# Propositional Logic – I

Madhavan Mukund

Chennai Mathematical Institute
http://www.cmi.ac.in/~madhavan

SAT-SMT School, TIFR
4 December 2016

# What is logic about?

- Structure of logical arguments

  *All men are mortal.*
  *Socrates is a man.*
  *Therefore, Socrates is mortal.*

- Which word are important? *All*? *Mortal*?

  *Borogoves are mimsy whenever it is brillig.*
  *It is now brillig and this thing is a borogove.*
  *Hence this thing is mimsy.*

# Propositional logic

- Propositions are atomic facts that can be either *True* or *False*
  - *The sky is blue*
  - *Donald Trump won the election*
- Connect propositions to make complex statements
  - *The sky is blue* and *it is raining*
  - If *Hillary Clinton won the election* then *demonetization will be rolled back*

# Propositional logic

- *A snobbish club takes in members only if they are rich or famous, with the added provision that no one who is famous but not rich is admitted.*
  - To join the club, you must be (a) rich, (b) rich but not famous, (c) famous but not rich, (d) both rich and famous?
- Let $\boldsymbol{R}$ denote rich, $\boldsymbol{F}$ denote famous
  - Membership criteria: ($\boldsymbol{R}$ or $\boldsymbol{F}$) and not($\boldsymbol{F}$ and not($\boldsymbol{R}$))
  - Try all possible combinations of setting $\boldsymbol{R}$ and $\boldsymbol{F}$ to $\{True, False\}$

# Syntax

- Assume a countably infinite set $\mathcal{P} = \{A_1, A_2, \ldots\}$ of atomic propositions
  - Ignore interpretation, just formal symbols
- Two logical connectives
  - $\neg$, unary (not, negation)
  - $\vee$, binary (or, disjunction)
- The set of formulas $\mathcal{F}$ is defined as follows
  - Every atomic proposition belongs to $\mathcal{F}$
  - If $F \in \mathcal{F}$, then $\neg F \in \mathcal{F}$
  - If $F, G \in \mathcal{F}$, then $F \vee G \in \mathcal{F}$

# Semantics

- The *meaning* of a formula is a truth value in { *True*, *False* }
  - For convenience, denote *True* by 1, *False* by 0
- An assignment $\mathcal{A} : \mathcal{P} \to \{0, 1\}$ fixes the truth value of each atomic proposition
- Extend to all formulas: $\mathcal{A} : \mathcal{F} \to \{0, 1\}$ is defined as follows
  - $F = A \in \mathcal{P}$: $\mathcal{A}(F) = \mathcal{A}(A)$
  - $F = \neg G$: $\mathcal{A}(F) = 1 - \mathcal{A}(G)$
  - $F = G_1 \vee G_1$: $\mathcal{A}(F) = 1$ if either $\mathcal{A}(G_1) = 1$ or $\mathcal{A}(G_2) = 1$ (or both)
- $\vee$ is *inclusive* — in natural language, *or* is usually *exclusive*
  - *I'll take a bus or a taxi*

# Derived connectives

- And: $F \wedge G \overset{defn}{=} \neg(\neg F \vee \neg G)$
  - $\mathcal{A}(F \wedge G) = 1$ iff $\mathcal{A}(F) = 1$ and $\mathcal{A}(G) = 1$
  - Note: Use parentheses for disambiguation where needed.
- Implies: $F \rightarrow G \overset{defn}{=} \neg F \vee G$
  - $\mathcal{A}(F \rightarrow G) = 0$ iff $\mathcal{A}(F) = 1$ and $\mathcal{A}(G) = 0$
  - If the premise is false, the formula is automatically true
  - *Hillary won election* $\rightarrow$ *demonetization rolled back*
- Iff: $F \leftrightarrow G \overset{defn}{=} (F \rightarrow G) \wedge (G \rightarrow F)$
  - $\mathcal{A}(F \wedge G) = 1$ iff $\mathcal{A}(F) = \mathcal{A}(G)$
- Truth values:
  - $\top \overset{defn}{=} (A_1 \vee \neg A_1)$, $\mathcal{A}(\top) = 1$
  - $\bot \overset{defn}{=} (A_1 \wedge \neg A_1)$, $\mathcal{A}(\bot) = 0$

# Derived connectives ...

- We will use derived connectives freely
- Derived connectives are convenient for writing formulas
- Minimal set of basic connectives makes proofs easier
- $\{\neg, \wedge\}$ can also be used as a basis

# Satisfiability, validity

- $\mathcal{A} \models F$ denotes that $\mathcal{A}(F) = 1$
- A formula $F$ is satisfiable if there is some assignment $\mathcal{A}$ such that $\mathcal{A} \models F$
  - $A$, $A \rightarrow B$
- A formula $F$ is valid if $\mathcal{A} \models F$ for every assignment $\mathcal{A}$
  - $A \vee \neg A$, $((A \rightarrow B) \wedge (B \rightarrow C)) \rightarrow (A \rightarrow C)$
- A formula $F$ is a contradiction if $\mathcal{A} \not\models F$ for every assignment $\mathcal{A}$
  - $C \wedge \neg C$, $((A \rightarrow B) \wedge (B \rightarrow C)) \rightarrow (A \rightarrow \neg C)$

# Satisfiability, validity

- Decision problem: Is $F$ satisfiable/valid?

**Fact**

$F$ is valid iff $\neg F$ is not satisfiable

- Sufficient to develop an algorithm for one of the two

# Deciding satisfiability

- Truth value of $F$ depends only on atomic propositions mentioned in $F$ — vocabulary of $F$
  - $\mathcal{A}(A \to (B \to A))$ is independent of $\mathcal{A}(C)$
- Formulas are finite, construct a truth table enumerating all possible values of atomic propositions

| $A$ | $B$ | $B \to A$ | $A \to (B \to A)$ |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |

- Satisfiable: at least one row evaluates to 1
- Valid: all rows evaluate to 1
- Truth table has $2^n$ rows — exponential algorithm

# Logical consequence

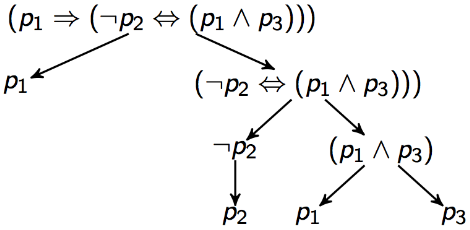
- $G$ is a logical consequence of $F$ if, whenver $F$ is true, $G$ must also be true
  - For every assignment $\mathcal{A}$, if $\mathcal{A} \models F$, then $\mathcal{A} \models G$
  - We write $F \models G$
- For a set $X = \{F_1, F_2, \ldots, F_m\}$, $X \models G$ if, whenever $\mathcal{A} \models F_i$ for each $i \in \{1, 2, \ldots, m\}$, it is also the case that $\mathcal{A} \models G$
- $F$ and $G$ are equivalent if $F \models G$ and $G \models F$
  - $F$ is true exactly when $G$ is true
  - Write $F \equiv G$

# Equivalences

- $\vee$ and $\wedge$ distribute over each other
  - $F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$
  - $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$
- De Morgan's laws, pushing negations inside $\vee$ and $\wedge$
  - $\neg(F \wedge G) \equiv \neg F \vee \neg G$
  - $\neg(F \vee G) \equiv \neg F \wedge \neg G$
- Double negation
  - $\neg\neg F \equiv F$

# Subformulas

- Any formula is a subformula of itself.
- Any subformula of $\boldsymbol{F}$ is also a subformula of $\neg\boldsymbol{F}$
- Any subformula of $\boldsymbol{F}$ or $\boldsymbol{G}$ is also a subformula of $\boldsymbol{F} \vee \boldsymbol{G}$.
- As usual, can associate a unique parse tree with every formula




- Subformulas correspond to subtrees of the parse tree

# Subformulas and substitution

**Substitution Theorem**

Let $F$ be a subformula of $H$, and let $F \equiv G$. Let $H'$ be the formula obtained by replacing $F$ in $H$ with $G$. Then $H \equiv H'$.

- How does one prove such a result?

# Structural induction

- To prove that property $\Theta$ holds for all formulas in $\mathcal{F}$, use induction over the structural complexity of the formula
  - Every atomic proposition in $\mathcal{P}$ satisfies $\Theta$
  - If $F \in \mathcal{F}$ satisfies $\Theta$, so does $\neg F$
  - If $F, G \in \mathcal{F}$ satisfy $\Theta$, so does $F \vee G$
- Having a small set of connectives reduces the number of cases to consider

# Negation normal form (NNF)

- Connectives are $\neg$, $\vee$, $\wedge$
- Negations appear only next to atomic propositions
- Translate $\rightarrow$, $\leftrightarrow$, ... into $\neg$, $\vee$, $\wedge$
- Use De Morgan's laws, double negation to push negations inwords
- $\neg(\boldsymbol{A} \rightarrow (\boldsymbol{B} \rightarrow \boldsymbol{A}))$
  $\Rightarrow \neg(\neg\boldsymbol{A} \vee (\neg\boldsymbol{B} \vee \boldsymbol{A}))$
  $\Rightarrow \neg\neg\boldsymbol{A} \wedge \neg(\neg\boldsymbol{B} \vee \boldsymbol{A})$
  $\Rightarrow \boldsymbol{A} \wedge (\neg\neg\boldsymbol{B} \wedge \neg\boldsymbol{A})$
  $\Rightarrow \boldsymbol{A} \wedge (\boldsymbol{B} \wedge \neg\boldsymbol{A})$

# Conjunctive normal form (CNF)

- Conjunction of clauses
- A clause is disjunction of literals
- A literal is an atomic proposition $\boldsymbol{A}$ or its negation $\neg \boldsymbol{A}$
- $(\boldsymbol{A} \vee \boldsymbol{B}) \wedge (\neg \boldsymbol{A} \vee \boldsymbol{C} \vee \neg \boldsymbol{D})$
- Can assume no literals are duplicated in a clause, no clauses are duplicated
- Each clause is a set of literals
- A formula in CNF is a set of clauses (a set of sets of literals)
- CNF is most convenient input format for SAT solving algorithms

# Converting NNF to CNF

- Use distributivity of $\vee$ over $\wedge$
- $(F_1 \wedge \neg\neg F_2) \vee (\neg G_1 \rightarrow G_2)$
  $\Rightarrow (F_1 \wedge F_2) \vee (\neg\neg G_1 \vee G_2)$
  $\Rightarrow (F_1 \wedge F_2) \vee (G_1 \vee G_2)$
  $\Rightarrow (F_1 \vee (G_1 \vee G_2)) \wedge (F_2 \vee (G_1 \vee G_2))$
  $\Rightarrow (F_1 \vee G_1 \vee G_2) \wedge (F_2 \vee G_1 \vee G_2)$
- Distributivity can cause exponential blowup
  - Input: $(A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \cdots (A_n \wedge B_n)$
  - CNF has $2^n$ clauses $(A_1 \vee A_2 \vee \cdots \vee A_n)$, $(B_1 \vee A_2 \vee \cdots \vee A_n)$, $\ldots$, $(B_1 \vee B_2 \vee \cdots \vee B_n)$

# Disjunctive normal form (DNF)

- Disjunction of conjuncts
- $(\boldsymbol{A} \wedge \boldsymbol{B} \wedge \neg \boldsymbol{C}) \vee (\neg \boldsymbol{A} \wedge \neg \boldsymbol{D} \wedge \boldsymbol{E})$
- Conversion procedure is similar to CNF — use distributivity
- Again exponential blowup, but satisfiability checking is easy
  - Check conjunctive clause by conjunctive clause

# Efficient transformation to CNF

- CNF and DNF conversion produce equivalent formulas
  - $F \equiv CNF(F)$, $F \equiv DNF(F)$
- For checking satisfiability, weaker transformation suffices
- $F$ and $G$ are equisatisfiable if $F$ is satisfiable whenever $G$ is satisfiable
  - Need not be satisfied in same assignment
  - There is some $\mathcal{A}_F$ with $\mathcal{A}_F \models F$ iff there is some $\mathcal{A}_G$ with $\mathcal{A}_G \models G$
- Can efficiently transform $F$ into CNF formula that is equisatifiable

# Tseitin transformation

- Want to transform $(A_1 \wedge A_2) \vee (B_1 \wedge B_2)$ into CNF
- Introduce a new switching proposition for $\vee$
- $(Z \rightarrow (A_1 \wedge A_2)) \wedge (\neg Z \rightarrow (B_1 \wedge B_2))$
- Rewrite as $(\neg Z \vee (A_1 \wedge A_2)) \wedge (Z \vee (B_1 \wedge B_2))$
- Expands as $(\neg Z \vee A_1) \wedge (\neg Z \vee A_2) \wedge (Z \vee B_1) \wedge (Z \vee B_2)$
- Do this recursively
  - To transform $((A_1 \wedge A_2) \vee (B_1 \wedge B_2)) \vee (C_1 \wedge C_2)$
  - Switching proposition $Z$ accounts for inner $\vee$ $((\neg Z \vee A_1) \wedge (\neg Z \vee A_2) \wedge (Z \vee B_1) \wedge (Z \vee B_2)) \vee (C_1 \wedge C_2)$
  - Add another switching proposition $Y$ for outer $\vee$ $(\neg Y \vee \neg Z \vee A_1) \wedge (\neg Y \vee \neg Z \vee A_2) \wedge (\neg Y \vee Z \vee B_1) \wedge (\neg Y \vee Z \vee B_2) \wedge (Y \vee C_1) \wedge (Y \vee C_2)$

# Tseitin transformation

- More formally, assume input $F$ is in NNF (only $\neg$, $\vee$, $\wedge$)
- Suppose $F$ has a subformula $G_1 \wedge \cdots \wedge G_n$ below an $\vee$
- Replace $G_1 \wedge \cdots \wedge G_n$ by a new proposition $Z$, resulting in $F(Z)$
- New formula is $F(Z) \wedge (\neg Z \vee G_1) \wedge (\neg Z \vee G_2) \wedge \cdots \wedge (\neg Z \vee G_n)$
- Equisatisfiable — by structural induction
- Blowup is quadratic — each literal becomes a clause, attached to new switching propositions according to nesting depth with respect to $\vee$
- Tseitin has also defined another transformation with a linear blowup

# Encoding hard problems

- Satisfiability is decidable using truth tables, but the procedure has exponential complexity
- Is this inherent?
- Apparently, yes! SAT was the first problem shown to be NP-Complete
  - Cook's Theorem: Encode computation of an NP machine $M$ on input $I$ as a polynomial-size propositional formula that is satisfiable iff $M$ accepts $I$
- Let's look at a simpler example

# Graph colouring

- Colour $G = (V, E)$ with at most $d$ colours
- Each vertex is assigned a colour so that any pair of vertices connected by an edge has different colours
- $V = \{v_1, v_2, \ldots, v_n\}$, $C = \{1, 2, \ldots, d\}$
- Proposition $p_{ij}$ — vertex $v_i$ is assigned colour $j$
- Each vertex has a colour

  For each $i \in \{1, 2, \ldots, n\}$, $(p_{i1} \vee p_{i2} \vee \cdots \vee p_{id})$

- Endpoints of edges are coloured differently

  For each $(v_i, v_j) \in E$, for each colour $k$, $(\neg p_{ik} \vee \neg p_{jk})$