1. Recap: Syntax and Semantics of Propositional Logic
2. Questions
3. Conjunctive Normal Form
4. Algorithms for Satisfiability
Recap: Syntax and Semantics of Propositional Logic

1. Recap: Syntax and Semantics of Propositional Logic
   - Propositional Atoms
   - Syntax of Propositional Logic
   - Evaluation of Formulas

2. Questions

3. Conjunctive Normal Form

4. Algorithms for Satisfiability
Atoms

Convention

We usually use $p$, $q$, $p_1$, etc, instead of sentences like “The sun is shining today”.
Atoms

Convention
We usually use $p$, $q$, $p_1$, etc, instead of sentences like “The sun is shining today”.

Atoms
More formally, we fix a set $A$ of propositional atoms.
Models assign truth values

A *model* assigns truth values ($F$ or $T$) to each atom.
Meaning of Atoms

**Models assign truth values**

A *model* assigns truth values (*F* or *T*) to each atom.

**More formally**

A model (valuation) for a propositional logic for the set *A* of atoms is a mapping from *A* to \{ *T*, *F* \}. 
Definition

For a given set $A$ of propositional atoms, the set of *well-formed formulas in propositional logic* is the least set $F$ that fulfills the following rules:

- The constant symbols $\bot$ and $\top$ are in $F$.
- Every element of $A$ is in $F$.
- If $\phi$ is in $F$, then $(\neg \phi)$ is also in $F$.
- If $\phi$ and $\psi$ are in $F$, then $(\phi \land \psi)$ is also in $F$.
- If $\phi$ and $\psi$ are in $F$, then $(\phi \lor \psi)$ is also in $F$.
- If $\phi$ and $\psi$ are in $F$, then $(\phi \rightarrow \psi)$ is also in $F$. 
A formula:

\(((\neg p) \land q) \rightarrow (p \land (q \lor (\neg r))))\)
A formula

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

...and its parse tree:
A formula

$(((\neg p) \land q) \rightarrow (p \land (q \lor (\neg r))))$  

...and its parse tree:
Definition

The result of evaluating a well-formed propositional formula $\phi$ with respect to a valuation $v$, denoted $v(\phi)$ is defined as follows:

- If $\phi$ is the constant $\bot$, then $v(\phi) = F$.
- If $\phi$ is the constant $\top$, then $v(\phi) = T$.
- If $\phi$ is an propositional atom $p$, then $v(\phi) = p^v$.
- If $\phi$ has the form $(\neg \psi)$, then $v(\phi) = \neg v(\psi)$.
- If $\phi$ has the form $(\psi \land \tau)$, then $v(\phi) = v(\psi) \land v(\tau)$.
- If $\phi$ has the form $(\psi \lor \tau)$, then $v(\phi) = v(\psi) \lor v(\tau)$.
- If $\phi$ has the form $(\psi \rightarrow \tau)$, then $v(\phi) = v(\psi) \rightarrow v(\tau)$.
Valid and Satisfiable Formulas

Definition

A formula is called valid if it evaluates to $T$ with respect to every possible valuation.
Valid and Satisfiable Formulas

Definition
A formula is called *valid* if it evaluates to $T$ with respect to every possible valuation.

Definition
A formula is called *satisfiable* if it evaluates to $T$ with respect to at least one valuation.
Recap: Syntax and Semantics of Propositional Logic

Questions

Conjunctive Normal Form

Algorithms for Satisfiability
Questions about Propositional Formula

- Is a given formula valid?
- Is a given formula satisfiable?
- Is a given formula invalid?
- Is a given formula unsatisfiable?
- Are two formulas equivalent?
A decision problem is a question in some formal system with a yes-or-no answer.
Definition

A *decision problem* is a question in some formal system with a yes-or-no answer.

Examples

The question whether a given propositional formula is satisfiable (unsatisfiable, valid, invalid) is a decision problem.

The question whether two given propositional formulas are equivalent is also a decision problem.
How do you decide whether a given propositional formula is satisfiable/valid?
Question

How do you decide whether a given propositional formula is satisfiable/valid?

The good news

We can construct a truth table for the formula and check if some/all rows have $T$ in the last column.
Satisfiability is Decidable

An algorithm for satisfiability

Using a truth table, we can implement an algorithm that returns “yes” if the formula is satisfiable, and that returns “no” if the formula is unsatisfiable.
Satisfiability is Decidable

An algorithm for satisfiability

Using a truth table, we can implement an algorithm that returns “yes” if the formula is satisfiable, and that returns “no” if the formula is unsatisfiable.

Decidability

Decision problems for which there is an algorithm computing “yes” whenever the answer is “yes”, and “no” whenever the answer is “no”, are called decidable.
Satisfiability is Decidable

An algorithm for satisfiability

Using a truth table, we can implement an *algorithm* that returns “yes” if the formula is satisfiable, and that returns “no” if the formula is unsatisfiable.

Decidability

Decision problems for which there is an algorithm computing “yes” whenever the answer is “yes”, and “no” whenever the answer is “no”, are called *decidable*.

Decidability of satisfiability

The question, whether a given propositional formula is satisfiable, is decidable.
The Bad News

Concern

In practice, propositional formulas can be large. Example:
http://www.comp.nus.edu.sg/~cs3234/prop.txt
The Bad News

Concern
In practice, propositional formulas can be large. Example:
http://www.comp.nus.edu.sg/~cs3234/prop.txt

Techniques so far inadequate
Proving satisfiability/validity using truth tables or natural deduction is impractical for large formulas.
Is there a practical way of deciding satisfiability?

Question

Is there an efficient algorithm that decides whether a given formula is satisfiable?

Answer

We do not know!
Is there a practical way of deciding satisfiability?

**Question**

Is there an efficient algorithm that decides whether a given formula is satisfiable?

**More precisely...**

Is there a polynomial-time algorithm that decides whether a given formula is satisfiable?
Is there a practical way of deciding satisfiability?

Question
Is there an efficient algorithm that decides whether a given formula is satisfiable?

More precisely...
Is there a polynomial-time algorithm that decides whether a given formula is satisfiable?

Answer
We do not know!
What *do* we know about satisfiability?

**Truth assignment as witness**

If the answer is “yes”, then a satisfying truth assignment can serve as a proof that the answer is indeed “yes”.

If the answer is “yes”, then a satisfying truth assignment can serve as a proof that the answer is indeed “yes”.

Witness for satisfiability

Such a proof is called a witness.

Checking the witness

We can quickly check whether indeed the witness assignment makes the formula true. This can be done in time proportional to the size of the formula.
What do we know about satisfiability?

Truth assignment as witness
If the answer is “yes”, then a satisfying truth assignment can serve as a proof that the answer is indeed “yes”.

Witness for satisfiability
Such a proof is called a witness.
What do we know about satisfiability?

Truth assignment as witness
If the answer is “yes”, then a satisfying truth assignment can serve as a proof that the answer is indeed “yes”.

Witness for satisfiability
Such a proof is called a witness.

Checking the witness
We can quickly check whether indeed the witness assignment makes the formula true. This can be done in time proportional to the size of the formula.
Definition

Decision problems for which the “yes” answer has a proof that can be checked in polynomial time, are called $NP$. 

Origin of name

NP stands for “Non-deterministic Polynomial time”. 

Original definition

NP is the set of decision problems solvable in polynomial time by a non-deterministic Turing machine.
The Complexity Class NP

**Definition**
Decision problems for which the “yes” answer has a proof that can be checked in polynomial time, are called *NP*.

**Origin of name**
NP stands for “*Non*-deterministic *Polynomial* time”. 
The Complexity Class NP

Definition
Decision problems for which the “yes” answer has a proof that can be checked in polynomial time, are called *NP*.

Origin of name
NP stands for “Non-deterministic Polynomial time”.

Original definition
NP is the set of decision problems solvable in polynomial time by a non-deterministic Turing machine.
The class NP was introduced by Stephen Cook in 1971 at the 3rd Annual ACM Symposium on Theory of Computing.
Some History

- The class NP was introduced by Stephen Cook in 1971 at the 3rd Annual ACM Symposium on Theory of Computing.
- At the conference, there was a fierce debate whether there could be a polynomial time algorithm to solve such problems.
Some History

- The class NP was introduced by Stephen Cook in 1971 at the 3rd Annual ACM Symposium on Theory of Computing.
- At the conference, there was a fierce debate whether there could be a polynomial time algorithm to solve such problems.
- John Hopcroft convinced the delegates that the problem should be put off to be solved at some later date.
Some History

- The class NP was introduced by Stephen Cook in 1971 at the 3rd Annual ACM Symposium on Theory of Computing.
- At the conference, there was a fierce debate whether there could be a polynomial time algorithm to solve such problems.
- John Hopcroft convinced the delegates that the problem should be put off to be solved at some later date.
- In 1972, Richard Karp presented 21 mutually equivalent problems in NP, for which no polynomial time algorithms was known.
The class NP was introduced by Stephen Cook in 1971 at the 3rd Annual ACM Symposium on Theory of Computing. At the conference, there was a fierce debate whether there could be a polynomial time algorithm to solve such problems. John Hopcroft convinced the delegates that the problem should be put off to be solved at some later date. In 1972, Richard Karp presented 21 mutually equivalent problems in NP, for which no polynomial time algorithms was known. Cook and Leonid Levin proved independently that propositional satisifiability is in this class (called NP-complete).
Clearly $P \subseteq NP$
Clearly $P \subseteq NP$. Why?
P = NP?

- Clearly $P \subseteq NP$. Why?
- But does $NP \subseteq P$ hold?
Clearly $P \subseteq NP$. Why?

But does $NP \subseteq P$ hold?

To date, no proof of $P = NP$ or $P \neq NP$ has been discovered.
Clearly $P \subseteq NP$. Why?

But does $NP \subseteq P$ hold?

To date, no proof of $P = NP$ or $P \neq NP$ has been discovered.

Many computer scientists assume $P \neq NP$, and therefore consider NP-complete problems as “intractable”.
P = NP?

- Clearly $P \subseteq NP$. Why?
- But does $NP \subseteq P$ hold?
- To date, no proof of $P = NP$ or $P \neq NP$ has been discovered.
- Many computer scientists assume $P \neq NP$, and therefore consider NP-complete problems as “intractable”.
- Many “proofs” for one or the other answer have been proposed, and subsequently rejected, most recently by Vinay Deolalikar (a researcher at HP), in August 2010.
1 Recap: Syntax and Semantics of Propositional Logic

2 Questions

3 Conjunctive Normal Form

4 Algorithms for Satisfiability
Conjunctive Normal Form

Definition

A literal $L$ is either an atom $p$ or the negation of an atom $\neg p$. A formula $C$ is in *conjunctive normal form* (CNF) if it is a conjunction of clauses, where each clause is a disjunction of literals:

\[
L ::= p | \neg p \\
D ::= L | L \lor D \\
C ::= D | D \land C
\]
Recap: Syntax and Semantics of Propositional Logic

Questions

Conjunctive Normal Form

Algorithms for Satisfiability

Examples

\((\neg p \lor q \lor r) \land (\neg q \lor r) \land (\neg r)\) is in CNF.

\((\neg p \lor q \lor r) \land ((p \land \neg q) \lor r) \land (\neg r)\) is not in CNF.

\((\neg p \lor q \lor r) \land \neg (\neg q \lor r) \land (\neg r)\) is not in CNF.
Usefulness of CNF

Lemma

A disjunction of literals $L_1 \lor L_2 \lor \cdots \lor L_m$ is valid iff there are $1 \leq i, j \leq m$ such that $L_i$ is $\neg L_j$. 
Usefulness of CNF

Lemma

A disjunction of literals \( L_1 \lor L_2 \lor \cdots \lor L_m \) is valid iff there are \( 1 \leq i, j \leq m \) such that \( L_i \) is \( \neg L_j \).

How to disprove

\[
\models (\neg q \lor p \lor q) \land (\neg p \lor r) \land q
\]
A disjunction of literals \( L_1 \lor L_2 \lor \cdots \lor L_m \) is valid iff there are \( 1 \leq i, j \leq m \) such that \( L_i \) is \( \neg L_j \).

How to disprove

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

Use lemma to disprove any of:

\[ \models (\neg q \lor p \lor r) \quad \models (\neg p \lor r) \quad \models q \]
Usefulness of CNF

Lemma

A disjunction of literals $L_1 \lor L_2 \lor \cdots \lor L_m$ is valid iff there are $1 \leq i, j \leq m$ such that $L_i$ is $\neg L_j$.

How to prove

$$\models (\neg q \lor p \lor q) \land (p \lor r \neg p) \land (r \lor \neg r)$$
A disjunction of literals $L_1 \lor L_2 \lor \cdots \lor L_m$ is valid iff there are $1 \leq i, j \leq m$ such that $L_i$ is $\neg L_j$.

How to prove

$$\models (\neg q \lor p \lor q) \land (p \lor r \neg p) \land (r \lor \neg r)$$

Use lemma to prove all of:

$$\models (\neg q \lor p \lor q) \quad \models (p \lor r \neg p) \quad \models (r \lor \neg r)$$
Proposition

Let $\phi$ be a formula of propositional logic. Then $\phi$ is satisfiable iff $\neg\phi$ is not valid.
Usefulness of CNF

Proposition
Let \( \phi \) be a formula of propositional logic. Then \( \phi \) is satisfiable iff \( \neg \phi \) is not valid.

Satisfiability test
We can test satisfiability of \( \phi \) by transforming \( \neg \phi \) into CNF, and show that some clause is not valid.
Theorem

Every formula in the propositional calculus can be transformed into an equivalent formula in CNF.
Algorithm for CNF Transformation

1. Eliminate implication using:
   \[ A \rightarrow B \equiv \neg A \lor B \]

2. Push all negations inward using De Morgan’s laws:
   \[ \neg(A \land B) \equiv (\neg A \lor \neg B) \]
   \[ \neg(A \lor B) \equiv (\neg A \land \neg B) \]

3. Eliminate double negations using the equivalence \( \neg
   \neg A \equiv A \)

4. The formula now consists of disjunctions and conjunctions of literals. Use the distributive laws
   \[ A \lor (B \land C) \equiv (A \lor B) \land (A \lor C) \]
   \[ (A \land B) \lor C \equiv (A \lor C) \land (B \lor C) \]
   to eliminate conjunctions within disjunctions.
Example

\[(\neg p \rightarrow \neg q) \rightarrow (p \rightarrow q) \equiv \neg (\neg \neg p \lor \neg q) \lor (\neg p \lor q)\]

\[\equiv (\neg \neg \neg p \land q) \lor (\neg p \lor q)\]

\[\equiv (\neg p \land q) \lor (\neg p \lor q)\]

\[\equiv (\neg p \lor \neg p \lor q) \land (q \lor \neg p \lor q)\]
Algorithms for Proving Satisfiability of $\psi$

- Transform $\neg \psi$ into Conjunctive Normal Form $ncnf$ and prove validity (non-validity) of $ncnf$
Algorithms for Proving Satisfiability of $\psi$

- Transform $\neg \psi$ into Conjunctive Normal Form $ncnf$ and prove validity (non-validity) of $ncnf$
- Transform $\psi$ into Conjunctive Normal Form $cnf$ and search for a satisfying valuation
Algorithms for Proving Satisfiability of $\psi$

- Transform $\neg \psi$ into Conjunctive Normal Form $ncnf$ and prove validity (non-validity) of $ncnf$
- Transform $\psi$ into Conjunctive Normal Form $cnf$ and search for a satisfying valuation
  - Complete algorithms: guaranteed to terminate with correct answer
Algorithms for Proving Satisfiability of $\psi$

- Transform $\neg\psi$ into Conjunctive Normal Form $ncnf$ and prove validity (non-validity) of $ncnf$
- Transform $\psi$ into Conjunctive Normal Form $cnf$ and search for a satisfying valuation
  - Complete algorithms: guaranteed to terminate with correct answer
    - example: DPLL
- Incomplete algorithms: Return "yes" for some satisfiable formulas, and run forever for other satisfiable formulas and all unsatisfiable formulas;
  - example: WalkSAT
  - Transform $\psi$ into DAG; return "yes" for some satisfiable formulas, return "no" for some unsatisfiable formulas, return "don't know" otherwise;
    - example: propagation-based linear solver
Transform \( \neg \psi \) into Conjunctive Normal Form \( ncnf \) and prove validity (non-validity) of \( ncnf \)

Transform \( \psi \) into Conjunctive Normal Form \( cnf \) and search for a satisfying valuation

- Complete algorithms: guaranteed to terminate with correct answer
  - example: DPLL
- Incomplete algorithms: Return “yes” for some satisfiable formulas, and run forever for other satisfiable formulas and all unsatisfiable formulas;
Algorithms for Proving Satisfiability of $\psi$

- Transform $\neg \psi$ into Conjunctive Normal Form $ncnf$ and prove validity (non-validity) of $ncnf$
- Transform $\psi$ into Conjunctive Normal Form $cnf$ and search for a satisfying valuation
  - Complete algorithms: guaranteed to terminate with correct answer
    - example: DPLL
  - Incomplete algorithms: Return “yes” for some satisfiable formulas, and run forever for other satisfiable formulas and all unsatisfiable formulas; example: WalkSAT
Algorithms for Proving Satisfiability of $\psi$

- Transform $\neg\psi$ into Conjunctive Normal Form $ncnf$ and prove validity (non-validity) of $ncnf$
- Transform $\psi$ into Conjunctive Normal Form $cnf$ and search for a satisfying valuation
  - Complete algorithms: guaranteed to terminate with correct answer
    - example: DPLL
  - Incomplete algorithms: Return “yes” for some satisfiable formulas, and run forever for other satisfiable formulas and all unsatisfiable formulas; example: WalkSAT
- Transform $\psi$ into DAG; return “yes” for some satisfiable formulas, return “no” for some unsatisfiable formulas, return “don’t know” otherwise;
Transform $\neg \psi$ into Conjunctive Normal Form $ncnf$ and prove validity (non-validity) of $ncnf$

Transform $\psi$ into Conjunctive Normal Form $cnf$ and search for a satisfying valuation
- Complete algorithms: guaranteed to terminate with correct answer
  - example: DPLL
- Incomplete algorithms: Return “yes” for some satisfiable formulas, and run forever for other satisfiable formulas and all unsatisfiable formulas; example: WalkSAT

Transform $\psi$ into DAG; return “yes” for some satisfiable formulas, return “no” for some unsatisfiable formulas, return “don’t know” otherwise; example: propagation-based linear solver