula is bound by $30n + 2$ where $n$ is size of original formula.
Tseitin’s Transformation Example

Convert \( F : (p \lor q) \rightarrow (p \land \neg r) \) to equisatisfiable CNF formula.
Tseitin’s Transformation Example

Convert $F : (p \lor q) \rightarrow (p \land \neg r)$ to equisatisfiable CNF formula.

1. For each subformula, introduce new variables: $p_1$ for $F$, $p_2$ for $p \lor q$, $p_3$ for $p \land \neg r$, and $p_4$ for $\neg r$. 
Tseitin’s Transformation Example

Convert $F : (p \lor q) \rightarrow (p \land \neg r)$ to equisatisfiable CNF formula.

1. For each subformula, introduce new variables: $p_1$ for $F$, $p_2$ for $p \lor q$, $p_3$ for $p \land \neg r$, and $p_4$ for $\neg r$.

2. Stipulate equivalences and convert them to CNF:
Tseitin’s Transformation Example

Convert $F : (p \lor q) \rightarrow (p \land \neg r)$ to equisatisfiable CNF formula.

1. For each subformula, introduce new variables: $p_1$ for $F$, $p_2$ for $p \lor q$, $p_3$ for $p \land \neg r$, and $p_4$ for $\neg r$.

2. Stipulate equivalences and convert them to CNF:

   $p_1 \leftrightarrow (p_2 \rightarrow p_3)$
Tseitin’s Transformation Example

Convert $F : (p \lor q) \rightarrow (p \land \neg r)$ to equisatisfiable CNF formula.

1. For each subformula, introduce new variables: $p_1$ for $F$, $p_2$ for $p \lor q$, $p_3$ for $p \land \neg r$, and $p_4$ for $\neg r$.

2. Stipulate equivalences and convert them to CNF:

   $$p_1 \leftrightarrow (p_2 \rightarrow p_3) \quad \Rightarrow \quad F_1 : (\neg p_1 \lor \neg p_2 \lor p_3) \land (p_2 \lor p_1) \land (\neg p_3 \lor p_1)$$
Tseitin’s Transformation Example

Convert \( F : (p \lor q) \rightarrow (p \land \neg r) \) to equisatisfiable CNF formula.

1. For each subformula, introduce new variables: \( p_1 \) for \( F \), \( p_2 \) for \( p \lor q \), \( p_3 \) for \( p \land \neg r \), and \( p_4 \) for \( \neg r \).

2. Stipulate equivalences and convert them to CNF:

   \[
   p_1 \iff (p_2 \rightarrow p_3) \quad \Rightarrow \quad F_1 : (\neg p_1 \lor \neg p_2 \lor p_3) \land (p_2 \lor p_1) \land (\neg p_3 \lor p_1)
   \]

   \[
   p_2 \iff (p \lor q) \quad \Rightarrow \quad F_2 : (\neg p_2 \lor p_2) \land (p_2 \lor p_1) \land (\neg p_3 \lor p_1)
   \]

   The formula \( p_1 \land F_1 \land F_2 \land F_3 \land F_4 \) is equisatisfiable to \( F \) and is in CNF.
Tseitin’s Transformation Example

Convert $F : (p \lor q) \rightarrow (p \land \neg r)$ to equisatisfiable CNF formula.

1. For each subformula, introduce new variables: $p_1$ for $F$, $p_2$ for $p \lor q$, $p_3$ for $p \land \neg r$, and $p_4$ for $\neg r$.

2. Stipulate equivalences and convert them to CNF:

   $p_1 \leftrightarrow (p_2 \rightarrow p_3) \quad \Rightarrow F_1 : (\neg p_1 \lor \neg p_2 \lor p_3) \land (p_2 \lor p_1) \land (\neg p_3 \lor p_1)$

   $p_2 \leftrightarrow (p \lor q) \quad \Rightarrow F_2 : (\neg p_2 \lor p \lor q) \land (\neg p \lor p_2) \land (\neg q \lor p_2)$
Tseitin’s Transformation Example

Convert $F: (p ∨ q) → (p ∧ ¬r)$ to equisatisfiable CNF formula.

1. For each subformula, introduce new variables: $p_1$ for $F$, $p_2$ for $p ∨ q$, $p_3$ for $p ∧ ¬r$, and $p_4$ for $¬r$.

2. Stipulate equivalences and convert them to CNF:

\[
\begin{align*}
p_1 &\leftrightarrow (p_2 → p_3) \quad \Rightarrow F_1 : (¬p_1 ∨ ¬p_2 ∨ p_3) ∧ (p_2 ∨ p_1) ∧ (¬p_3 ∨ p_1) \\
p_2 &\leftrightarrow (p ∨ q) \quad \Rightarrow F_2 : (¬p_2 ∨ p ∨ q) ∧ (¬p ∨ p_2) ∧ (¬q ∨ p_2) \\
p_3 &\leftrightarrow (p ∧ p_4)
\end{align*}
\]
Tseitin’s Transformation Example

Convert $F : (p \lor q) \rightarrow (p \land \neg r)$ to equisatisfiable CNF formula.

1. For each subformula, introduce new variables: $p_1$ for $F$, $p_2$ for $p \lor q$, $p_3$ for $p \land \neg r$, and $p_4$ for $\neg r$.

2. Stipulate equivalences and convert them to CNF:

\[
\begin{align*}
  p_1 & \leftrightarrow (p_2 \rightarrow p_3) \quad \Rightarrow \quad F_1 : (\neg p_1 \lor \neg p_2 \lor p_3) \land (p_2 \lor p_1) \land (\neg p_3 \lor p_1) \\
  p_2 & \leftrightarrow (p \lor q) \quad \Rightarrow \quad F_2 : (\neg p_2 \lor p \lor q) \land (\neg p \lor p_2) \land (\neg q \lor p_2) \\
  p_3 & \leftrightarrow (p \land p_4) \quad \Rightarrow \quad F_3 : (\neg p_3 \lor p) \land (\neg p_3 \lor p_4) \land (\neg p \lor \neg p_4 \lor p_3)
\end{align*}
\]
Tseitin’s Transformation Example

Convert $F : (p \lor q) \rightarrow (p \land \neg r)$ to equisatisfiable CNF formula.

1. For each subformula, introduce new variables: $p_1$ for $F$, $p_2$ for $p \lor q$, $p_3$ for $p \land \neg r$, and $p_4$ for $\neg r$.

2. Stipulate equivalences and convert them to CNF:

\[
\begin{align*}
    & p_1 \leftrightarrow (p_2 \rightarrow p_3) \quad \Rightarrow F_1 : (\neg p_1 \lor \neg p_2 \lor p_3) \lor (p_2 \lor p_1) \lor (\neg p_3 \lor p_1) \\
    & p_2 \leftrightarrow (p \lor q) \quad \Rightarrow F_2 : (\neg p_2 \lor p \lor q) \lor (\neg p \lor p_2) \lor (\neg q \lor p_2) \\
    & p_3 \leftrightarrow (p \land p_4) \quad \Rightarrow F_3 : (\neg p_3 \lor p) \lor (\neg p_3 \lor p_4) \lor (\neg p \lor \neg p_4 \lor p_3) \\
    & p_4 \leftrightarrow \neg r
\end{align*}
\]
Tseitin’s Transformation Example

Convert $F : (p \lor q) \rightarrow (p \land \neg r)$ to equisatisfiable CNF formula.

1. For each subformula, introduce new variables: $p_1$ for $F$, $p_2$ for $p \lor q$, $p_3$ for $p \land \neg r$, and $p_4$ for $\neg r$.

2. Stipulate equivalences and convert them to CNF:

   - $p_1 \leftrightarrow (p_2 \rightarrow p_3) \Rightarrow F_1 : (\neg p_1 \lor \neg p_2 \lor p_3) \land (p_2 \lor p_1) \land (\neg p_3 \lor p_1)$
   - $p_2 \leftrightarrow (p \lor q) \Rightarrow F_2 : (\neg p_2 \lor p \lor q) \land (\neg p \lor p_2) \land (\neg q \lor p_2)$
   - $p_3 \leftrightarrow (p \land p_4) \Rightarrow F_3 : (\neg p_3 \lor p) \land (\neg p_3 \lor p_4) \land (\neg p \lor \neg p_4 \lor p_3)$
   - $p_4 \leftrightarrow \neg r \Rightarrow F_4 : (\neg p_4 \lor \neg r) \land (p_4 \lor r)$
Tseitin’s Transformation Example

Convert \( F : (p \lor q) \rightarrow (p \land \neg r) \) to equisatisfiable CNF formula.

1. For each subformula, introduce new variables: \( p_1 \) for \( F \), \( p_2 \) for \( p \lor q \), \( p_3 \) for \( p \land \neg r \), and \( p_4 \) for \( \neg r \).

2. Stipulate equivalences and convert them to CNF:

\[
\begin{align*}
p_1 & \leftrightarrow (p_2 \rightarrow p_3) \quad \Rightarrow \quad F_1 : (\neg p_1 \lor \neg p_2 \lor p_3) \land (p_2 \lor p_1) \land (\neg p_3 \lor p_1) \\
p_2 & \leftrightarrow (p \lor q) \quad \Rightarrow \quad F_2 : (\neg p_2 \lor p \lor q) \land (\neg p \lor p_2) \land (\neg q \lor p_2) \\
p_3 & \leftrightarrow (p \land p_4) \quad \Rightarrow \quad F_3 : (\neg p_3 \lor p) \land (\neg p_3 \lor p_4) \land (\neg p \lor \neg p_4 \lor p_3) \\
p_4 & \leftrightarrow \neg r \quad \Rightarrow \quad F_4 : (\neg p_4 \lor \neg r) \land (p_4 \lor r)
\end{align*}
\]

3. The formula

\[ p_1 \land F_1 \land F_2 \land F_3 \land F_4 \]

is equisatisfiable to \( F \) and is in CNF.
Almost all SAT solvers today are based on an algorithm called DPLL (Davis-Putnam-Logemann-Loveland).
Almost all SAT solvers today are based on an algorithm called DPLL (Davis-Putnam-Logemann-Loveland)
1962: the original algorithm known as DP (Davis-Putnam) ⇒ “simple” procedure for automated theorem proving

Davis and Putnam hired two programmers, George Logemann and David Loveland, to implement their ideas on the IBM 704. Not all of their ideas worked out as planned ⇒ refined algorithm to what is known today as DPLL.

DPLL: Historical Perspective
DPLL: Historical Perspective

- 1962: the original algorithm known as DP (Davis-Putnam) ⇒ “simple” procedure for automated theorem proving

- Davis and Putnam hired two programmers, George Logemann and David Loveland, to implement their ideas on the IBM 704.
1962: the original algorithm known as DP (Davis-Putnam) ⇒ “simple” procedure for automated theorem proving

Davis and Putnam hired two programmers, George Logemann and David Loveland, to implement their ideas on the IBM 704.
1962: the original algorithm known as DP (Davis-Putnam) ⇒ “simple” procedure for automated theorem proving

Davis and Putnam hired two programmers, George Logemann and David Loveland, to implement their ideas on the IBM 704.

Not all of their ideas worked out as planned ⇒ refined algorithm to what is known today as DPLL
There are two distinct ways to approach the boolean satisfiability problem:

- Search: Find satisfying assignment in by searching through all possible assignments. The most basic incarnation: truth table!

DPLL combines search and deduction in a very effective way!
DPLL insight

- There are two distinct ways to approach the boolean satisfiability problem:

  - **Search**
    - Find satisfying assignment in by searching through all possible assignments
      ⇒ most basic incarnation: truth table!

  - **Deduction**
    - Deduce new facts from set of known facts
      ⇒ application of proof rules, semantic argument method
There are two distinct ways to approach the boolean satisfiability problem:

- **Search**
  - Find satisfying assignment in by searching through all possible assignments
  - ⇒ most basic incarnation: truth table!

- **Deduction**
  - Deduce new facts from set of known facts ⇒ application of proof rules, semantic argument method
DPLL insight

- There are two distinct ways to approach the boolean satisfiability problem:

  - **Search**
    - Find satisfying assignment in by searching through all possible assignments
      \[\Rightarrow\] most basic incarnation: truth table!

  - **Deduction**
    - Deduce new facts from set of known facts
      \[\Rightarrow\] application of proof rules, semantic argument method

- DPLL combines search and deduction in a very effective way!
Deduction in DPLL

- Deductive principle underlying DPLL is *propositional resolution*
Deduction in DPLL

- Deductive principle underlying DPLL is **propositional resolution**

- Resolution can only be applied to formulas in CNF
Deduction in DPLL

- Deductive principle underlying DPLL is propositional resolution
- Resolution can only be applied to formulas in CNF
- SAT solvers convert formulas to CNF to be able to perform resolution
Propositional Resolution

- Consider two clauses in CNF:

\[ C_1 : (l_1 \lor \ldots p \ldots \lor l_k) \quad C_2 : (l'_1 \lor \ldots \neg p \ldots \lor l'_n) \]
Propositional Resolution

- Consider two clauses in CNF:
  \[ C_1 : (l_1 \lor \ldots \lor p \ldots \lor l_k) \quad C_2 : (l_1' \lor \ldots \lor \neg p \ldots \lor l_n') \]

- From these, we can deduce a new clause \( C_3 \), called resolvent:
  \[ C_3 : (l_1 \lor \ldots \lor l_k \lor l_1' \lor \ldots \lor l_n') \]
Propositional Resolution

- Consider two clauses in CNF:

\[ C_1 : \ (l_1 \lor \ldots \lor p \ldots \lor l_k) \quad C_2 : \ (l'_1 \lor \ldots \neg p \ldots \lor l'_n) \]

- From these, we can deduce a new clause \( C_3 \), called resolvent:

\[ C_3 : \ (l_1 \lor \ldots \lor l_k \lor l'_1 \lor \ldots \lor l'_n) \]

- Correctness:

  Suppose \( p \) is assigned \( \top \):
  Since \( C_2 \) must be satisfied and since \( \neg p \) is \( \bot \),
  \( (l'_1 \lor \ldots \lor l'_n) \) must be true.

  Suppose \( p \) is assigned \( \bot \):
  Since \( C_1 \) must be satisfied and since \( p \) is \( \bot \),
  \( (l_1 \lor \ldots \lor l_k) \) must be true.

  Thus, \( C_3 \) must be true.
Propositional Resolution

- Consider two clauses in CNF:

\[ C_1 : (l_1 \lor \ldots \lor l_k) \quad C_2 : (l'_1 \lor \ldots \lor \neg p \lor \ldots \lor l'_n) \]

- From these, we can deduce a new clause \( C_3 \), called resolvent:

\[ C_3 : (l_1 \lor \ldots \lor l_k \lor l'_1 \lor \ldots \lor l'_n) \]

- Correctness:
  - Suppose \( p \) is assigned \( \top \): Since \( C_2 \) must be satisfied and since \( \neg p \) is \( \bot \), \((l'_1 \lor \ldots \lor l'_n)\) must be true.
Propositional Resolution

▶ Consider two clauses in CNF:

\[ C_1 : (l_1 \lor \ldots \lor p \lor \ldots \lor l_k) \quad C_2 : (l'_1 \lor \ldots \lor \neg p \lor \ldots \lor l'_n) \]

▶ From these, we can deduce a new clause \( C_3 \), called resolvent:

\[ C_3 : (l_1 \lor \ldots \lor l_k \lor l'_1 \lor \ldots \lor l'_n) \]

▶ Correctness:

▶ Suppose \( p \) is assigned \( \top \): Since \( C_2 \) must be satisfied and since \( \neg p \) is \( \bot \), \( (l'_1 \lor \ldots \lor l'_n) \) must be true.

▶ Suppose \( p \) is assigned \( \bot \): Since \( C_1 \) must be satisfied and since \( p \) is \( \bot \), \( (l_1 \lor \ldots \lor l_k) \) must be true.
Propositional Resolution

- Consider two clauses in CNF:

\[ C_1 : (l_1 \lor \ldots \lor p \lor \ldots \lor l_k) \quad \quad C_2 : (l'_1 \lor \ldots \lor \neg p \lor \ldots \lor l'_n) \]

- From these, we can deduce a new clause \( C_3 \), called resolvent:

\[ C_3 : (l_1 \lor \ldots \lor l_k \lor l'_1 \lor \ldots \lor l'_n) \]

- Correctness:
  - Suppose \( p \) is assigned \( \top \): Since \( C_2 \) must be satisfied and since \( \neg p \) is \( \bot \), \((l'_1 \lor \ldots \lor l'_n)\) must be true.

  - Suppose \( p \) is assigned \( \bot \): Since \( C_1 \) must be satisfied and since \( p \) is \( \bot \), \((l_1 \lor \ldots \lor l_k)\) must be true.

- Thus, \( C_3 \) must be true.
Unit Resolution

- DPLL uses a restricted form of resolution, known as unit resolution.
Unit Resolution

- DPLL uses a restricted form of resolution, known as unit resolution.
- Unit resolution is propositional resolution, but one of the clauses must be a unit clause (i.e., contains only one literal).
Unit Resolution

- DPLL uses a restricted form of resolution, known as unit resolution.

- Unit resolution is propositional resolution, but one of the clauses must be a unit clause (i.e., contains only one literal)

- \[ C_1 : p \quad C_2 : (l_1 \lor \ldots \lor \neg p \ldots \lor l_n) \]
Unit Resolution

- DPLL uses a restricted form of resolution, known as unit resolution.

- Unit resolution is propositional resolution, but one of the clauses must be a unit clause (i.e., contains only one literal)

- \( C_1 : p \quad C_2 : (l_1 \lor \ldots \neg p \ldots \lor l_n) \)

- Resolvent: \( (l_1 \lor \ldots \lor l_n) \)
Unit Resolution

- DPLL uses a restricted form of resolution, known as unit resolution.

- Unit resolution is propositional resolution, but one of the clauses must be a unit clause (i.e., contains only one literal)

- \( C_1 : p \quad C_2 : \ (l_1 \lor \ldots \lor \neg p \ldots \lor l_n) \)

- Resolvent: \( (l_1 \lor \ldots \lor l_n) \)

- Performing unit resolution on \( C_1 \) and \( C_2 \) is same as replacing \( p \) with true in the original clauses.
Unit Resolution

- DPLL uses a restricted form of resolution, known as unit resolution.

- Unit resolution is propositional resolution, but one of the clauses must be a unit clause (i.e., contains only one literal)

- \( C_1 : p \quad C_2 : (l_1 \lor \ldots \neg p \ldots \lor l_n) \)

- Resolvent: \((l_1 \lor \ldots \lor l_n)\)

- Performing unit resolution on \( C_1 \) and \( C_2 \) is same as replacing \( p \) with true in the original clauses.

- In DPLL, all possible applications of unit resolution called Boolean Constraint Propagation (BCP).
Boolean Constraint Propagation (BCP) Example

Apply BCP to CNF formula:

\[(p) \land (\neg p \lor q) \land (r \lor \neg q \lor s)\]
Boolean Constraint Propagation (BCP) Example

- Apply BCP to CNF formula:

\[(p) \land (\neg p \lor q) \land (r \lor \neg q \lor s)\]

- Resolvent of first and second clause:
Boolean Constraint Propagation (BCP) Example

- Apply BCP to CNF formula:

\[(p) \land (\neg p \lor q) \land (r \lor \neg q \lor s)\]

- Resolvent of first and second clause: \(q\)
Apply BCP to CNF formula:

\((p) \land (\neg p \lor q) \land (r \lor \neg q \lor s)\)

Resolvent of first and second clause: \(q\)

New formula:
Boolean Constraint Propagation (BCP) Example

- Apply BCP to CNF formula:

\[(p) \land (\neg p \lor q) \land (r \lor \neg q \lor s)\]

- Resolvent of first and second clause: \(q\)

- New formula: \(q \land (r \lor \neg q \lor s)\)
Boolean Constraint Propagation (BCP) Example

- Apply BCP to CNF formula:

\[ (p) \land (\neg p \lor q) \land (r \lor \neg q \lor s) \]

- Resolvent of first and second clause: \( q \)

- New formula: \( q \land (r \lor \neg q \lor s) \)

- Apply unit resolution again:
Boolean Constraint Propagation (BCP) Example

- Apply BCP to CNF formula:

\[(p) \land (\neg p \lor q) \land (r \lor \neg q \lor s)\]

- Resolvent of first and second clause: \(q\)

- New formula: \(q \land (r \lor \neg q \lor s)\)

- Apply unit resolution again: \((r \lor s)\)
Apply BCP to CNF formula:

\[(p) \land (\neg p \lor q) \land (r \lor \neg q \lor s)\]

Resolvent of first and second clause: \(q\)

New formula: \(q \land (r \lor \neg q \lor s)\)

Apply unit resolution again: \((r \lor s)\)

No more unit resolution possible, so this is the result of BCP.
Basic DPLL

bool DPLL($\phi$)
{
}

- Recursive procedure; input is formula in CNF
Basic DPLL

```
bool DPLL(φ)
{
  1. φ' = BCP(φ)
}
```

- Recursive procedure; input is formula in CNF
Basic DPLL

```c
bool DPLL(\phi)
{
    1. \phi' = BCP(\phi)
    2. if(\phi' = \top) then return SAT;

}
```

- Recursive procedure; input is formula in CNF
- Formula is \top if no more clauses left
Basic DPLL

bool DPLL(φ)
{
    1. φ’ = BCP(φ)
    2. if(φ’ = ⊤) then return SAT;
    3. else if(φ’ = ⊥) then return UNSAT;
}

- Recursive procedure; input is formula in CNF
- Formula is ⊤ if no more clauses left
- Formula becomes ⊥ if we derive ⊥ due to unit resolution
Basic DPLL

```cpp
bool DPLL(\phi)
{
    1. \phi' = BCP(\phi)
    2. if(\phi' = \top) then return SAT;
    3. else if(\phi' = \bot) then return UNSAT;
    4. p = choose_var(\phi');
}
```

- Recursive procedure; input is formula in CNF
- Formula is \top if no more clauses left
- Formula becomes \bot if we derive \bot due to unit resolution
Basic DPLL

bool DPLL(\phi)
{
  1. \phi' = BCP(\phi)
  2. if(\phi' = \top) then return SAT;
  3. else if(\phi' = \bot) then return UNSAT;
  4. p = choose_var(\phi');
  5. if(DPLL(\phi'[p \mapsto \top])) then return SAT;
}

- Recursive procedure; input is formula in CNF
- Formula is \top if no more clauses left
- Formula becomes \bot if we derive \bot due to unit resolution
**Basic DPLL**

```cpp
bool DPLL(\(\phi\))
{
1. \(\phi' = BCP(\phi)\)
2. if(\(\phi' = \top\)) then return SAT;
3. else if(\(\phi' = \bot\)) then return UNSAT;
4. \(p = \text{choose}_{\text{var}}(\phi')\);
5. if(DPLL(\(\phi'[p \mapsto \top]\))) then return SAT;
6. else return (DPLL(\(\phi'[p \mapsto \bot]\)));
}
```

- Recursive procedure; input is formula in CNF
- Formula is \(\top\) if no more clauses left
- Formula becomes \(\bot\) if we derive \(\bot\) due to unit resolution
An Optimization: Pure Literal Propagation

- If variable $p$ occurs only positively in the formula (i.e., no $\neg p$), $p$ must be set to $\top$.

Similarly, if $p$ occurs only negatively (i.e., only appears as $\neg p$), $p$ must be set to $\bot$.

This is known as Pure Literal Propagation (PLP).
An Optimization: Pure Literal Propagation

- If variable $p$ occurs only positively in the formula (i.e., no $\neg p$), $p$ must be set to $\top$

- Similarly, if $p$ occurs only negatively (i.e., only appears as $\neg p$), $p$ must be set to $\bot$
An Optimization: Pure Literal Propagation

- If variable $p$ occurs only positively in the formula (i.e., no $\neg p$), $p$ must be set to $\top$.

- Similarly, if $p$ occurs only negatively (i.e., only appears as $\neg p$), $p$ must be set to $\bot$.

- This is known as Pure Literal Propagation (PLP).
DPLL with Pure Literal Propagation

```c
bool DPLL(\phi)
{
1. \phi' = BCP(\phi)
2. \phi'' = PLP(\phi')
3. if(\phi'' = \top) then return SAT;
4. else if(\phi'' = \bot) then return UNSAT;
5. p = choose_var(\phi'');
6. if(DPLL(\phi''[p \mapsto \top])) then return SAT;
7. else return (DPLL(\phi''[p \mapsto \bot]));
}
```
Example

\[ F : (\neg p \lor q \lor r) \land (\neg q \lor r) \land (\neg q \lor \neg r) \land (p \lor \neg q \lor \neg r) \]
Example

\[ F : (\neg p \lor q \lor r) \land (\neg q \lor r) \land (\neg q \lor \neg r) \land (p \lor \neg q \lor \neg r) \]

- No BCP possible because no unit clause
Example

\[ F : (\neg p \lor q \lor r) \land (\neg q \lor r) \land (\neg q \lor \neg r) \land (p \lor \neg q \lor \neg r) \]

- No BCP possible because no unit clause

- No PLP possible because there are no pure literals
Example

\[ F : (¬p \lor q \lor r) \land (¬q \lor r) \land (¬q \lor ¬r) \land (p \lor ¬q \lor ¬r) \]

- No BCP possible because no unit clause

- No PLP possible because there are no pure literals

- Choose variable \( q \) to branch on:

\[ F[q \mapsto \top] : (r) \land (¬r) \land (p \lor ¬r) \]
Example

\[ F : (\neg p \lor q \lor r) \land (\neg q \lor r) \land (\neg q \lor \neg r) \land (p \lor \neg q \lor \neg r) \]

- No BCP possible because no unit clause

- No PLP possible because there are no pure literals

- Choose variable \( q \) to branch on:

  \[ F[q \mapsto \top] : (r) \land (\neg r) \land (p \lor \neg r) \]

- Unit resolution using \( r \) and \( \neg r \) deduces \( \bot \Rightarrow \) backtrack
Example Cont.

\[ F : (\neg p \lor q \lor r) \land (\neg q \lor r) \land (\neg q \lor \neg r) \land (p \lor \neg q \lor \neg r) \]

- Now, try \( q = \bot \)

\[ F[q \mapsto \bot] : (\neg p \lor r) \]

▶ By PLP, set \( p \) to \( \bot \) and \( r \) to \( \top \)

▶ \( F[q \mapsto \bot, p \mapsto \bot, r \mapsto \top] : \top \)

▶ Thus, \( F \) is satisfiable and the assignment \( [q \mapsto \bot, p \mapsto \bot, r \mapsto \top] \) is a model (i.e., a satisfying interpretation) of \( F \).
Example Cont.

\[ F : (\neg p \lor q \lor r) \land (\neg q \lor r) \land (\neg q \lor \neg r) \land (p \lor \neg q \lor \neg r) \]

- Now, try \( q = \bot \)
  \[
  F[q \mapsto \bot] : (\neg p \lor r)
  \]

- By PLP, set \( p \) to \( \bot \) and \( r \) to \( \top \)
Example Cont.

\[ F : (\neg p \lor q \lor r) \land (\neg q \lor r) \land (\neg q \lor \neg r) \land (p \lor \neg q \lor \neg r) \]

- Now, try \( q = \bot \)
  \[ F[q \mapsto \bot] : (\neg p \lor r) \]

- By PLP, set \( p \) to \( \bot \) and \( r \) to \( \top \)

- \( F[q \mapsto \bot, p \mapsto \bot, r \mapsto \top] : \top \)
Example Cont.

\[
F : (\neg p \lor q \lor r) \land (\neg q \lor r) \land (\neg q \lor \neg r) \land (p \lor \neg q \lor \neg r)
\]

- Now, try \( q = \bot \)
  
  \[
  F[q \mapsto \bot] : (\neg p \lor r)
  \]

- By PLP, set \( p \) to \( \bot \) and \( r \) to \( \top \)

\[
F[q \mapsto \bot, p \mapsto \bot, r \mapsto \top] : \top
\]

- Thus, \( F \) is satisfiable and the assignment \([q \mapsto \bot, p \mapsto \bot, r \mapsto \top]\) is a model (i.e., a satisfying interpretation) of \( F \).
Summar

- Normals forms: NNF, DNF, CNF (will come up again)
Summary

- Normals forms: NNF, DNF, CNF (will come up again)

- For every formula, there exists an equivalent formula in normal form
Summary

- Normals forms: NNF, DNF, CNF (will come up again)
- For every formula, there exists an equivalent formula in normal form
- But equivalence-preserving transformation to DNF and CNF causes exponential blowup
Summary

- Normals forms: NNF, DNF, CNF (will come up again)

- For every formula, there exists an equivalent formula in normal form

- But equivalence-preserving transformation to DNF and CNF causes exponential blowup

- However, Tseitin’s transformation gives an equisatisfiable formula in CNF with only linear increase in size
Summary

- Normals forms: NNF, DNF, CNF (will come up again)

- For every formula, there exists an equivalent formula in normal form

- But equivalence-preserving transformation to DNF and CNF causes exponential blowup

- However, Tseitin’s transformation gives an equisatisfiable formula in CNF with only linear increase in size

- Almost all SAT solvers work on CNF formulas to perform BCP
Summary

- Normals forms: NNF, DNF, CNF (will come up again)
- For every formula, there exists an equivalent formula in normal form
- But equivalence-preserving transformation to DNF and CNF causes exponential blowup
- However, Tseitin’s transformation gives an equisatisfiable formula in CNF with only linear increase in size
- Almost all SAT solvers work on CNF formulas to perform BCP
- DPLL basis of most state-of-the-art SAT solvers
Substantial improvements over basic DPLL used by modern SAT solvers: non-chronological backtracking and learning
Substantial improvements over basic DPLL used by modern SAT solvers: non-chronological backtracking and learning

Implementation tricks used to perform BCP very efficiently
Next Lecture

- Substantial improvements over basic DPLL used by modern SAT solvers: non-chronological backtracking and learning
- Implementation tricks used to perform BCP very efficiently
- Useful heuristics for choosing variable to branch on