A brief introduction to Logic – part I

A Brief Introduction to Logic - Outline

- Brief historical notes on logic
- Propositional Logic :Syntax
- Propositional Logic :Semantics
- Satisfiability and validity
- Modeling with Propositional logic
- Normal forms
- Deductive proofs and resolution

Historical view

- Philosophical Logic
  - 500 BC to 19th Century
- Symbolic Logic
  - Mid to late 19th Century
- Mathematical Logic
  - Late 19^{th} to mid 20^{th} Century
- Logic in Computer Science

Philosophical Logic

- 500 B.C – 19th Century
- Logic dealt with arguments in the natural language used by humans.
- Example
  - All men are mortal.
  - Socrates is a man
  - Therefore, Socrates is mortal.

Philosophical Logic

- Natural languages are very ambiguous.
  - Eric does not believe that Mary can pass any test.
    - ...does not believe that she can pass *some* test, or
    - ...does not believe that she can pass *all* tests
  - I only borrowed your car.
    - And not ‘borrowed and used’, or
    - And not ‘car and coat’
  - Tom hates Jim and he likes Mary.
    - Tom likes Mary, or
    - Jim likes Mary

- It led to many paradoxes.
  - “This sentence is a lie.” (The Liar’s Paradox)

Sophism
(From Wikipedia)

- ... **Sophism** generally refers to a particularly confusing, illogical and/or insincere argument used by someone to make a point, or, perhaps, not to make a point.
- **Sophistry** refers to [...] rhetoric that is designed to appeal to the listener on grounds other than the strict logical cogency of the statements being made.

The Sophist's Paradox

- A Sophist is sued for his tuition by the school that educated him. He argues that he must win, since, if he loses, the school didn't educate him well enough, and doesn't deserve the money.

- The school argues that he must lose, since, if he wins, he was educated well enough, and therefore should pay for it.

Logic in Computer Science

- Logic has a profound impact on computer-science. Some examples:
  - Propositional logic – the foundation of computers and circuitry
  - Databases – query languages
  - Programming languages (e.g. prolog)
  - Design Validation and verification
  - AI (e.g. inference systems)
  - ...

Logic in Computer Science

- Propositional Logic
- First Order Logic
- Higher Order Logic
- Temporal Logic
- ...
- ...

Propositional logic

- A proposition – a sentence that can be either true or false.
- Propositions:
  - x is greater than y
  - Noam wrote this letter

Propositional logic: Syntax

- The symbols of the language:
  - Propositional symbols (Prop): A, B, C,…
  - Connectives:
    - $\wedge$ and
    - $\vee$ or
    - $\neg$ not
    - $\rightarrow$ implies
    - $\leftrightarrow$ equivalent to
    - $\oplus$ xor (different than)
    - $\perp$, $\top$ False, True
  - Parenthesis:(,).
- Q1: how many different binary symbols can we define ?
- Q2: what is the minimal number of such symbols?

Formulas

- Grammar of well-formed propositional formulas
  - Formula := prop | (¬Formula) | (Formula o Formula).
  - ... where prop ∈ Prop and o is one of the binary relations

Formulas

- Examples of well-formed formulas:
  - $(\neg A)$
  - $(\neg(\neg A))$
  - $(A \land (B \land C))$
  - $(A \rightarrow (B \rightarrow C))$

- Correct expressions of Propositional Logic are full of unnecessary parenthesis.

Formulas

- **Abbreviations**. We write

$$A \circ B \circ C \circ \ldots$$

- in place of

$$(A \circ (B \circ (C \circ \ldots)))$$

- Thus, we write

$$A \wedge B \wedge C, \qquad A \rightarrow B \rightarrow C, \ldots$$

- in place of

$$(A \wedge (B \wedge C)), \qquad (A \rightarrow (B \rightarrow C))$$

Formulas

- We omit parenthesis whenever we may restore them through operator precedence:
- $\neg$ binds more strictly than $\wedge$, $\vee$, and $\wedge$, $\vee$ bind more strictly than $\rightarrow$, $\leftrightarrow$.
- Thus, we write:

$\neg\neg A$	for	$(\neg(\neg A))$,
$\neg A \wedge B$	for	$((\neg A) \wedge B)$
$A \wedge B \rightarrow C$	for	$((A \wedge B) \rightarrow C)$, …

Propositional Logic: Semantics

- Truth tables define the semantics (=meaning) of the operators
- Convention: 0 = false, 1 = true

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	1

Propositional Logic: Semantics

- Truth tables define the semantics (=meaning) of the operators

p	q	¬p	p ↔ q	p ⊕ q
0	0	1	1	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	0

Back to Q1

- Q1: How many binary operators can we define that have different semantic definition ?
  - A: 16

Assignments

- Definition: A truth-values assignment, α, is an element of 2^{Prop} (i.e., $\alpha \in 2^{Prop}$).
- In other words, α is a subset of the variables that are assigned true.
- Equivalently, we can see α as a mapping from variables to truth values:
  $\alpha : Prop \mapsto \{0,1\}$
  - Example: α: $\{A \mapsto 0, B \mapsto 1, \ldots\}$

Satisfaction relation (⊨): intuition

- An assignment can either satisfy or not satisfy a given formula.

- $\alpha \models \varphi$ means
  - α satisfies φ or
  - φ holds at α or
  - α is a model of φ
- We will first see an example.
- Then we will define these notions formally.

Example

- Let $\phi = (A \vee (B \rightarrow C))$
- Let $\alpha = \{A \mapsto 0, B \mapsto 0, C \mapsto 1\}$
- Q: Does α satisfy ϕ?
  - (in symbols: does it hold that $\alpha \models \phi$?)
- A: $(0 \vee (0 \rightarrow 1)) = (0 \vee 1) = 1$
  - Hence, $\alpha \models \phi$.
- Let us now formalize an evaluation process.

The satisfaction relation ($\models$): formalities

- $\models$ is a relation: $\models \subseteq (2^{Prop} \text{ x Formula})$
  - Examples:
    - $(\{a\}, a \vee b)$ // the assignment $\alpha = \{a\}$ satisfies $a \vee b$
    - $(\{a,b\}, a \wedge b)$

- Alternatively: $\models \subseteq (\{0,1\}^{Prop} \times \text{Formula})$
  - Examples:
    - $(01, a \vee b)$ // the assignment $\alpha = \{a \mapsto 0, b \mapsto 1\}$ satisfies $a \vee b$
    - $(11, a \wedge b)$

The satisfaction relation ($\models$): formalities

- $\models$ is defined recursively:
  - $\alpha \models p$ if α (p) = true
  - $\alpha \models \neg\varphi$ if $\alpha \not\models \varphi$.
  - $\alpha \models \varphi_1 \wedge \varphi_2$ if $\alpha \models \varphi_1$ and $\alpha \models \varphi_2$
  - $\alpha \models \varphi_1 \vee \varphi_2$ if $\alpha \models \varphi_1$ or $\alpha \models \varphi_2$
  - $\alpha \models \varphi_1 \rightarrow \varphi_2$ if $\alpha \models \varphi_1$ implies $\alpha \models \varphi_2$
  - $\alpha \models \varphi_1 \leftrightarrow \varphi_2$ if $\alpha \models \varphi_1$ iff $\alpha \models \varphi_2$

From definition to an evaluation algorithm

- Truth Evaluation Problem
  - Given $\varphi \in$ Formula and $\alpha \in 2^{AP(\varphi)}$, does $\alpha \vDash \varphi$?

```
Eval(φ, α){
   If φ ≡ A, return α(A).
   If φ ≡ (¬φ1) return ¬Eval(φ1, α))
   If φ ≡ (φ1 ○ φ2)
     return Eval(φ1, α) ○ Eval(φ2, α)
   }
```

- Eval uses polynomial time and space.

It doesn't give us more than what we already know...

- Recall our example
  - Let $\phi = (A \vee (B \rightarrow C))$
  - Let $\alpha = \{A \mapsto 0, B \mapsto 0, C \mapsto 1\}$

- $Eval(\phi,\alpha) = Eval(A, \alpha) \vee Eval(B \rightarrow C, \alpha) =$
  $0 \vee Eval(B, \alpha) \rightarrow Eval(C, \alpha) =$
  $0 \vee (0 \rightarrow 1) = 0 \vee 1 = 1$

- Hence, $\alpha \vDash \phi$.

We can now extend the truth table to formulas

p	q	$(p \rightarrow (q \rightarrow p))$	$(p \wedge \neg p)$	$p \vee \neg q$
0	0	1	0	1
0	1	1	0	0
1	0	1	0	1
1	1	1	0	1

We can now extend the truth table to formulas

x_1	x_2	x_3	$x_1 \rightarrow (x_2 \rightarrow \neg x_3)$
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Set of assignments

- Intuition: a formula specifies a set of truth assignments.
- Function models: Formula $\mapsto 2^{2^{\mathbf{Prop}}}$
  (a formula $\mapsto$ set of satisfying assignments)
- Recursive definition:
  - $\text{models}(A) = \{\alpha \mid \alpha(A) = 1\}, A \in \text{Prop}$
  - $\text{models}(\neg\varphi_1) = 2^{\text{Prop}} - \text{models}(\varphi_1)$
  - $\text{models}(\varphi_1 \wedge \varphi_2) = \text{models}(\varphi_1) \cap \text{models}(\varphi_2)$
  - $\text{models}(\varphi_1 \vee \varphi_2) = \text{models}(\varphi_1) \cup \text{models}(\varphi_2)$
  - $\text{models}(\varphi_1 \rightarrow \varphi_2) = (2^{\text{Prop}} - \text{models}(\varphi_1)) \cup \text{models}(\varphi_2)$

Example

- models $(A \vee B) = \{\{10\},\{01\},\{11\}\}$

- This is compatible with the recursive definition:

  $models(A \vee B) =$
  $models(A) \cup models(B) =$
  $\{\{10\},\{11\}\} \cup \{\{01\},\{11\}\} =$
  $\{\{10\},\{01\},\{11\}\}$

Theorem

- Let $\varphi \in$ Formula and $\alpha \in 2^{Prop}$, then the following statements are equivalent:
  1. $\alpha \models \varphi$
  2. $\alpha \in$ models(φ)

Only the projected assignment matters...

- AP(φ) – the Atomic Propositions in φ.
- Clearly AP(φ) $\subseteq$ Prop.
- Let $\alpha_1, \alpha_2 \in 2^{Prop}$, $\varphi \in$ Formula.
- Lemma: if $\alpha_1|_{AP(\varphi)} = \alpha_2|_{AP(\varphi)}$, then

$$\alpha_1 \vDash \varphi \text{ iff } \alpha_2 \vDash \varphi$$

Projection

Corollary: $\alpha \vDash \varphi$ iff $\alpha|_{AP(\varphi)} \vDash \varphi$

- We will assume, for simplicity, that Prop = AP(φ).

Extension of $\models$ to sets of assignments

- Let $\varphi \in$ Formula
- Let T be a set of assignments, i.e., $T \subseteq 2^{2^{Prop}}$
- Definition.

  $T \models \varphi$ if $T \subseteq models(\varphi)$

  - i.e., $\models \; \subseteq 2^{2^{Prop}} \times$ Formula

Extension of $\models$ to formulas

- $\models \subseteq 2^{\text{Formula}} \times 2^{\text{Formula}}$
- Definition. Let Γ_1, Γ_2 be prop. formulas.

$\Gamma_1 \models \Gamma_2$

iff $\text{models}(\Gamma_1) \subseteq \text{models}(\Gamma_2)$

iff for all $\alpha \in 2^{\text{Prop}}$

if $\alpha \models \Gamma_1$ then $\alpha \models \Gamma_2$

Examples:

$x_1 \wedge x_2 \models x_1 \vee x_2$

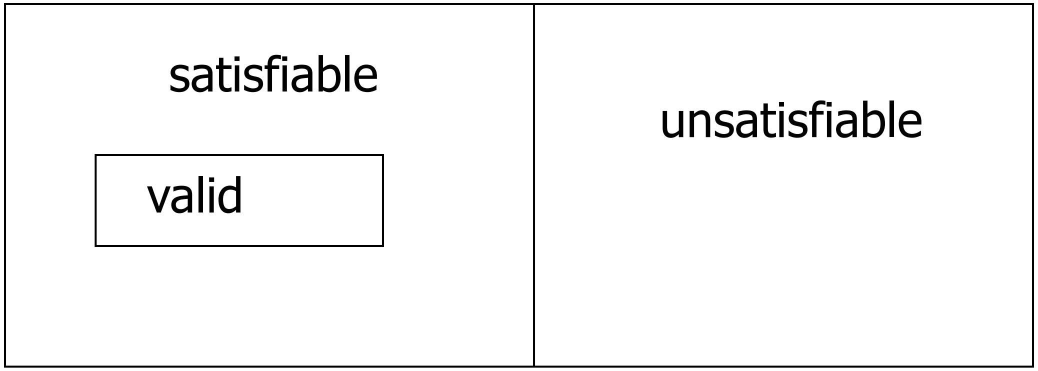
$x_1 \wedge x_2 \models x_2 \vee x_3$

Semantic Classification of formulas

- A formula φ is called valid if models(φ) = 2^{Prop}. (also called a tautology).

- A formula φ is called satisfiable if models(φ) ≠ ∅.

- A formula φ is called unsatisfiable if models(φ) = ∅. (also called a contradiction).

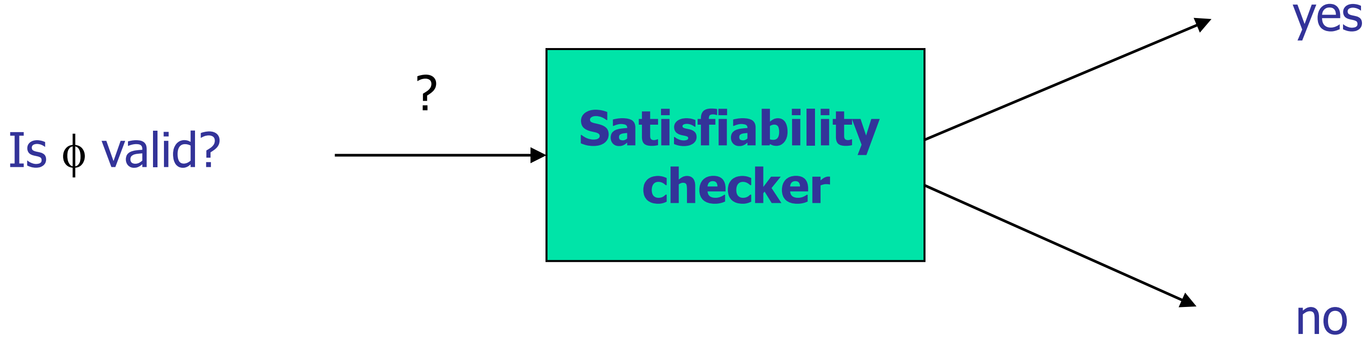


Validity, satisfiability... in truth tables

p	q	$(p \rightarrow (q \rightarrow q))$	$(p \wedge \neg p)$	$p \vee \neg q$
0	0	1	0	1
0	1	1	0	0
1	0	1	0	1
1	1	1	0	1

Characteristics of valid/sat. formulas...

- Lemma
  - A formula φ is valid iff $\neg\varphi$ is unsatisfiable
  - φ is satisfiable iff $\neg\varphi$ is not valid



Look what we can do now...

- We can write:
  - $\models \phi$ when ϕ is valid
  - $\not\models \phi$ when ϕ is not valid
  - $\not\models \neg\phi$ when ϕ is satisfiable
  - $\models \neg\phi$ when ϕ is unsatisfiable

Examples

- $(x_1 \wedge x_2) \rightarrow (x_1 \vee x_2)$ is valid
- $(x_1 \vee x_2) \rightarrow x_1$ is satisfiable
- $(x_1 \wedge x_2) \wedge \neg x_1$ is unsatisfiable

Time for equivalences

- Here are some valid formulas:
  - $\models A \wedge 1 \leftrightarrow A$
  - $\models A \wedge 0 \leftrightarrow 0$
  - $\models \neg\neg A \leftrightarrow A$ // The double-negation rule
  - $\models A \wedge (B \vee C) \leftrightarrow (A \wedge B) \vee (A \wedge C)$

- Some more (De-Morgan rules):
  - $\models \neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$
  - $\models \neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$

A minimal set of binary operators

- Recall the question: what is the minimal set of operators necessary?
- A: Through such equivalences all Boolean operators can be written with a single operator (NAND).
  - Indeed, typically industrial circuits only use one type of logical gate

- We'll see how two are enough: $\neg$ and $\wedge$
  - Or: $\models (A \vee B) \leftrightarrow \neg(\neg A \wedge \neg B)$
  - Implies: $\models (A \rightarrow B) \leftrightarrow (\neg A \vee B)$
  - Equivalence: $\models (A \leftrightarrow B) \leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$
  - ...

The decision problem of formulas

- The decision problem:

Given a propositional formula ϕ, is ϕ satisfiable ?

- An algorithm that always terminates with a correct answer to this problem is called a **decision procedure** for propositional logic.

A Brief Introduction to Logic - Outline

- Brief historical notes on logic
- Propositional Logic :Syntax
- Propositional Logic :Semantics
- Satisfiability and validity
- Modeling with Propositional logic
- Normal forms
- Deductive proofs and resolution

Before we solve this problem...

- Q: Suppose we can solve the satisfiability problem... how can this help us?

- A: There are numerous problems in the industry that are solved via the satisfiability problem of propositional logic
  - Logistics...
  - Planning...
  - Electronic Design Automation industry...
  - Cryptography...
  - ... (every NP-P problem...)

Example 2: placement of wedding guests

- Three chairs in a row: 1,2,3
- We need to place Aunt, Sister and Father.
- Constraints:
  - Aunt doesn't want to sit near Father
  - Aunt doesn't want to sit in the left chair
  - Sister doesn't want to sit to the right of Father

- Q: Can we satisfy these constraints?

Example 2 (cont'd)

- Denote: Aunt = 1, Sister = 2, Father = 3
- Introduce a propositional variable for each pair (person, place).
- x_{ij} = person i is sited in place j, for $1 \le i,j \le 3$
- Constraints:
  - Aunt doesn't want to sit near Father:
    $((x_{1,1} \vee x_{1,3}) \rightarrow \neg x_{3,2}) \wedge (x_{1,2} \rightarrow (\neg x_{3,1} \wedge \neg x_{3,3}))$
  - Aunt doesn't want to sit in the left chair
    $\neg x_{1,1}$
  - Sister doesn't want to sit to the right of Father
    $x_{3,1} \rightarrow \neg x_{2,2} \wedge x_{3,2} \rightarrow \neg x_{2,3}$

Example 2 (cont'd)

- More constraints:
  - Each person is placed:
    $(x_{1,1} \vee x_{1,2} \vee x_{1,3}) \wedge$
    $(x_{2,1} \vee x_{2,2} \vee x_{2,3}) \wedge$
    $(x_{3,1} \vee x_{3,2} \vee x_{3,3})$
  - Or, more concisely: $\bigwedge_{i=1}^{3} \bigvee_{j=1}^{3} x_{i,j}$
  - Not more than one person per chair:

$$\bigwedge_{j=1}^{3} \bigwedge_{i=1}^{2} \bigwedge_{k=i+1}^{3} (\neg x_{i,j} \vee \neg x_{k,j})$$

- Overall 9 variables, 23 conjoined constraints.

Example 3: assignment of frequencies

- n radio stations
- For each assign one of k transmission frequencies, $k < n$.
- E -- set of pairs of stations, that are too close to have the same frequency.

- Q: which graph problem does this remind you of ?

Example 3 (cont'd)

- $x_{i,j}$ – station i is assigned frequency j, for $1 \leq i \leq n$, $1 \leq j \leq k$.
  - Every station is assigned at least one frequency:

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{k} x_{ij}$$

  - Every station is assigned not more than one frequency:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{k-1} (x_{ij} \rightarrow \bigwedge_{j<t\leq k} \neg x_{it})$$

  - Close stations are not assigned the same frequency. For each $(i,j) \in E$,

$$\bigwedge_{t=1}^{k} (x_{it} \rightarrow \neg x_{jt})$$

Two classes of algorithms for validity

- Q: Is φ satisfiable (/¬φ is valid) ?
- Complexity: NP-Complete (the first-ever! – Cook's theorem)

- Two classes of algorithms for finding out:
  1. Enumeration of possible solutions (Truth tables etc).
  2. Deduction

- More generally (beyond propositional logic):
  - Enumeration is possible only in some logics.
  - Deduction cannot necessarily be fully automated.

The satisfiability problem: enumeration

- Given a formula φ, is φ satisfiable?

```
Boolean SAT(φ) {
  B:=false
  for all α ∈ 2^AP(φ)
    B = B ∨ Eval(φ,α)
  end
  return B
 }
```

- There must be a better way to do that in practice.

A Brief Introduction to Logic - Outline

- Brief historical notes on logic
- Propositional Logic :Syntax
- Propositional Logic :Semantics
- Satisfiability and validity
- Modeling with Propositional logic
- Normal forms
- Deductive proofs and resolution

Definitions…

- Definition: A literal is either an atom or a negation of an atom.
- Let $\phi = \neg(A \vee \neg B)$. Then:
  - Atoms: $AP(\phi) = \{A,B\}$
  - Literals: $lit(\phi) = \{A, \neg B\}$

- Equivalent formulas can have different literals
  - $\phi = \neg(A \vee \neg B) = \neg A \wedge B$
  - Now $lit(\phi) = \{\neg A, B\}$

Definitions…

- Definition: a term is a conjunction of literals
  - Example: $(A \wedge \neg B \wedge C)$

- Definition: a clause is a disjunction of literals
  - Example: $(A \vee \neg B \vee C)$

Negation Normal Form (NNF)

- Definition: A formula is said to be in Negation Normal Form (NNF) if it only contains $\neg$, $\wedge$ and $\vee$ connectives and only atoms can be negated.

- Examples:
  - $\phi_1 = \neg(A \vee \neg B)$ is not in NNF
  - $\phi_2 = \neg A \wedge B$ is in NNF

Converting to NNF

- Every formula can be converted to NNF in linear time:
  - Eliminate all connectives other than $\wedge$, $\vee$, $\neg$
  - Use De Morgan and double-negation rules to push negations to the right
- Example: $\phi = \neg(A \rightarrow \neg B)$
  - Eliminate '$\rightarrow$': $\phi = \neg(\neg A \vee \neg B)$
  - Push negation using De Morgan: $\phi = (\neg\neg A \wedge \neg\neg B)$
  - Use Double negation rule: $\phi = (A \wedge B)$

Disjunctive Normal Form (DNF)

- Definition: A formula is said to be in Disjunctive Normal Form (DNF) if it is a disjunction of terms.
  - In other words, it is a formula of the form

$$\bigvee_i \left(\bigwedge_j l_{i,j} \right)$$

    where $l_{i,j}$ is the j-th literal in the i-th term.

- Examples
  - $\phi = (A \wedge \neg B \wedge C) \vee (\neg A \wedge D) \vee (B)$ is in DNF

- DNF is a special case of NNF

Converting to DNF

- Every formula can be converted to DNF in exponential time and space:
  - Convert to NNF
  - Distribute disjunctions following the rule:
    $\models A \wedge (B \vee C) \leftrightarrow ((A \wedge B) \vee (A \wedge C))$
- Example:
  - $\phi = (A \vee B) \wedge (\neg C \vee D) =$
    $((A \vee B) \wedge (\neg C)) \vee ((A \vee B) \wedge D) =$
    $(A \wedge \neg C) \vee (B \wedge \neg C) \vee (A \wedge D) \vee (B \wedge D)$
  - Q: how many clauses would the DNF have had we started from a conjunction of n clauses ?

Satisfiability of DNF

- Is the following DNF formula satisfiable?

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

- What is the complexity of satisfiability of DNF formulas?

Conjunctive Normal Form (CNF)

- Definition: A formula is said to be in Conjunctive Normal Form (CNF) if it is a conjunction of clauses.
  - In other words, it is a formula of the form

$$\bigwedge_i \left(\bigvee_j l_{i,j} \right)$$

where $l_{i,j}$ is the j-th literal in the i-th term.

- Examples
  - $\phi = (A \vee \neg B \vee C) \wedge (\neg A \vee D) \wedge (B)$ is in CNF
- CNF is a special case of NNF

Converting to CNF

- Every formula can be converted to CNF:
  - in exponential time and space with the same set of atoms
  - in linear time and space if new variables are added.
    - In this case the original and converted formulas are "equi-satisfiable".
    - This technique is called Tseitin's encoding.

Converting to CNF: the exponential way

CNF(ϕ) {

case

ϕ is a literal: return ϕ

ϕ is $\psi_1 \wedge \psi_2$: return CNF(ψ_1) $\wedge$ CNF(ψ_2)

ϕ is $\psi_1 \vee \psi_2$: return Dist(CNF(ψ_1),CNF(ψ_2))

}

Dist(ψ_1,ψ_2) {

case

ψ_1 is $\phi_{11} \wedge \phi_{12}$: return Dist($\phi_{11}$,$\psi_2$) $\wedge$ Dist(ψ_{12},ψ_2)
ψ_2 is $\phi_{21} \wedge \phi_{22}$: return Dist($\psi_1$,$\phi_{21}$) $\wedge$ Dist(ψ_1,ϕ_{22})
else: return $\psi_1 \vee \psi_2$

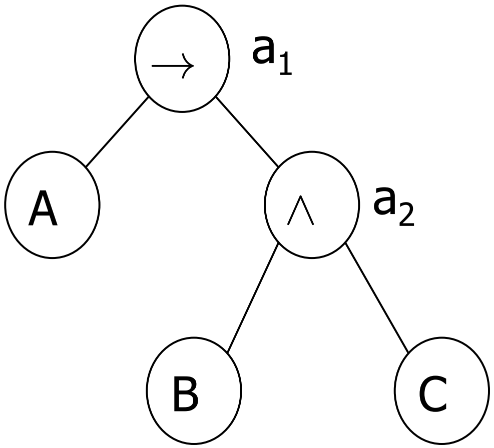
Converting to CNF: the exponential way

- Consider the formula
  $\phi = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$
- CNF(ϕ)=
  $(x_1 \vee x_2) \wedge$
  $(x_1 \vee y_2) \wedge$
  $(y_1 \vee x_2) \wedge$
  $(y_1 \vee y_2)$

- Now consider: $\phi_n = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \cdots \vee (x_n \wedge y_n)$
- Q: How many clauses CNF(ϕ) returns ?
- A: 2^n

Converting to CNF: Tseitin's encoding

- Consider the formula $\phi = (A \rightarrow (B \wedge C))$
- The parse tree:



- Associate a new auxiliary variable with each gate.
- Add constraints that define these new variables.
- Finally, enforce the root node.

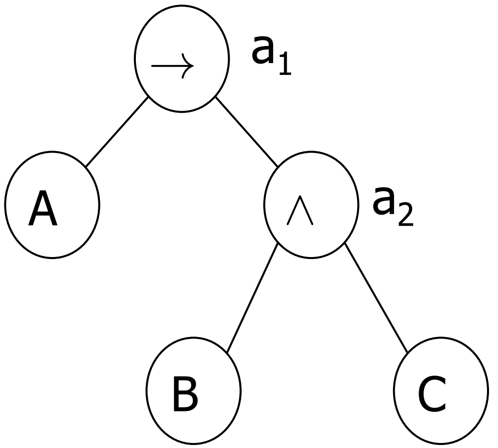
Converting to CNF: Tseitin's encoding

- Need to satisfy:

$(a_1 \leftrightarrow (A \rightarrow a_2)) \wedge$

$(a_2 \leftrightarrow (B \wedge C)) \wedge$

(a_1)



- Each such constraint has a CNF representation with 3 or 4 clauses.

Converting to CNF: Tseitin's encoding

- Need to satisfy:

  $(a_1 \leftrightarrow (A \rightarrow a_2)) \wedge$

  $(a_2 \leftrightarrow (B \wedge C)) \wedge$

  (a_1)

- First: $(a_1 \vee A) \wedge (a_1 \vee \neg a_2) \wedge (\neg a_1 \vee \neg A \vee a_2)$
- Second: $(\neg a_2 \vee B) \wedge (\neg a_2 \vee C) \wedge (a_2 \vee \neg B \vee \neg C)$

Converting to CNF: Tseitin's encoding

- Let's go back to
  $\phi_n = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \cdots \vee (x_n \wedge y_n)$
- With Tseitin's encoding we need:
  - n auxiliary variables $a_1, \ldots, a_n$.
  - Each adds 3 constraints.
  - Top clause: $(a_1 \vee \cdots \vee a_n)$

- Hence, we have
  - $3n + 1$ clauses, instead of 2^n.
  - $3n$ variables rather than $2n$.

What now?

- Time to solve the decision problem for propositional logic.

  - The only algorithm we saw so far was building truth tables.

Two classes of algorithms for validity

- Q: Is φ valid ?
  - Equivalently: is $\neg\varphi$ satisfiable?
- Two classes of algorithm for finding out:
  1. Enumeration of possible solutions (Truth tables etc).
  2. Deduction

- In general (beyond propositional logic):
  - Enumeration is possible only in some theories.
  - Deduction typically cannot be fully automated.

The satisfiability Problem: enumeration

- Given a formula φ, is φ satisfiable?

```
Boolean SAT(φ) {
   B:=false
   for all α ∈ 2^AP(φ)
      B = B ∨ Eval(φ,α)
   end
   return B
 }
```

- NP-Complete (the first-ever! – Cook's theorem)

A Brief Introduction to Logic - Outline

- Brief historical notes on logic
- Propositional Logic :Syntax
- Propositional Logic :Semantics
- Satisfiability and validity
- Modeling with Propositional logic
- Normal forms
- Deductive proofs and resolution

Deduction requires axioms and Inference rules

- Inference rules:

$$\frac{\text{Antecedents}}{\text{Consequent}} \quad \text{(rule-name)}$$

- Examples:

$$\frac{A \rightarrow B \qquad B \rightarrow C}{A \rightarrow C} \quad \text{(trans)}$$

$$\frac{A \rightarrow B \qquad A}{B} \quad \text{(M.P.)}$$

Axioms

- Axioms are inference rules with no antecedents, e.g.,

$$\frac{}{A \rightarrow (B \rightarrow A)} \quad (H1)$$

- We can turn an inference rule into an axiom if we have '→' in the logic.

- So the difference between them is not sharp.

Proofs

- A proof uses a given set of inference rules and axioms.
- This is called the *proof system*.
- Let $\mathcal{H}$ be a proof system.

- $\Gamma \vdash_{\mathcal{H}} \varphi$ means: there is a proof of φ in system $\mathcal{H}$ whose premises are included in Γ

- $\vdash_{\mathcal{H}}$ is called the provability relation.

Example

- Let $\mathcal{H}$ be the proof system comprised of the rules Trans and M.P. that we saw earlier.

- Does the following relation holds?

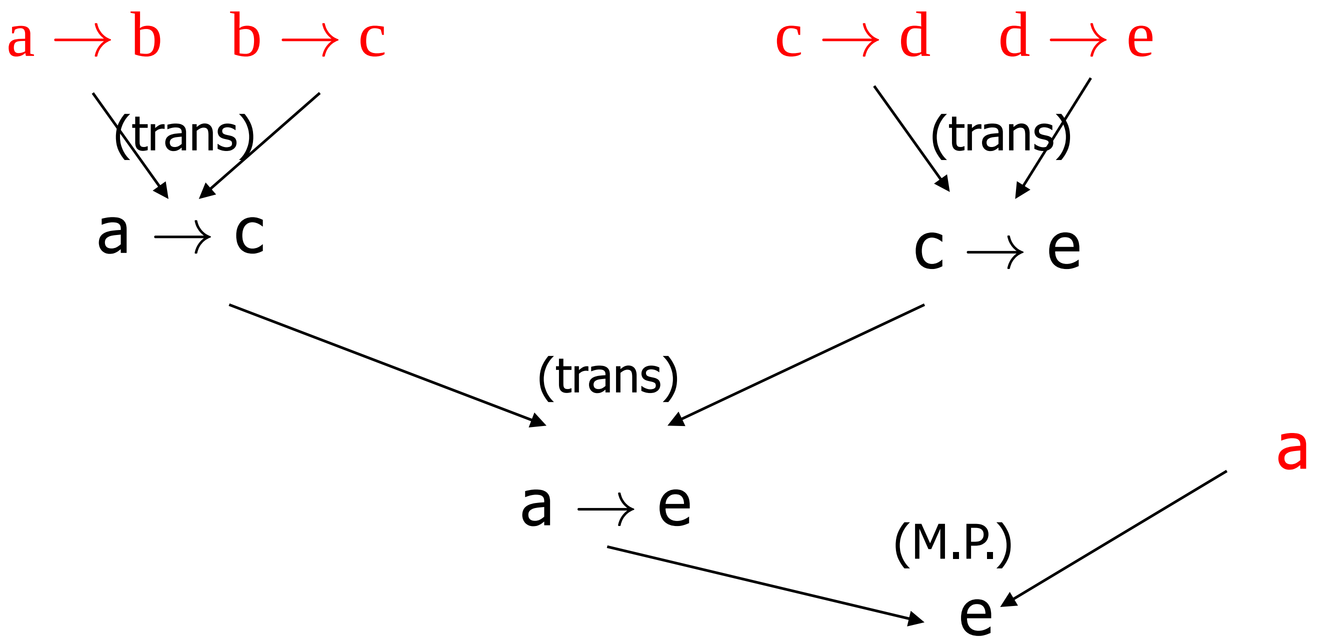
$$a \rightarrow b, b \rightarrow c,\ c \rightarrow d,\ d \rightarrow e,\ a\ \vdash_{\mathcal{H}} e$$

Deductive proof: example

$$a \rightarrow b, b \rightarrow c, c \rightarrow d, d \rightarrow e, a \vdash_{\mathcal{H}} e$$

1. $a \rightarrow b$	premise
2. $b \rightarrow c$	premise
3. $a \rightarrow c$	1,2,Trans
4. $c \rightarrow d$	premise
5. $d \rightarrow e$	premise
6. $c \rightarrow e$	4,5, Trans
7. $a \rightarrow e$	3,6, Trans
8. a	premise
9. e	7,8.M.P.

Proof graph (DAG)



Roots: premises

Proofs

- The problem: $\vdash$ is a relation defined by syntactic transformations of the underlying proof system.

- For a given proof system $\mathcal{H}$,
  - does $\vdash$ conclude "correct" conclusions from premises ?
  - Can we conclude all true statements with $\mathcal{H}$?

- Correct with respect to what ?
  - With respect to the semantic definition of the logic. In the case of propositional logic truth tables gives us this.

Soundness and completeness

- Let $\mathcal{H}$ be a proof system
- *Soundness* of $\mathcal{H}$: if $\vdash_{\mathcal{H}} \varphi$ then $\models \varphi$
- *Completeness* of $\mathcal{H}$: if $\models \varphi$ then $\vdash_{\mathcal{H}} \varphi$
- How to prove soundness and completeness ?

Soundness and completeness of procedures

- The definitions so far referred to proof systems
- What does soundness and completeness mean for general decision algorithms ?

Soundness and Completeness

- Definition [soundness]: A procedure for the decision problem is sound if
  - when it returns “Valid”...
  - ... the input formula is valid.

- Definition [completeness]: A procedure for the decision problem is complete if
  - it always terminates, and
  - it returns “Valid” when the input formula is valid.

Decidability

- Definition [decidability]
  A logic is decidable if there is a sound and complete algorithm that decides if a well-formed expression in this logic is valid.

Soundness and Completeness

- Soundness: “when I say that it rains, it rains”
- Completeness: “If asked, I always reply (in a finite time…). When it rains, I say that it rains”

Soundness and Completeness (cont'd)

- Algorithm #1: for checking if it rains outside:
  "stand right outside the door and say 'it rains'"

- It is not sound because you might say it rains when it doesn't.
- But it is complete: you always get an answer in a finite time, and when it rains you say it rains.

Soundness and Completeness (cont'd)

- Algorithm #2 for checking if it rains outside:
  "stand right outside the door and say 'it rains' if and only if you feel the rain"

- It is sound because you say it rains only if it actually rains.
- It is incomplete because you do not say anything if it doesn't rain (we do not know whether it doesn't rain, or it takes the person too long to answer…).

Soundness and Completeness (cont'd)

- Algorithm #2 for checking if it rains outside:
  "stand right outside the door and say 'it rains' if and only if you feel the rain, *after a second*"

- It is sound because you say it rains only if it actually rains.
- It is complete.

- Now let's go back to proof systems…

Example: Hilbert axiom system ($\mathcal{H}$)

- Let $\mathcal{H}$ be (M.P) + the following axiom schemas:

$$\frac{}{A \to (B \to A)} \quad (H1)$$

$$\frac{}{((A \to (B \to C)) \to ((A \to B) \to (A \to C))} \quad (H2)$$

$$\frac{}{(\neg B \to \neg A) \to (A \to B)} \quad (H3)$$

- $\mathcal{H}$ is sound and complete

Soundness and completeness

- To prove soundness of $\mathcal{H}$, prove the soundness of its axioms and inference rules (easy with truth-tables). For example:

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

- Completeness – harder, but possible.

The resolution inference system

- The resolution inference rule for CNF:

$$\frac{(l \lor l_1 \lor .. \lor l_n) \qquad (\neg l \lor l'_1 \lor .. \lor l'_n)}{(l_1 \lor .. \lor l_n \lor l'_1 \lor .. \lor l'_n)} \quad \text{(Resolution)}$$

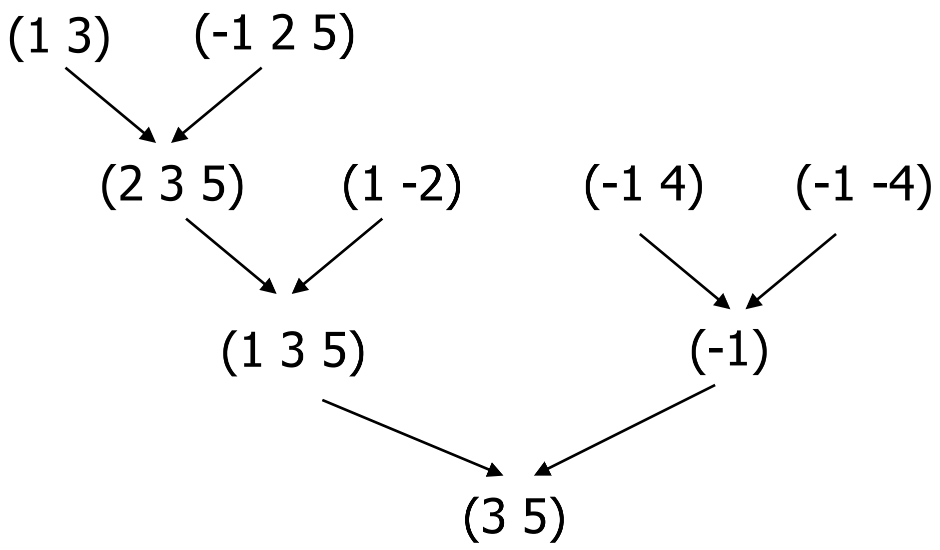
- Example:

$$\frac{(a \lor b) \qquad (\neg a \lor c)}{(b \lor c)}$$

Proof by resolution

- Let $\varphi = (1\ 3) \wedge (-1\ 2\ 5) \wedge (-1\ 4) \wedge (-1\ -4) \wedge (1\ -2)$

- We'll try to prove $\varphi \rightarrow (3\ 5)$



Resolution

- Resolution is a sound and complete inference system for CNF
- If the input formula is unsatisfiable, there exists a proof of the empty clause

Example

- Let φ = (1 3) ∧ (-1 2) ∧ (-1 4) ∧ (-1 -4) ∧ (1 -2) ∧ (-3)