04a—Propositional Logic II

CS 3234: Logic and Formal Systems

Martin Henz

September 2, 2010

Generated on Thursday 2nd September, 2010, 11:40

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.

Meaning of 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 $\{T, F\}$.

Inductive Definition

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 ϕ is in F, then $(\neg\phi)$ is also in F.
- If ϕ and ψ are in F, then $(\phi \wedge \psi)$ is also in F.
- If ϕ and ψ are in F, then $(\phi \vee \psi)$ is also in F.
- If ϕ and ψ are in F, then $(\phi \rightarrow \psi)$ is also in F.

Parse trees

A formula

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

Parse trees

A formula

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

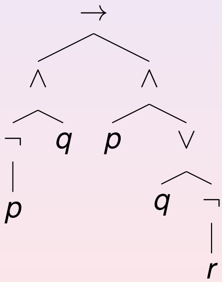
...and its parse tree:

Parse trees

A formula

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

...and its parse tree:



Evaluation of Formulas

Definition

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

- If ϕ is the constant $\bot$, then $v(\phi) = F$.
- If ϕ is the constant $\top$, then $v(\phi) = T$.
- If ϕ is an propositional atom p, then $v(\phi) = p^v$.
- If ϕ has the form $(\neg\psi)$, then $v(\phi) = \backslash v(\psi)$.
- If ϕ has the form $(\psi \land \tau)$, then $v(\phi) = v(\psi) \& v(\tau)$.
- If ϕ has the form $(\psi \lor \tau)$, then $v(\phi) = v(\psi) \mid v(\tau)$.
- If ϕ has the form $(\psi \to \tau)$, 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.

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?

Decision Problems

Definition

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

Decision Problems

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 satisifiable (unsatisfiable, valid, invalid) is a decision problem.

The question whether two given propositional formulas are equivalent is also a decision problem.

How to Solve the Decision Problem?

Question

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

How to Solve the Decision Problem?

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.

Satisifiability is Decidable

An algorithm for satisifiability

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

Satisifiability is Decidable

An algorithm for satisifiability

Using a truth table, we can implement an *algorithm* that returns *"yes"* if the formula is satisifiable, 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*.

Satisifiability is Decidable

An algorithm for satisifiability

Using a truth table, we can implement an *algorithm* that returns *"yes"* if the formula is satisifiable, 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 satisifiable, 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?

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”.

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.

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*.

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 “**N**on-deterministic **P**olynomial 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 “**N**on-deterministic **P**olynomial time”.

Original definition

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

Some History

- 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.

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.
- Cook and Leonid Levin proved independently that propositional satisifiability is in this class (called NP-complete).

P = NP?

- Clearly $P \subseteq NP$

P = NP?

- Clearly $P \subseteq NP$. Why?

P = NP?

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

P = NP?

- Clearly $P \subseteq NP$. Why?
- But does $NP \subseteq P$ hold?
- To date, no proof of $P = \mathrm{NP}$ or $P \neq \mathrm{NP}$ has been discovered.

P = NP?

- Clearly $P \subseteq NP$. Why?
- But does $NP \subseteq P$ hold?
- To date, no proof of $P = \mathrm{NP}$ or $P \neq \mathrm{NP}$ has been discovered.
- Many computer scientists assume $P \neq \mathrm{NP}$, 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 $P = \mathrm{NP}$ or $P \neq \mathrm{NP}$ has been discovered.
- Many computer scientists assume $P \neq \mathrm{NP}$, 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.

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:

$$\begin{aligned} L & ::= & p \mid \neg p \\ D & ::= & L \mid L \vee D \\ C & ::= & D \mid D \wedge C \end{aligned}$$

Examples

$(\neg p \vee q \vee r) \wedge (\neg q \vee r) \wedge (\neg r)$ is in CNF.
$(\neg p \vee q \vee r) \wedge ((p \wedge \neg q) \vee r) \wedge (\neg r)$ is not in CNF.
$(\neg p \vee q \vee r) \wedge \neg(\neg q \vee r) \wedge (\neg r)$ 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 $L_1 \vee L_2 \vee \cdots \vee L_m$ is valid iff there are $1 \leq i, j \leq m$ such that L_i is $\neg L_j$.

How to disprove

$$\models (\neg q \vee p \vee q) \wedge (\neg p \vee r) \wedge q$$

Usefulness of CNF

Lemma

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

How to disprove

$$\models (\neg q \vee p \vee q) \wedge (\neg p \vee r) \wedge q$$

Use lemma to disprove any of:

$$\models (\neg q \vee p \vee r) \qquad \models (\neg p \vee r) \qquad \models q$$

Usefulness of CNF

Lemma

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

How to prove

$$\models (\neg q \vee p \vee q) \wedge (p \vee r \neg p) \wedge (r \vee \neg r)$$

Usefulness of CNF

Lemma

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

How to prove

$$\models (\neg q \vee p \vee q) \wedge (p \vee r \neg p) \wedge (r \vee \neg r)$$

Use lemma to prove all of:

$$\models (\neg q \vee p \vee q) \qquad \models (p \vee r \neg p) \qquad \models (r \vee \neg r)$$

Usefulness of CNF

Proposition

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

Usefulness of CNF

Proposition

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

Satisfiability test

We can test satisfiability of ϕ by transforming $\neg\phi$ into CNF, and show that some clause is not valid.

Transformation to CNF

Theorem

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

Algorithm for CNF Transformation

1. Eliminate implication using:
   $A \to B \equiv \neg A \lor B$
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

$$
\begin{aligned}
(\neg p \rightarrow \neg q) \rightarrow (p \rightarrow q) & \equiv \neg(\neg\neg p \vee \neg q) \vee (\neg p \vee q) \\
& \equiv (\neg\neg\neg p \wedge q) \vee (\neg p \vee q) \\
& \equiv (\neg p \wedge q) \vee (\neg p \vee q) \\
& \equiv (\neg p \vee \neg p \vee q) \wedge (q \vee \neg p \vee q)
\end{aligned}
$$

Algorithms for Proving Satisfiability of ψ

- Transform $\neg\psi$ into Conjunctive Normal Form *ncnf* and prove validity (non-validity) of *ncnf*

Algorithms for Proving Satisfiability of ψ

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

Algorithms for Proving Satisfiability of ψ

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

Algorithms for Proving Satisfiability of ψ

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

Algorithms for Proving Satisfiability of ψ

- Transform $\neg\psi$ into Conjunctive Normal Form *ncnf* and prove validity (non-validity) of *ncnf*
- Transform ψ 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 ψ

- Transform $\neg\psi$ into Conjunctive Normal Form *ncnf* and prove validity (non-validity) of *ncnf*
- Transform ψ 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 ψ

- Transform $\neg\psi$ into Conjunctive Normal Form *ncnf* and prove validity (non-validity) of *ncnf*
- Transform ψ 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 ψ into DAG; return "yes" for some satisfiable formulas, return "no" for some unsatisfiable formulas, return "don't know" otherwise;

Algorithms for Proving Satisfiability of ψ

- Transform $\neg\psi$ into Conjunctive Normal Form *ncnf* and prove validity (non-validity) of *ncnf*
- Transform ψ 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 ψ into DAG; return "yes" for some satisfiable formulas, return "no" for some unsatisfiable formulas, return "don't know" otherwise; example: propagation-based linear solver