ECE570 Lecture 15: Propositional Logic

Jeffrey Mark Siskind

School of Electrical and Computer Engineering

Fall 2013

PURDUE UNIVERSITY

What is a Logic?

A *logic* defines a:

syntax A set of *well formed* formulas

- a 'language' in a formal sense
- can be specified by a grammar

semantics A 'meaning' for each well formed formula

- typically meanings are Boolean truth values

Syntax of Propositional Logic

atomic formulas $p, q, p_1, p(a), \ldots$

- given a finite or infinite set of atomic formulas
- $p(a)$ and $p(b)$ are distinct and unrelated

compound formulas $\neg\Phi, \Phi \wedge \Psi, \Phi \vee \Psi, \Phi \rightarrow \Psi, \Phi \leftrightarrow \Psi$

- $\Phi \rightarrow \Psi$ sometimes written as $\Phi \supset \Psi$
- $\Phi \leftrightarrow \Psi$ sometimes written as $\Phi \equiv \Psi$

Syntax of Propositional Logic as a Nondeterministic Scheme Program

```
(define (an-atomic-formula p) (a-member-of p))

(define (a-compound-formula p)
 (either `(not ,(a-formula p))
         `(and ,(a-formula p) ,(a-formula p))
         `(or ,(a-formula p) ,(a-formula p))
         `(implies ,(a-formula p) ,(a-formula p))
         `(iff ,(a-formula p) ,(a-formula p))))

(define (a-formula p)
 (either (an-atomic-formula p)
         (a-compound-formula p)))
```

Semantics of Propositional Logic

An *interpretation* I is a total map from atomic formulas to truth values.
The meaning or *value* of Φ under I is given by a *valuation function* $\mathcal{V}(\Phi, I)$:

- $\mathcal{V}(\Phi, I)$ is $I(\Phi)$ when Φ is an atomic formula.
- $\mathcal{V}(\neg\Phi, I)$ is true if and only if $\mathcal{V}(\Phi, I)$ is false.
- $\mathcal{V}(\Phi \wedge \Psi, I)$ is true if and only if both $\mathcal{V}(\Phi, I)$ and $\mathcal{V}(\Psi, I)$ are true.
- $\mathcal{V}(\Phi \vee \Psi, I)$ is true if and only if either $\mathcal{V}(\Phi, I)$ or $\mathcal{V}(\Psi, I)$ is true.
- $\mathcal{V}(\Phi \rightarrow \Psi, I)$ is true if and only if $\mathcal{V}(\Phi, I)$ is false or $\mathcal{V}(\Psi, I)$ is true.
- $\mathcal{V}(\Phi \leftrightarrow \Psi, I)$ is true if and only if $\mathcal{V}(\Phi, I)$ and $\mathcal{V}(\Psi, I)$ have the same truth value.

Semantics of Propositional Logic as a Nondeterministic Scheme Program

```
(define (v phi i)
 (if (list? phi)
     (case (first phi)
      ((not) (eq? (v (second phi) i) #f))
      ((and) (and (eq? (v (second phi) i) #t)
                  (eq? (v (third phi) i) #t)))
      ((or) (or (eq? (v (second phi) i) #t)
                (eq? (v (third phi) i) #t)))
      ((implies) (or (eq? (v (second phi) i) #f)
                     (eq? (v (third phi) i) #t)))
      ((iff) (eq? (v (second phi) i) (v (third phi) i))))
     (if (member phi i) #t #f)))
```

Some Definitions—I

- I is a *model* of Σ iff for all $\Phi \in \Sigma$ $\mathcal{V}(\Phi, I)$ is true.
- Σ is *valid* (*universally valid*) iff every I is a model of Σ.
- Σ is *satisfiable* (*consistent*) iff some I is a model of Σ.
- Σ is *unsatisfiable* (*inconsistent*) iff no I is a model of Σ.
- $M(\Sigma)$ denotes the set of all models of Σ.
- Treat Φ as $\{\Phi\}$.

Some Definitions—II

- A valid formula is called a *tautology*.
- An inconsistent formula is called an *inconsistency*.
- $\Sigma \models \Phi$ iff $M(\Sigma) \subseteq M(\Phi)$.
- Φ and Ψ are *equivalent* iff $M(\Phi) = M(\Psi)$.
  - $\Phi \rightarrow \Psi$ is equivalent to $\neg\Phi \vee \Psi$.
  - $\Phi_1 \wedge \cdots \wedge \Phi_n \rightarrow \Psi$ is equivalent to $\neg\Phi_1 \vee \cdots \vee \neg\Phi_n \vee \Psi$.

Conjunctive Normal Form (CNF)

literal: An atomic formula or a negated atomic formula

clause: A disjunction of literals

CNF formula: A conjunction of clauses

- CNF formulas can be represented as sets of sets of literals.
- Empty clause denotes inconsistency.
- Empty CNF formula denotes a tautology.

Conversion to CNF

1. Construct a truth table for Φ.
2. Take every row for which Φ evaluates to false and construct a clause as follows:
   1. If the row maps the atomic formula Ψ to true then include $\neg\Psi$ in the clause.
   2. If the row maps the atomic formula Ψ to false then include Ψ in the clause.

Disadvantage: can require an exponential number of clauses.

Better Conversion to CNF—I

$$\textsc{ConvertCNF}(\Omega, \neg\Phi) \triangleq \begin{pmatrix} (\Omega \vee \Theta) \wedge \\ (\neg\Theta \vee \neg\Omega) \wedge \\ \textsc{ConvertCNF}(\Theta, \Phi) \end{pmatrix}$$

$$\textsc{ConvertCNF}(\Omega, \Phi \wedge \Psi) \triangleq \begin{pmatrix} (\neg\Omega \vee \Theta) \wedge \\ (\neg\Omega \vee \Upsilon) \wedge \\ (\neg\Theta \vee \neg\Upsilon \vee \Omega) \wedge \\ \textsc{ConvertCNF}(\Theta, \Phi) \wedge \\ \textsc{ConvertCNF}(\Upsilon, \Psi) \end{pmatrix}$$

$$\textsc{ConvertCNF}(\Omega, \Phi \vee \Psi) \triangleq \begin{pmatrix} (\Omega \vee \neg\Theta) \wedge \\ (\Omega \vee \neg\Upsilon) \wedge \\ (\Theta \vee \Upsilon \vee \neg\Omega) \wedge \\ \textsc{ConvertCNF}(\Theta, \Phi) \wedge \\ \textsc{ConvertCNF}(\Upsilon, \Psi) \end{pmatrix}$$

Better Conversion to CNF—II

$$\textsc{ConvertCNF}(\Omega, \Phi \rightarrow \Psi) \triangleq \left(\begin{array}{l} (\Omega \vee \Theta) \wedge \\ (\Omega \vee \neg \Upsilon) \wedge \\ (\neg \Theta \vee \Upsilon \vee \neg \Omega) \wedge \\ \textsc{ConvertCNF}(\Theta, \Phi) \wedge \\ \textsc{ConvertCNF}(\Upsilon, \Psi) \end{array} \right)$$

$$\textsc{ConvertCNF}(\Omega, \Phi \leftrightarrow \Psi) \triangleq \left(\begin{array}{l} (\neg \Omega \vee \neg \Theta \vee \Upsilon) \wedge \\ (\neg \Omega \vee \neg \Upsilon \vee \Theta) \wedge \\ (\Omega \vee \neg \Theta \vee \neg \Upsilon) \wedge \\ (\Omega \vee \Theta \vee \Upsilon) \wedge \\ \textsc{ConvertCNF}(\Theta, \Phi) \wedge \\ \textsc{ConvertCNF}(\Upsilon, \Psi) \end{array} \right)$$

Better Conversion to CNF—III

$$\textsc{ConvertCNF}(\neg\Phi) \triangleq \begin{pmatrix} \neg\Theta\wedge \\ \textsc{ConvertCNF}(\Theta,\Phi) \end{pmatrix}$$

$$\textsc{ConvertCNF}(\Phi\wedge\Psi) \triangleq \begin{pmatrix} \Theta\wedge \\ \Upsilon\wedge \\ \textsc{ConvertCNF}(\Theta,\Phi)\wedge \\ \textsc{ConvertCNF}(\Upsilon,\Psi) \end{pmatrix}$$

$$\textsc{ConvertCNF}(\Phi\vee\Psi) \triangleq \begin{pmatrix} (\Theta\vee\Upsilon)\wedge \\ \textsc{ConvertCNF}(\Theta,\Phi)\wedge \\ \textsc{ConvertCNF}(\Upsilon,\Psi) \end{pmatrix}$$

Better Conversion to CNF—IV

$$\textsc{ConvertCNF}(\Phi \rightarrow \Psi) \triangleq \left(\begin{array}{l} (\neg\Theta \vee \Upsilon)\wedge \\ \textsc{ConvertCNF}(\Theta, \Phi)\wedge \\ \textsc{ConvertCNF}(\Upsilon, \Psi) \end{array} \right)$$

$$\textsc{ConvertCNF}(\Phi \leftrightarrow \Psi) \triangleq \left(\begin{array}{l} (\neg\Theta \vee \Upsilon)\wedge \\ (\neg\Upsilon \vee \Theta)\wedge \\ \textsc{ConvertCNF}(\Theta, \Phi)\wedge \\ \textsc{ConvertCNF}(\Upsilon, \Psi) \end{array} \right)$$

Conversion to CNF as Scheme code

```
(define (negate literal)
 (list (not (first literal)) (second literal)))

(define (clauses p atomic-formulas)
 (all-values
  (let ((clause (map (lambda (atomic-formula)
                       (list (a-boolean) atomic-formula))
                     atomic-formulas)))
   (when (apply p (map first clause)) (fail))
   (map negate clause))))
```

Disjunctive Normal Form (DNF)

literal: An atomic formula or a negated atomic formula

minterm: A conjunction of literals

DNF formula: A disjunction of minterms

- DNF formulas can be represented as sets of sets of literals.
- Empty minterm denotes tautology.
- Empty DNF formula denotes an inconsistency.

- Can convert any formula to DNF by analog of the first technique.
- No analog of the second technique exists.

Subsumption and Covering

- Φ *subsumes* Ψ (or Ψ *covers* Φ) if $M(\Phi) \subseteq M(\Psi)$.
- If Φ and Ψ are clauses represented as sets of literals then Φ *subsumes* Ψ iff $\Phi \subseteq \Psi$.
- If Φ and Ψ are minterms represented as sets of literals then Φ *covers* Ψ iff $\Phi \subseteq \Psi$.
- If a CNF formula contains two clauses Φ and Ψ such that Φ subsumes Ψ then can remove Ψ.
- If a DNF formula contains two minterms Φ and Ψ such that Φ covers Ψ then can remove Ψ.
- Can remove tautological clauses that contain Φ and $\neg\Phi$.
- Can remove inconsistent minterms that contain Φ and $\neg\Phi$.

(Prime) Implicates and Implicants

- If Φ is a clause and $\Sigma \models \Phi$ then Φ is an *implicate* of Σ.
- If Φ is an implicate of Σ and no other implicate of Σ subsumes Φ, then Φ is a *prime implicate* of Σ.
- If Φ is a minterm and $\Sigma \models \Phi$, then Φ is an *implicant* of Σ.
- If Φ is an implicant of Σ and no other implicant of Σ covers Φ, then Φ is a *prime implicant* of Σ.
- Given a method for computing the set of all implicates or implicants, one can compute the set of all prime implicates or prime implicants, respectively, simply by removing subsumed clauses or covered minterms.
- Furthermore, if one has the set of all implicates or implicants then one can compute the other set by 'multiplying out,' i.e. distributing $\wedge$ over $\vee$ or vice versa.

Motivation for Resolution—I

$$\frac{\Sigma \models \Phi \qquad \Sigma \models \Phi \rightarrow \Psi}{\Sigma \models \Psi}$$

$$\frac{\Sigma \models \Phi \rightarrow \Psi \qquad \Sigma \models \Psi \rightarrow \Upsilon}{\Sigma \models \Phi \rightarrow \Upsilon}$$

Motivation for Resolution—II

$$\frac{\Sigma \models \Phi_1 \wedge \cdots \wedge \Phi_n \rightarrow \Psi \qquad \Sigma \models \Psi \wedge \Theta_1 \wedge \cdots \wedge \Theta_m \rightarrow \Upsilon}{\Sigma \models \Phi_1 \wedge \cdots \wedge \Phi_n \wedge \Theta_1 \wedge \cdots \wedge \Theta_m \rightarrow \Upsilon}$$

$$\frac{\Sigma \models \neg\Phi_1 \vee \cdots \vee \neg\Phi_n \vee \Psi \qquad \Sigma \models \neg\Psi \vee \neg\Theta_1 \vee \cdots \vee \neg\Theta_m \vee \Upsilon}{\Sigma \models \neg\Phi_1 \vee \cdots \vee \neg\Phi_n \vee \neg\Theta_1 \vee \cdots \vee \neg\Theta_m \vee \Upsilon}$$

Resolution (Robinson 1965)

$$\frac{\Sigma \models \Phi_1 \vee \cdots \vee \Phi_n \vee \Psi \qquad \Sigma \models \neg\Psi \vee \Theta_1 \vee \cdots \vee \Theta_m}{\Sigma \models \Phi_1 \vee \cdots \vee \Phi_n \vee \Theta_1 \vee \cdots \vee \Theta_m}$$

Finding All Prime Implicates

1. Let C be the initial set of clauses.
2. Loop over all $c_1 \in C$.
3. Loop over all $c_2 \in C$.
4. Loop over all literals $l \in c_1$.
5. If c_2 does not contain $\neg l$, continue with the next literal in step 4.
6. Let $c = (c_1 \setminus l) \cup (c_2 \setminus \neg l)$.
7. If c contains some literal l' and its complement, continue with the next literal in step 4.
8. If C already contains some clause that subsumes c, continue with the next literal in step 4.
9. Remove from C any clause subsumed by c.
10. Add c into C.
11. Continue with the next literal in step 4.
12. When finished with the nested loops in steps 2, 3, and 4, if any clause c was added to C by step 10 then repeat step 2. Do this until you finish a complete pass through steps 2, 3, and 4 without adding any new clauses to C.

Conversion to Prime Implicates as Scheme code

```
(define (clauses p atomic-formulas)
 (prime-implicates
  (all-values
   (let ((clause (map (lambda (atomic-formula)
                        (list (a-boolean) atomic-formula))
                      atomic-formulas)))
    (when (apply p (map first clause)) (fail))
    (map negate clause)))))
```