

# Ingredients of Logic

DOMAIN OF DISCOURSE: What we want to talk about.

LANGUAGE: To formulate statements. Assign meanings in terms of domain of discourse.

DEDUCTIVE SYSTEM: For drawing inferences within the language.

## Propositional logic

Propositional logic is a good vehicle to introduce basic properties of logic. It does not provide means to determine the validity (truth or false) of atomic statements. Instead, it allows to evaluate the validity of compound statements given the validity of its atomic components.

For example,

| I like Pat or I like Joe.  
| If I like Pat then I like Joe.  
| Do I like Joe?

| Pigs can fly or fish can sing  
| If pigs can fly then fish can sing.  
| Can fish sing?

We can see that the answer is YES in both cases. The above two sets of statements can be both abstracted as follows:

|  $P \vee Q$   
|  $P \rightarrow Q$   
|  $Q?$

## Syntax

PROPOSITIONAL SYMBOLS. A set **Prop** of some symbols. E.g. p,q,r,...

LOGICAL CONNECTIVES:  $\vee, \wedge, \neg, \rightarrow$ .

PARENTHESES: (,).

EXPRESSION: string of propositional symbols, parenthesis, and logical connectives.

FORMULAS. The set **Form** of formulas is the smallest set of expressions such that:

1. **Prop**  $\subseteq$  **Form**
2. If  $\phi, \psi \in$  **Form** then  $(\phi \wedge \psi), (\phi \vee \psi), (\neg\phi), (\phi \rightarrow \psi) \in$  **Form**

Another way to define formulas is as the language defined by the following CFG (with start symbol **Form**):

**Form**  $\rightarrow$  **Prop**, where **Prop** stands for any propositional symbol,

**Form**  $\rightarrow$  (**Form**  $\circ$  **Form**), where  $\circ \in \{\vee, \wedge, \rightarrow\}$

**Form**  $\rightarrow$  ( $\neg$ **Form**)

**Fact 1** (Unique readability) The above CFG is unambiguous (i.e: each formula it generates has a unique parse tree).

## Semantics

The function of a formula is to create meanings of statements given meanings of atomic statements. The semantics of a formula  $\phi$  with propositional symbols  $p_1, \dots, p_n$  is a mapping associating to each truth assignment  $\theta$  to  $p_1, \dots, p_n$  a truth value (0 or 1) for  $\phi$ . Semantics is well defined due to Fact 1.

The simplest way to specify semantics is via a truth table.

	$p$	$q$	$p \wedge q$
	0	0	0
<b>Ex.</b>	0	1	0
	1	0	0
	1	1	1

**Q.** Can one always find a formula that implements any given semantics? YES, any truth table is realized by a formula. The formula can be found as follows. “Represent” the rows where  $\phi = 1$  by conjuncting the true proposition symbols and negations of the false ones. Finally write the disjunction of the results.

	$p$	$q$	$\phi$	
	0	0	1	$\neg p \wedge \neg q$
<b>Ex.</b>	0	1	0	
	1	0	1	$p \wedge \neg q$
	1	1	0	

$$\phi : (p \wedge \neg q) \vee (\neg p \wedge \neg q)$$

**Corollary 1** Every formula is equivalent to a disjunction of conjunctions of propositional symbols or negation of propositional symbols (DNF).

Dual of DNF is CNF. To get  $\phi$  in CNF:

- Describe cases when  $\phi$  is false. **Ex.**  $(p \wedge q) \vee (\neg p \wedge q)$  - DNF  $\psi$ .

- Note that  $\phi$  is true when  $\neg\psi$  is false. Hence, negate  $\psi$  using DeMorgan's laws. **Ex.**  $(\neg p \vee \neg q) \wedge (p \vee \neg q)$ .

There are cases when DNF (resp. CNF) is exponentially larger than the original formula. **Ex.** For  $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \dots \wedge (x_n \vee y_n)$  the equivalent DNF is exponential in size.

**Q.** Does each truth table have a polynomial size formula implementing it? More precisely, does there exist  $k$  such that every truth table with  $n$  propositional symbols has a form  $\phi$  of size  $\leq n^k$ ? Answer: NO.

**Proof:** Assume there exists such  $k$ . The number of truth tables for  $n$  propositional symbols is  $2^{2^n}$ . The number of formulas of size  $\leq n^k$  is  $\leq (n+6)^{n^k}$  ( $n$  propositional symbols, 4 connectives and parentheses.) Clearly,  $(n+6)^{n^k} < 2^{2^n}$ , for sufficiently large  $n$ . ■

#### BASIC CONCEPTS.

- *Satisfaction.*  
Satisfaction of a formula  $\phi$  by a truth assignment  $\tau$ . Notation:  $\tau \models \phi$  ( $\phi$  is true for  $\tau$ ).
- *Implication.*  
A set of formulas  $\Sigma$  implies  $\phi$ , denoted  $\Sigma \models \phi$ , iff every truth assignment to the propositional symbols in  $\Sigma$  that satisfies  $\Sigma$  also satisfies  $\phi$ . Note:  $\Sigma$  could be an infinite set of formulas using infinitely many propositional symbols.

### Classes of formulas of special interest

- **VALID**- the set of formulas that are always true (tautologies).  
**Ex.**  $(p \vee \neg p), (p \rightarrow p), ((p \vee q) \wedge (p \rightarrow q)) \rightarrow q$  are valid formulas.
- **UNSAT**- the set of formulas that are never true (unsatisfiable).
- In between: **SAT**- the set of formulas for which there exists a satisfying assignment (not unsatisfiable).

Note.  $\phi \in \mathbf{VALID} \iff \neg\phi \in \mathbf{UNSAT}$ .

**Claim 2**  $\Sigma \models \phi \iff (\Sigma \cup \{\neg\phi\}) \in \mathbf{UNSAT}$

**Claim 3** **SAT** is NP-complete.

**Proof:**

- **SAT**  $\in$  **NP**: guess a satisfying assignment, then verify that the formula is true (a satisfying assignment is a certificate).
- **Hardness**. **3COL**(graph 3-coloring)  $\in$  **NP** (there also exists a direct proof). We reduce **3COL** to **SAT**. Let  $G = (V, E)$  be a graph with  $n$  nodes  $\{1, \dots, n\}$ . We use propositional variables  $p_{i,g}, p_{i,r}, p_{i,b}$  to indicate that vertex  $i$  is colored with green, red, or blue. Construct  $\phi$  as follows.

$$\phi : \bigwedge_{i=1}^n ((p_{i,g} \wedge \neg p_{i,r} \wedge \neg p_{i,b}) \vee (p_{i,r} \wedge \neg p_{i,g} \wedge \neg p_{i,b}) \vee (p_{i,b} \wedge \neg p_{i,r} \wedge \neg p_{i,g})) \wedge \bigwedge_{(i,j) \in E} \neg(p_{i,g} \wedge p_{j,g}) \wedge \neg(p_{i,r} \wedge p_{j,r}) \wedge \neg(p_{i,b} \wedge p_{j,b})$$

**Claim 4**  $G \in \mathbf{3COL} \iff \phi \in \mathbf{SAT}$ .

It is also possible to prove that **SAT** is NP-complete directly (cf. the Cook-Levine theorem, e.g. see Sipser) ■

**Claim 5** **VALID**  $\in$  **co-NP**.

**HORN** We now consider a special class of formulas for which SAT is in polynomial time, denoted HORN. The class HORN consists of conjunctions of Horn clauses. A *Horn clause* is a disjunction of literals with at most one positive literal. Example of a formula in HORN:

$$p \wedge (\neg p \vee \neg q \vee s) \wedge q \wedge r \wedge (\neg s \vee \neg r \vee t) \wedge (\neg t)$$

Note that there are two kinds of possible Horn clauses:

1. clause has 1 positive literal

$$p \\ p \vee \neg x_1 \vee \dots \vee \neg x_k : x_1 \wedge \dots \wedge x_k \rightarrow p$$

2. no positive literal

$$\neg x_1 \vee \dots \vee \neg x_k : \neg(x_1 \wedge \dots \wedge x_k) \\ x_1 \wedge \dots \wedge x_k \rightarrow \text{false}$$

Claim: For every finite set  $\Sigma$  of HORN clauses, checking whether  $\Sigma$  is satisfiable is in  $P$

Idea: Let  $\Sigma_1$  be the subset of  $\Sigma$  containing only clauses of type 1 (i.e. one positive literal), and  $\Sigma_2$  the subset of  $\Sigma$  containing clauses of type 2 (i.e no positive literals). Note first that  $\Sigma_1$  is satisfiable. To obtain a minimum satisfying assignment  $\sigma$ , start with literals from single-literal clauses and crank the rules. It now remains to check consistency of  $\sigma$  with the

clauses in  $\Sigma_2$ . To do this, it is enough to check that for each clause  $x_1 \wedge \dots \wedge x_k \rightarrow false$  in  $\Sigma_2$ ,  $\sigma$  is not true for all of  $x_1, \dots, x_k$ .

More concretely stated, to find the minimum satisfying assignment  $\theta$  (ie: the least number of literals that must be set to true to satisfy the clauses) for  $\Sigma$ :

1. Set the propositions in single literal clauses of  $\Sigma_1$  to true (e.g.  $\{p, q, r\}$ )
2. For each rule  $(x_1 \wedge x_2 \dots \wedge x_k) \rightarrow p$  in  $\Sigma_1$ : If  $x_1, x_2 \dots x_k$  are set to true, set  $p$  to true. Repeat this step while there are changes to  $\theta$ .
3. Answer yes iff  $\{x_1, x_2 \dots x_k\} \not\subseteq \theta$  for any  $\neg(x_1 \wedge \dots \wedge x_k)$  in  $\Sigma_2$ .

**Example** Consider the set  $\Sigma$  of Horn clauses:

$$\begin{aligned} p \\ q \\ r \\ \neg p \vee \neg q \vee s \\ \neg s \vee \neg r \vee t \\ \neg t \end{aligned}$$

The set  $\Sigma_1$  of clauses of type 1 consists of the first 5 clauses, and  $\Sigma_2$  consists of the last clause. Note that  $\Sigma_1$  can also be written as:

$$\begin{aligned} p \\ q \\ r \\ p \wedge q \rightarrow s \\ s \wedge r \rightarrow t \end{aligned}$$

The minimum satisfying assignment for  $\Sigma_1$  is obtained as follows:

1. start with  $\{p, q, r\}$
2. use the first implication to infer  $s$
3. use the second implication to infer  $t$

Thus, the minimum satisfying assignment makes  $\{p, q, r, s, t\}$  true. This contradicts  $\Sigma_2$ , which states that  $t$  must be false. Thus,  $\Sigma$  is not satisfiable.

# Compactness Theorem

The Compactness Theorem allows to reduce satisfiability of infinite sets of formulas to satisfiability of finite sets. It is most useful in the context of first-order logic. Let  $\Sigma$  be an infinite set of formulas over Prop. We say that  $\Sigma$  is *finitely satisfiable* iff every finite subset  $\Sigma_0 \subseteq \Sigma$  is satisfiable.

**Theorem:** An infinite set  $\Sigma$  of propositional formulas is satisfiable iff it is finitely satisfiable.

In order to prove the Compactness Theorem, we begin with some preliminary definitions and observations.

**Definition 6** Consider a truth assignment  $\tau$  for Prop. The Theory of  $\tau$ , denoted  $Th(\tau)$ , is the set of formulas over Prop satisfied by  $\tau$ , that is,  $\{\phi \mid \tau \models \phi\}$ .

**Definition 7** A set  $\Delta$  of formulas is *complete* iff for every formula  $\phi$ , either  $\phi \in \Delta$  or  $\neg\phi \in \Delta$ .

- Note:  $Th(\tau)$  is complete. Indeed, for every formula  $\phi$ ,  $\tau \models \phi$  or  $\tau \models \neg\phi$ , so  $\phi \in Th(\tau)$ , or  $\neg\phi \in Th(\tau)$
- Note: Suppose  $\Delta$  is complete and satisfiable. Then there is a unique truth assignment satisfying  $\Delta$ :  $\tau_\Delta(p)$  is true iff  $p \in \Delta$ .

**Lemma** If  $\Delta$  is a complete and finitely satisfiable set of formulas, then  $\tau_\Delta \models \phi$  iff  $\phi \in \Delta$ .

**Proof:** The proof is by structural induction on  $\phi$ . We can assume without loss of generality that  $\phi$  uses only  $\neg, \wedge$ , since this is a functionally complete set of operators.

- **Base Case:** If  $\phi = p$ ,  $\tau_\Delta(p)$  is true iff  $p \in \Delta$  (by definition).
- **Induction:**
  - If  $\phi = \neg\theta$ , then  $\tau_\Delta \models \phi$  iff  $\tau_\Delta \not\models \theta$  iff  $\theta \notin \Delta$  (by induction hypothesis) iff  $\neg\theta \in \Delta$  (by completeness).
  - If  $\phi = \theta_1 \wedge \theta_2$ , then  $\tau_\Delta \models \phi$  iff  $\tau_\Delta \models \theta_1$  and  $\tau_\Delta \models \theta_2$ . By induction, this holds iff  $\theta_1 \in \Delta$  and  $\theta_2 \in \Delta$ . We show that  $(\theta_1 \wedge \theta_2) \in \Delta$  iff  $\theta_1 \in \Delta$  and  $\theta_2 \in \Delta$ :
    - Only if: the proof is by contradiction. Suppose  $(\theta_1 \wedge \theta_2) \in \Delta$  but  $\theta_1 \notin \Delta$ . By completeness,  $\neg\theta_1 \in \Delta$ . Then  $\{(\theta_1 \wedge \theta_2), \neg\theta_1\} \in \Delta$ . This is unsatisfiable, contradicting the finite satisfiability of  $\Delta$ . Similarly for  $\theta_2$ .
    - If: Suppose  $\theta_1 \in \Delta$  and  $\theta_2 \in \Delta$ . We show, again by contradiction, that  $(\theta_1 \wedge \theta_2) \in \Delta$ . Suppose  $(\theta_1 \wedge \theta_2) \notin \Delta$ . Then by completeness,  $\neg(\theta_1 \wedge \theta_2) \in \Delta$ , so  $\{\neg(\theta_1 \wedge \theta_2), \theta_1, \theta_2\} \in \Delta$ . This is unsatisfiable, contradicting the finite satisfiability of  $\Delta$ . Thus,  $(\theta_1 \wedge \theta_2) \in \Delta$ .

■

By the above lemma, if  $\Delta$  is complete and finitely satisfiable, then  $\tau_\Delta \models \Delta$ , so  $\Delta$  is satisfiable. Therefore, to prove that  $\Sigma$  is satisfiable, it is enough to show that  $\Sigma$  is contained in a complete, finitely satisfiable set of formulas  $\Delta$ .

We construct  $\Delta$  as follows: first, enumerate all formulas  $\phi_1, \phi_2, \dots$ . We define by induction a sequence of formulas  $\{\Delta_i\}_{i \geq 0}$ , where  $\Delta_i$  contains for every  $j \leq i$  either  $\phi_j$  or  $\neg\phi_j$  and is finitely satisfiable:

- $\Delta_0 = \Sigma$ . Then  $\Delta_0$  is finitely satisfiable.
- Inductively, assume that  $\Delta_i$  is finitely satisfiable and contains  $\Sigma$ . It must be the case that either  $\Delta_i \cup \{\phi_{i+1}\}$  or  $\Delta_i \cup \{\neg\phi_{i+1}\}$  are finitely satisfiable. Suppose not. Then there exist finite subsets  $X$  and  $Y$  of  $\Delta_i$  such that  $X \cup \{\phi_{i+1}\}$  and  $Y \cup \{\neg\phi_{i+1}\}$  are not satisfiable. But  $X \cup Y \subseteq \Delta_i$  is satisfiable by some  $\tau$ , and  $\tau$  must satisfy either  $\phi_{i+1}$  or  $\neg\phi_{i+1}$ , so  $X \cup \{\phi_{i+1}\}$  or  $Y \cup \{\neg\phi_{i+1}\}$  is satisfiable, and we have arrived at a contradiction. Consequently, if  $\Delta_i \cup \{\phi_{i+1}\}$  is finitely satisfiable, then let  $\Delta_{i+1} = \Delta_i \cup \{\phi_{i+1}\}$ . If  $\Delta_i \cup \{\neg\phi_{i+1}\}$  is finitely satisfiable, then  $\Delta_{i+1} = \Delta_i \cup \{\neg\phi_{i+1}\}$ .

Now let  $\Delta = \bigcup_{i=0}^{\infty} \Delta_i$ .  $\Delta$  is complete, since every formula  $\phi_i$  or its negation  $\neg\phi_i$  is in  $\Delta$ . Lastly, we must show that  $\Delta$  is finitely satisfiable. Suppose not. Then there is a finite subset  $X \subseteq \Delta$  that is not satisfiable. Since  $X$  is finite, there must be some  $m$  such that  $X \subseteq \Delta_m$ , but this contradicts the finite satisfiability of  $\Delta_m$ , which was proven above.

**Example 1** A simple application of the compactness theorem: prove that an infinite, countable graph is 3-colorable iff every finite subgraph is 3-colorable.

## Deductive Systems for Propositional Logic

### Deductive Systems

Informally, a deductive system is a mechanism for proving new statements from given statements. Let  $\Sigma$  be a set of given statements (propositional formulas). In a deductive system, there are two components: inference rules and proofs.

1. **Inference rule:** An inference rule indicates that if certain set of statements (formulas)  $\varphi_1, \dots, \varphi_k$  is true, then a given statement  $\varphi$  must be true. An inference rule  $H$  is denoted as

$$H: \frac{\varphi_1, \dots, \varphi_k}{\varphi}.$$

Example (modus ponens):  $\frac{p, p \rightarrow q}{q}$

2. **Proofs:** A proof of  $\varphi$  from  $\Sigma$  is sequence of formulas  $\varphi_1, \dots, \varphi_n$  such that  $\varphi_n = \varphi$  and for all  $i \leq n$

- Each formula  $\varphi_i \in \Sigma$ , or
- There are a subset of formulas  $\varphi_{i_1}, \dots, \varphi_{i_k}$   $i_1, \dots, i_k < i$ , such that,

$$\frac{\varphi_{i_1}, \dots, \varphi_{i_k}}{\varphi_i}$$

is an inference rule.

If  $\varphi$  has a proof from  $\Sigma$  using inference rule  $H$  we write  $\Sigma \vdash_H \varphi$

PROPERTIES:

- Soundness: If  $\Sigma \vdash_H \varphi$  then  $\Sigma \models \varphi$  (All sentences provable from  $\Sigma$  are implied by  $\Sigma$ .) This property is fundamental for the correctness of the deductive system.
- Completeness: If  $\Sigma \models \varphi$  then  $\Sigma \vdash_H \varphi$  (All sentences implied by  $\Sigma$  are provable.) This is a desirable property in deductive systems.

## Hilbert's System

Hilbert's System is an example of a sound and complete deductive system for proving valid statements starting from a given set of three tautologies ("axioms"). Digression: the system uses only  $\neg, \rightarrow$ . To see that these operators are functionally complete, note that we can express  $\wedge$  in the following way:  $p \wedge q \equiv \neg(p \rightarrow \neg q)$ .

### Axioms

- A1 (Simplification):  $(p \rightarrow (q \rightarrow p))$
- A2 (Frege's Axiom):  $((p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r)))$
- A3 (Reductio Ad Absurdum):  $((\neg p \rightarrow q) \rightarrow ((\neg p \rightarrow \neg q) \rightarrow p))$

### Inference Rules

- Modus Ponens:  $\frac{\phi, \phi \rightarrow \psi}{\psi}$
- Substitution:  $\frac{\phi}{\phi[\theta]}$ , where  $\theta$  substitutes an arbitrary formula to a propositional symbol in  $\phi$ .

Note the following subtlety about the substitution rule: in general,  $\phi \rightarrow \phi[\theta]$  is *not a tautology*. (Example:  $\phi = p$  and  $\theta(p) = p \wedge \neg p$ .) However, if  $\phi$  is valid, then  $\phi[\theta]$  is also valid. This is good enough, since the deductive system is only used to prove validity of formulas. The following can be shown:

**Theorem 8** Let  $H$  denote Hilbert's System. For every formula  $\varphi$ ,  $\{A1, A2, A3\} \models \varphi$  iff  $\{A1, A2, A3\} \vdash_H \varphi$ .

Since A1, A2, and A3 are tautologies, this makes Hilbert's System sound and complete for proving valid propositional formulas.

## Natural Deduction

Natural deduction is a collection of inference rules and proof schemas, capturing aspects of reasoning in an appealing way.

Let  $\perp$  denote contradiction, falsity. The following are the inference rules of natural deduction:

$\frac{\varphi, \psi}{\varphi \wedge \psi} \quad (1)$		$\left. \begin{array}{c} \varphi \\ \vdots \\ \psi \end{array} \right\} \quad (13)$
$\frac{\varphi \wedge \psi}{\varphi} \quad (2)$		$\frac{\psi}{\varphi \rightarrow \psi} \quad (14)$
$\frac{\varphi \wedge \psi}{\psi} \quad (3)$		$\left. \begin{array}{c} \varphi \\ \vdots \\ \perp \end{array} \right\} \quad (15)$
$\frac{\varphi, \varphi \rightarrow \psi}{\psi} \quad (4)$		$\frac{\perp}{\neg \varphi} \quad (16)$
$\frac{\varphi, \neg \varphi}{\perp} \quad (5)$		$\left. \begin{array}{c} \neg \varphi \\ \vdots \\ \perp \end{array} \right\} \quad (17)$
$\frac{\perp}{\neg \neg \varphi} \quad (6)$		$\frac{\perp}{\varphi} \quad (18)$
$\frac{\perp}{\varphi} \quad (7)$		$\left. \begin{array}{c} \varphi \vee \psi \quad \varphi \quad \psi \\ \vdots \quad \vdots \\ \rho \quad \rho \end{array} \right\} \quad (19)$
$\frac{\varphi \rightarrow \psi, \psi \rightarrow \varphi}{\varphi \leftrightarrow \psi} \quad (8)$		$\frac{\rho}{\rho} \quad (20)$
$\frac{\varphi \leftrightarrow \psi}{\varphi \rightarrow \psi} \quad (9)$		
$\frac{\varphi \leftrightarrow \psi}{\psi \rightarrow \varphi} \quad (10)$		
$\frac{\varphi}{\varphi \vee \psi} \quad (11)$		
$\frac{\psi}{\varphi \vee \psi} \quad (12)$		

Rules (13) allows us to prove valid statements of the form “If  $\varphi$  then  $\psi$ ” even if we don't know the truth value of the  $\varphi$  statement (ie.  $\varphi$  is not in the set  $\Sigma$  of known valid statements).

Indeed, for this rule, we start *assuming*  $\varphi$  is valid. If we can conclude  $\psi$  is valid in a world where  $\Sigma \cup \varphi$  are valid, then we conclude that the relation  $\varphi \rightarrow \psi$  is true, and we *release* the assumption  $\varphi$  is valid. Similarly, rules (14) -(16) describe *proof schemas* using assumptions that are released once the proof (a sub-proof of the overall proof) is completed. Applications of rules (13)-(16) may be nested, in which case each rule is applied in the context of a current set of assumptions (the obvious scoping rules apply). In a proof, each assumption must be eventually released (this is analogous to well-formed parenthesis). The detailed definition is a bit tedious but straightforward.

We now show how to apply the above inference rules.

**Example 2** One of De Morgan's Laws

$$\neg(\varphi \vee \psi) \leftrightarrow (\neg\varphi \wedge \neg\psi)$$

**Proof:** By rule (8) if we can prove  $\neg(\varphi \vee \psi) \rightarrow (\neg\varphi \wedge \neg\psi)$  and  $(\neg\varphi \wedge \neg\psi) \rightarrow \neg(\varphi \vee \psi)$  we can infer the desired result.

To prove the first direction, we use rule 13 and assume the hypothesis  $\neg(\varphi \vee \psi)$ . Then

$\neg(\varphi \vee \psi)$		// Assumption 1
	$\varphi$	// Assumption 1.1
	$\varphi \vee \psi$	// by rule 13
	$\perp$	// by rule 5
$\neg\varphi$		// by rule 14; release Assumption 1.1
	$\psi$	// Assumption 1.2
	$\varphi \vee \psi$	// by rule 13
	$\perp$	// by rule 5
$\neg\psi$		// by rule 14, release Assumption 1.2
$\neg\varphi \wedge \neg\psi$		// by rule 1
$\neg(\varphi \vee \psi) \rightarrow (\neg\varphi \wedge \neg\psi)$		// by rule 13, release Assumption 1

We now prove the second direction.

$\neg\varphi \wedge \neg\psi$		// Assumption 1
$\neg\varphi$		// by rule 2
$\neg\psi$		// by rule 3
	$\varphi \vee \psi$	// Assumption 1.1
	$\varphi \quad \psi$	// Assumptions 1.1.1
	$\perp \quad \perp$	// by rule 5
	$\perp$	// by rule 16, release Assumptions 1.1.1
$\neg(\varphi \vee \psi)$		// by rule 14, release Assumption 1.1
$(\neg\varphi \wedge \neg\psi) \rightarrow \neg(\varphi \vee \psi)$		// by rule 13, release Assumption 1

■

**Example 3** Prove Pierce's Law:  $((A \rightarrow B) \rightarrow A) \rightarrow A$ .

**Proof:**

$(A \rightarrow B) \rightarrow A$			// Assumption 1
	$\neg A$		// Assumption 1.1
		$A$	// Assumption 1.1.1
		$\perp$	// by rule 5
		$B$	// by rule 7
	$A \rightarrow B$		// by rule 13, release Assumption 1.1.1
	$A$		// by Assumption [1] and rule 4
	$\perp$		// by rule 5
$A$			// by rule 14, release Assumption 1.1
$((A \rightarrow B) \rightarrow A) \rightarrow A$			// by rule 13, release Assumption 1

■

**Fact 2** Natural deduction is sound.

To show that natural deduction is also complete we need to introduce propositional resolution.

## Propositional Resolution

Resolution is another procedure for checking validity of formulas indirectly, by showing unsatisfiability of their negation. A *formula* in this approach is a conjunction of clauses, where a *clause* is a disjunction of literals, described in detail below. The core of this method is a single *resolution* rule also described below.

### TERMINOLOGY

- **Clause:** A clause is a propositional formula composed by disjunction of literals. For example  $p \vee q \vee \neg r$ . It is usually denoted as the set of literals, eg.  $\{p, q, \neg r\}$ .

The empty clause, denoted as  $\square$ , is the disjunction of no literals. It is always false.

- **Formula:** A set of clauses, each of them satisfiable. For example,  $\{\{p, \neg q\}, \{r\}, \{\neg r, s\}\}$  represents the CNF formula  $(p \vee \neg q) \wedge (r) \wedge (\neg r \vee s)$ . Thus, intuitively we would like to find if there is a truth assignment such that atleast one literal is true in every clause of the formula.

The empty formula, denoted as  $\emptyset$ , is the set that contains no clauses. It is always true.

A proof system for resolution contains a single *resolution rule*, which, given clauses  $C$  (containing some literal  $y$ ) and  $C'$  (containing some literal  $\neg y$ ), allows to infer a new clause, called the *resolvent*  $res_y(C, C')$  of  $C$  and  $C'$  (with respect to  $y$ ). The resolvent is defined as follows.

$$res_y(C, C') = (C - \{y\}) \cup (C' - \{\neg y\}). \quad (17)$$

We denote by  $Res(\varphi)$  the smallest set of clauses containing  $\varphi$  and closed under resolvent. In other words,  $Res(\varphi)$  consists of all clauses that can be obtained from  $\varphi$  by repeated applications of the resolvent operator.

**Example 4** If  $C = \{p, y\}$  and  $C' = \{q, \neg y\}$ , then  $res_y(C, C') = \{p, q\}$ .

It is possible to show that the resolution rule so defined indeed computes a clause that can be inferred using natural deduction.

**Claim 9** Let  $C$  and  $C'$  be any two clauses such that  $y \in C$  and  $\neg y \in C'$ . Then  $C \wedge C' \rightarrow res_y(C, C')$ .

In order to prove the validity of a formulas  $\psi$ , we will prove the *negated* formula  $\neg\psi$  is unsatisfiable. To prove unsatisfiability of a formula  $\varphi$ , we need to define the *resolution refutation* of the formula  $\varphi$ .

**Definition 10** The resolution refutation tree of the formula  $\varphi$  is a tree rooted at the empty clause, where every leaf is a clause in  $\varphi$  and each internal node is computed as the resolvent of the two corresponding children.

Notice that clauses of  $\varphi$  can appear repeated as leaves. From Claim 9 we can conclude that

**Claim 11** If there exists a resolution refutation tree for formula  $\varphi$ , then  $\varphi \rightarrow \square$ , that is,  $\varphi$  is unsatisfiable.

**Example 5** A resolution refutation tree for  $\varphi = (p \vee q) \wedge (\neg q \vee r) \wedge (\neg r) \wedge (\neg p \vee \neg s) \wedge (s \vee \neg t) \wedge (t)$  is shown in Figure 1.

The order in which clauses are selected to compute the resolvent matters when computing the resolution refutation tree, as the following example shows.

**Example 6** Figure 2 shows an unsuccessful attempt to obtain a resolution refutation tree for the formula  $\psi = (p \vee q) \wedge (\neg q \vee r) \wedge (\neg p) \wedge (\neg q)$ . Such tree exists, as it is shown in Figure 3.

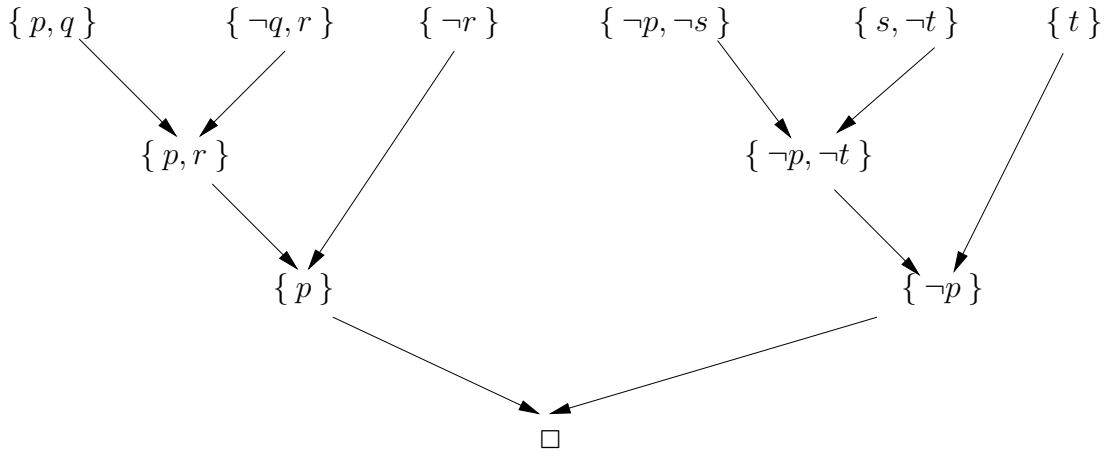


Figure 1: A resolution refutation tree for  $\varphi$ .

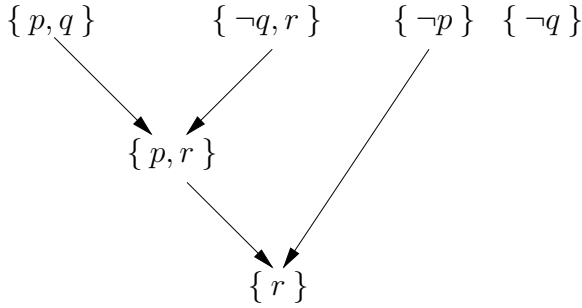


Figure 2: Unsuccessful attempt of resolution refutation tree for  $\psi$ .

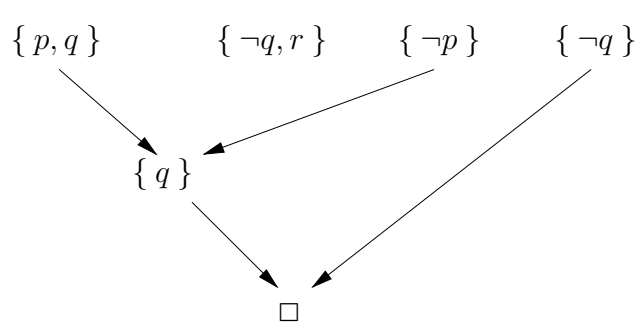


Figure 3: A resolution refutation tree for  $\psi$ .

## Properties of Propositional Resolution

### Soundness

Propositional resolution is sound, that is, if there exists a resolution refutation tree for a given formula  $\varphi$ , then  $\varphi$  must be unsatisfiable.

**Theorem 12** For any formula  $\varphi$ , if  $\square \in Res(\varphi)$ , then  $\varphi \rightarrow \square$  (so  $\varphi$  is unsatisfiable).

### Completeness

Propositional resolution is complete, that is, if a given formula  $\varphi$  is unsatisfiable, then  $\varphi$  has a resolution refutation tree.

**Theorem 13** For any formula  $\varphi$ , if  $\varphi$  is unsatisfiable, then  $\square \in Res(\varphi)$ .

**Proof:** We will prove a counter positive argument i.e. if  $\Box \notin Res(\varphi)$  then  $\varphi$  is satisfiable. We prove this by induction on the number of variables in  $\varphi$ .

Basis: We have one variable, say  $p$ . All possible clauses of  $\varphi$  are either  $\{p\}$  and/or  $\{\neg p\}$ . If  $\varphi$  is unsatisfiable then both clauses occur, and therefore  $\Box \in Res(\varphi)$ .

Induction step: Suppose the hypothesis is true for formulas with fewer than  $n$  variables ( $n > 1$ ). Let  $\varphi$  be a formula with  $n$  variables. Suppose  $\Box \notin Res(\varphi)$ , we will show  $\varphi$  is satisfiable. Let  $p$  be a variable of  $\varphi$ . Then either  $\{p\} \notin Res(\varphi)$  or  $\{\neg p\} \notin Res(\varphi)$  (if both hold then  $\Box \in Res(\varphi)$  immediately). First we assume  $\{\neg p\} \notin Res(\varphi)$ . We define the formula  $\varphi^p$  as containing all clauses that do not contain  $\{p\}$  and where the literal  $\neg p$  has been removed from each clause (in other words,  $\varphi^p$  is the formula resulting from setting  $p$  to true). Formally,

$$\varphi^p = \{ C - \{ \neg p \} : C \in \varphi, p \notin C \} \quad (18)$$

Intuitively this denotes the formula that we have to satisfy assuming  $p$  is set to true. By removing  $p$  we can apply induction. First, notice that

$$(\dagger) Res(\varphi^p) \subseteq \{ C - \{ \neg p \} : C \in Res(\varphi), p \notin C \}$$

Indeed, this follows from the fact that  $\varphi^p \subseteq \{ C - \{ \neg p \} : C \in Res(\varphi), p \notin C \}$ , and the easily checked fact that  $\{ C - \{ \neg p \} : C \in Res(\varphi), p \notin C \}$  is closed under resolvent. From  $(\dagger)$ , the assumption that  $\Box \notin Res(\varphi)$ , and the assumption that  $\{\neg p\} \notin Res(\varphi)$ , it follows that  $\Box \notin Res(\varphi^p)$ . By the induction hypothesis,  $\varphi^p$  is satisfiable. Then  $\varphi$  is satisfiable by an extension of the satisfying assignment of  $\varphi^p$  with  $p$  equal true. The case  $\{p\} \notin Res(\varphi)$  is analogous. ■

## Completeness of Natural Deduction

**Theorem 14** Let  $H$  be the set of inference rules of Natural Deduction. If  $\Sigma \models \varphi$  then  $\Sigma \vdash_H \varphi$ .

The idea behind the proof of completeness of natural deduction is as follows. Suppose  $\varphi$  is valid (then  $\neg\varphi$  is unsatisfiable). We mimic in natural deduction a resolution refutation for  $\neg\varphi$ , building a proof of  $\perp$  from  $\neg\varphi$ . We then apply the contradiction rule 15

$$\frac{\begin{array}{c} \neg\varphi \\ \vdots \\ \perp \end{array}}{\varphi}$$

to infer validity of  $\varphi$ .

**Proof: (Sketch)** Given a formula  $\varphi$  valid under  $\Sigma$ , we perform the following steps:

1. Prove that  $\neg\varphi$  is equivalent to some  $\psi$ , where  $\psi$  is in CNF.
2. Prove that  $\psi \rightarrow Res(\psi)$ , for all  $\psi$ .
3. By completeness of resolution, if  $\psi$  is unsatisfiable then  $\square \in Res(\psi)$ . Therefore,  $\{p\}$  and  $\{\neg p\} \in Res(\psi)$  for some literal  $p$ . This implies  $Res(\psi) \rightarrow \perp$ .
4. Conclude that  $\neg\varphi \rightarrow \perp$  and therefore  $\varphi$  is valid.

Step (1) can be easily done by repeated application of De Morgan's laws. Step (2) can be proven using natural deduction. Finally, step (3) can be proven by induction on the number of steps to obtain  $Res(\psi)$ . Clearly, each step can be simulated using natural deduction. ■

It is very likely that any algorithm for propositional resolution will take very long on the worst case (recall that checking validity of a formula  $\varphi$  is co-NP complete).

## Linear Resolution and PROLOG

Linear resolution is a particular resolution strategy that always resolves the most recent resolvent with some clause which is not used again. The resolution refutation tree so obtained is therefore linear. It is possible to prove that, if the set of clauses are Horn clauses, there exists a linear resolution strategy for any formula. This is, linear resolution is complete for the set of Horn clauses.

The language PROLOG uses resolution on a set of Horn clauses. Each clause is called a *program clause*. Moreover, clauses composed by a single literal are called *facts*. A clause with a single negated literal is called a *query*. Figure 4 shows a comparison of the different notations. In PROLOG, to query a statement  $t$ , the idea is to negate the statement ( $\neg t$ ) and to perform resolution with the set of known true statements. If a resolution refutation tree is found, the statement  $t$  is implied by the program.

Clausal Form	Conventional Syntax	PROLOG	
		Syntax	Terminology
$\{p\}$	$p$	$p.$	<div style="display: flex; align-items: center; justify-content: center;"> <div style="margin-right: 10px;"> <math>\left. \begin{array}{l} \} \\ \} \\ \} \end{array} \right\}</math> </div> <div>           facts         </div> <div style="margin-left: 10px;"> <math>\left. \begin{array}{l} \} \\ \} \\ \} \end{array} \right\}</math> </div> <div>           program clauses         </div> </div>
$\{q\}$	$q$	$q.$	
$\{r\}$	$r$	$r.$	
$\{s, \neg p, \neg q\}$	$p \wedge q \rightarrow s$	$s:-p,q.$	
$\{t, \neg s, \neg r\}$	$s \wedge r \rightarrow t$	$t:-s,r.$	
$\{\neg t\}$	$t \rightarrow \perp$	$?-t.$	

Figure 4: Compared PROLOG syntax

**Example 7** An example of linear resolution for the formula  $\phi = (p) \wedge (q) \wedge (r) \wedge (t \vee \neg s \wedge \neg r) \wedge (s \vee \neg p \vee \neg q) \wedge (\neg t)$  is shown in Figure 5.

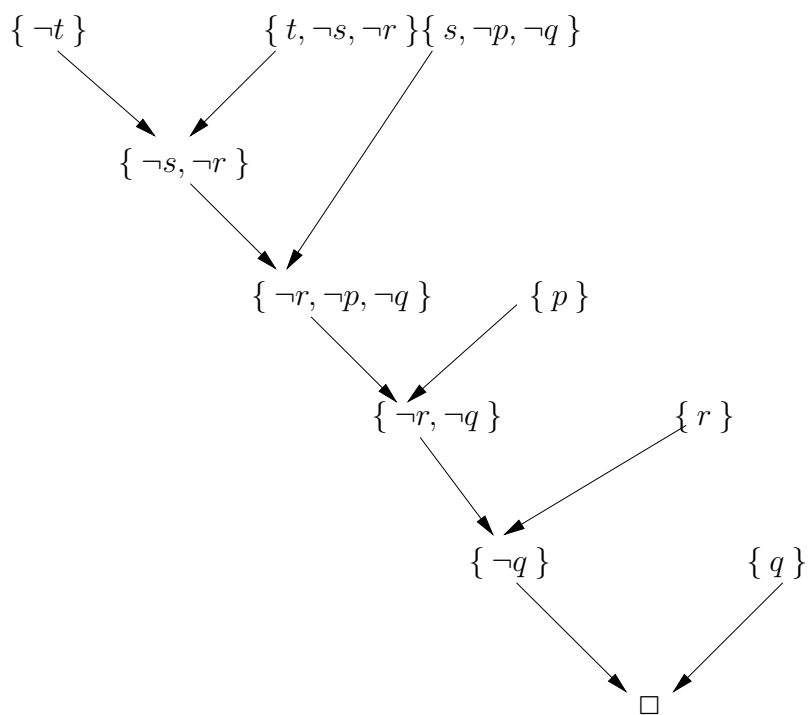


Figure 5: Linear resolution tree for  $\varphi$ .