

Lecture 11: Theorem Proving 1

10:30 AM, Mar 3, 2009

Contents

1 Overview	1
2 Modus Ponens	1
2.1 Horn Databases	2
2.2 Backward Chaining	3
2.3 Forward Chaining	3
3 Resolution	4
3.1 Normal Form	6
3.2 Proof-by-Refutation	7
3.3 Resolution Strategies	9

1 Overview

In this lecture, we describe two proof-theoretic engines, one based on modus ponens (which dates back to Aristotle), and the other on resolution, which dates back to the work of Robinson (a philosopher, mathematician, and computer scientist) in 1965. Iterated modus ponens is closely related to iterated resolution, and hence the distinction is sometimes blurred. Following Ginsburg's 1993 AI text, we present resolution as a generalization of modus ponens. More specifically, resolution is applicable to knowledge bases in normal form, and all knowledge bases can be converted to normal form, whereas modus ponens is applicable only to knowledge bases consisting of Horn clauses. Horn databases, while natural, are not sufficient to express all sentences of propositional logic.

2 Modus Ponens

Modus ponens, which captures one of Aristotle's syllogisms, generalizes the ($\rightarrow E$) rule of natural deduction.

MODUS PONENS

$$\frac{A_1 \wedge \dots \wedge A_m \rightarrow B \quad A_i}{A_1 \wedge \dots \wedge A_{i-1} \wedge A_{i+1} \wedge \dots \wedge A_m \rightarrow B} \text{ (MP)}$$

Exercise: Derive MP using ND.

Solution: (Sketch)

$$\frac{\frac{A \wedge C \rightarrow B \quad \frac{[A] \quad C}{A \wedge C} (\wedge I)}{B} (\rightarrow I)}{A \rightarrow B} (\rightarrow I)$$

ITERATED MODUS PONENS

$$\frac{A_1 \wedge \dots \wedge A_m \rightarrow B \quad C_1 \wedge \dots \wedge C_n \rightarrow A_i}{A_1 \wedge \dots \wedge A_{i-1} \wedge C_1 \wedge \dots \wedge C_n \wedge A_{i+1} \wedge \dots \wedge A_m \rightarrow B} (\text{IMP})$$

Exercise: Derive IMP using ND.

Solution: (Sketch)

$$\frac{\frac{(A_1 \wedge C) \rightarrow B \quad \frac{[A_1 \wedge A_2 \wedge A_3]}{A_1} (\wedge E) \quad \frac{\frac{[A_1 \wedge A_2 \wedge A_3]}{A_2 \wedge A_3} (\wedge E) \quad A_2 \wedge A_3 \rightarrow C}{C} (\wedge I)}{A_1 \wedge C} (\rightarrow E)}{B} (\rightarrow I)}{A_1 \wedge A_2 \wedge A_3 \rightarrow B} (\rightarrow I)$$

Theorem: IMP is sound.

Proof: Suppose not: *i.e.*, suppose that there exists an interpretation which satisfies the premises of IMP but does not satisfy the conclusion. In such an interpretation, the antecedent of the conclusion is satisfied, while the consequent is not; in particular, B is not satisfied. Since the antecedent is satisfied, however, (i) $C_1 \wedge \dots \wedge C_m$ is satisfied, and (ii) $A_1 \wedge \dots \wedge A_{i-1} \wedge A_{i+1} \wedge \dots \wedge A_n$ is satisfied. Now, by (i) and the fact that the second premise is satisfied, A_i must be satisfied. Together with (ii), this implies that B is satisfied. Contradiction. Therefore, IMP is sound. \diamond

2.1 Horn Databases

A *Horn clause* is a complex formula of the form $A_1 \wedge \dots \wedge A_n \rightarrow B$ in which the A_i 's and B are atomic. Atomic formulas and their negations are called *literals*; thus, Horn clauses are equivalently defined as disjunctions of literals with exactly one positive literal (*e.g.*, $\neg A_1 \vee \dots \vee \neg A_n \vee B$).

A *Horn database* is a knowledge base of Horn clauses. IMP is complete for Horn databases, but IMP is *not* complete for all knowledge bases. For example, $\text{KB} \models R$, but $\text{KB} \not\models_{\text{IMP}} R$, given $\text{KB} = \{C, C \rightarrow T \vee H, H \rightarrow R, T \rightarrow R \vee S, S \rightarrow R\}$.

Theorem provers that implement modus ponens proceed either by backward or forward chaining. Backward chaining involves reasoning backwards from the goal, while forward chaining is data-driven: new data drives new conclusions.

We demonstrate backward and forward chaining on the following Horn database: $\{P \wedge Q \wedge R \rightarrow S, T \wedge U \rightarrow P, V \rightarrow Q, R, T, U, V\}$. The complex formulas in this knowledge base are called *Horn rules*, and the literals are called *facts*.

2.2 Backward Chaining

Backward chaining is the repeated application of modus ponens to Horn databases. The inputs to backward chaining are a knowledge base KB of Horn clauses and set of goals Σ in some logical language (*e.g.*, propositional logic). If $\text{KB} \models \Sigma$, then backward chaining terminates with **true**. If $\text{KB} \not\models \Sigma$, then backward chaining terminates with **false**.

The backward chaining algorithm is inspired by the following observation: given the Horn database KB with select rule $A_1 \wedge \dots \wedge A_n \rightarrow A$ and given goal $A \in \Sigma$, if $\text{KB} \vdash A_1, \dots, A_n$, then $\text{KB} \vdash A$, by modus ponens. The main idea of backward chaining, as its name suggests, is to reduce the goal A to n subgoals A_1, \dots, A_n , and to proceed to further reduce these subgoals to facts.

A call to backward chaining, with goal S , would operate as follows on the sample Horn database given above:

```

 $\Sigma = \{S\}$ 
 $\Sigma = \{P, Q, R\}$ 
 $\Sigma = \{T, U, Q, R\}$ 
 $\Sigma = \{U, Q, R\}$ 
 $\Sigma = \{Q, R\}$ 
 $\Sigma = \{V, R\}$ 
 $\Sigma = \{R\}$ 
 $\Sigma = \emptyset$ , return true

```

If we were to add to our database the Horn rule $O \rightarrow S$, then backward chaining might instead begin like this

```

 $\Sigma = \{S\}$ 
 $\Sigma = \{O\}$ , return false

```

before proceeding as above. In other words, backward chaining is a backtracking depth-first search algorithm.

The implementation of backward chaining depicted in Table 1 relies on two mutually recursive subroutines. `BC_GOALS(KB, Σ)` determines if all of the goals in Σ are provable. To determine this, it calls `BC_RULES` on each such goal. `BC_RULES(KB, A)` determines if the goal A is provable. To determine this, it calls `BC_GOALS` on the antecedents of all rules in KB with A as their consequent.

2.3 Forward Chaining

Given a knowledge base KB, forward chaining derives all the conclusions implied by the knowledge base. This form of reasoning is useful in medical diagnosis, where it is important to consider all possible implications. But most often, forward chaining is less practical than backward chaining, since it derives many irrelevant conclusions.

The forward chaining algorithm is initialized with a knowledge base and a set of facts to which it repeatedly applies modus ponens in the forward direction: *i.e.*, if the rule $B_1 \wedge \dots \wedge B_n \rightarrow C \in \text{KB}$ and the fact $B_i \in \text{KB}$, then the rule $B_1 \wedge \dots \wedge B_{i-1} \wedge B_{i+1} \wedge \dots \wedge B_n \rightarrow C$ is added to KB. The pseudocode for forward chaining in propositional logic appears in Table 2.

Forward chaining operates as follows on the sample Horn database given above:

BC_GOALS(KB, Σ)	
Inputs	a set of Horn rules KB a set of sub/goals Σ
Outputs	if $\text{KB} \vdash \Sigma$ (<i>i.e.</i> , $\text{KB} \models \Sigma$), return true if $\text{KB} \not\vdash \Sigma$ (<i>i.e.</i> , $\text{KB} \not\models \Sigma$), return false
<ol style="list-style-type: none"> 1. let $x = \text{true}$ 2. for all $A \in \Sigma$ <ol style="list-style-type: none"> (a) $x = x \wedge \text{BC_RULES}(\text{KB}, A)$ 3. return x 	
BC_RULES(KB, A)	
Inputs	a set of Horn rules KB a sub/goal A
Outputs	if $\text{KB} \vdash A$ (<i>i.e.</i> , $\text{KB} \models A$), return true if $\text{KB} \not\vdash A$ (<i>i.e.</i> , $\text{KB} \not\models A$), return false
<ol style="list-style-type: none"> 1. let $y = \text{false}$ 2. for all Horn rules in KB of the form $A_1 \wedge \dots \wedge A_m \rightarrow A$ <ol style="list-style-type: none"> (a) if $m = 0$, let $\Sigma = \emptyset$ (b) else let $\Sigma = \{A_1, \dots, A_m\}$ (c) $y = y \vee \text{BC_GOALS}(\text{KB}, \Sigma)$ 3. return y 	

Table 1: Backward Chaining.

Pop R off Σ ; replace $P \wedge Q \wedge R \rightarrow S$ with $P \wedge Q \rightarrow S$
 Pop T off Σ ; replace $T \wedge U \rightarrow P$ with $U \rightarrow P$
 Pop U off Σ ; delete $U \rightarrow P$; insert P into Σ and Σ'
 Pop V off Σ ; delete $V \rightarrow Q$; insert Q into Σ and Σ'
 Pop P off Σ ; replace $P \wedge Q \rightarrow S$ with $Q \rightarrow S$
 Pop Q off Σ ; delete $Q \rightarrow S$; insert S into Σ and Σ'
 Pop S off Σ ; return $\Sigma' = \{R, T, U, V, P, Q, S\}$

3 Resolution

The simplest case of resolution, **unit resolution**, can be understood as an application of modus ponens of the form: given premises $\neg A \rightarrow B$ and $\neg A$, conclude B . For example, let A be the proposition “it is raining” and let B be the proposition “the ground is dry”. Now $\neg A \rightarrow B$ is interpreted as “if it is not raining, then the ground is dry.” Equivalently, “either it is raining or the ground is dry.” If $\neg A$ holds (*i.e.*, if it is not raining), then we conclude B (*i.e.*, the ground is dry).

FORWARD_CHAINING(KB, Σ)	
Inputs	a set of Horn rules KB an initial set of facts Σ
Output	$\{A \mid \text{KB} \cup \Sigma \models A\}$
Initialize	$\Sigma' = \Sigma$
while (Σ is not empty) do	
1. pop A_i off Σ	
2. for all Horn rules in KB of the form $A_i \rightarrow A$	
(a) add A to Σ and Σ'	
(b) delete $A_i \rightarrow A$ from KB	
/* for efficiency, not correctness */	
3. for all Horn rules in KB of the form $A_1 \wedge \dots \wedge A_n \rightarrow A$	
(a) add $A_1 \wedge \dots \wedge A_{i-1} \wedge A_{i+1} \wedge \dots \wedge A_n \rightarrow A$ to KB	
(b) delete $A_1 \wedge \dots \wedge A_n \rightarrow A$ from KB	
/* for efficiency, not correctness */	
return Σ'	

Table 2: Forward Chaining.

UNIT RESOLUTION (1R)

$$\frac{A \vee B \quad \neg B}{A} \text{ (1R)}$$

GROUND RESOLUTION (GR)

$$\frac{A_1 \vee \dots \vee A_m \vee C \quad B_1 \vee \dots \vee B_n \vee \neg C}{A_1 \vee \dots \vee A_m \vee B_1 \vee \dots \vee B_n} \text{ (GR)}$$

ITERATED GROUND RESOLUTION (IGR)

If $D_j = A_i$, then

$$\frac{A_1 \wedge \dots \wedge A_m \rightarrow B_1 \vee \dots \vee B_n \quad C_1 \wedge \dots \wedge C_k \rightarrow D_1 \vee \dots \vee D_l}{A_1 \wedge \dots \wedge A_{i-1} \wedge A_{i+1} \wedge \dots \wedge A_m \wedge C_1 \wedge \dots \wedge C_k \rightarrow B_1 \vee \dots \vee B_n \vee D_1 \vee \dots \vee D_{j-1} \vee D_{j+1} \vee \dots \vee D_l} \text{ (IGR)}$$

Theorem: IGR is sound.

Proof: Suppose not: i.e., suppose that there exists an interpretation which satisfies the premises of IGR but does not satisfy the conclusion. In such an interpretation, the antecedent of the conclusion is satisfied, while the consequent is not; in particular, none of $B_1, \dots, B_n, D_1, \dots, D_{j-1}, D_{j+1}, \dots, D_l$

are satisfied. Since the antecedent is satisfied, however, (i) $C_1 \wedge \dots \wedge C_k$ is satisfied, and (ii) $A_1 \wedge \dots \wedge A_{i-1} \wedge A_{i+1} \wedge \dots \wedge A_m$ is satisfied. Now, by (i) and the fact that the second premise is satisfied, one of D_1, \dots, D_l is satisfied. Two cases arise. If D_j is not satisfied, then one of $D_1, \dots, D_{j-1}, D_{j+1}, \dots, D_l$ is satisfied. Contradiction. Otherwise, if $D_j = A_i$ is satisfied, together with (ii), this implies that one of B_1, \dots, B_n is satisfied. Contradiction. Therefore, IGR is sound. \diamond

3.1 Normal Form

A formula is said to be in **normal form** iff it is of the form $\alpha_1 \wedge \dots \wedge \alpha_n \rightarrow \beta_1 \vee \dots \vee \beta_m$ where the α_i 's and β_j 's are atomic. All formulas of propositional logic (and indeed, predicate calculus) are convertible to normal form databases via the following procedure.¹

1. Eliminate implications.

- Rewrite $A \rightarrow B$ as $\neg A \vee B$.

2. Move negations inwards.

- Use DeMorgan's laws:
 - Rewrite $\neg(A \vee B)$ as $\neg A \wedge \neg B$.
 - Rewrite $\neg(A \wedge B)$ as $\neg A \vee \neg B$.
- Use the law of double negation:
 - Rewrite $\neg\neg A$ as A .

3. Distribute \vee over \wedge .

- Rewrite $A \vee (B \wedge C)$ as $(A \vee B) \wedge (A \vee C)$.
- Rewrite $(A \wedge B) \vee C$ as $(A \vee C) \wedge (B \vee C)$.

(Note: The knowledge base is now in conjunctive normal form (CNF).)

4. Split conjunctions: split up formulas like $A \wedge B$ into two separate entries in the knowledge base, namely A and B .

5. Flatten nested disjunctions: flatten formulas of the form $((A \vee B) \vee C)$ into $A \vee B \vee C$. (This is often done along the way.)

6. Eliminate negations by reintroducing implications.

- Rewrite $\neg A \vee B$ as $A \rightarrow B$.
- Rewrite $\neg A$ as $A \rightarrow \perp$.
- Rewrite A as $\top \rightarrow A$.

¹All formulas of first-order logic are also convertible to normal form databases, but that conversion procedure is a bit more complicated. See Lecture ??.

(As in Lecture 09, the connective \leftrightarrow is used as an abbreviation: in particular, $A \leftrightarrow B$ means $(A \rightarrow B) \wedge (B \rightarrow A)$.)

Example: [Ginsberg 1993] If a house is big and old, then it is a lot of work to maintain, unless it comes with a housecleaner and doesn't have a garden.

We express this sentence in propositional logic as $B \wedge O \rightarrow W \vee (C \wedge \neg G)$, using the following propositions:

B: The house is big.

O: The house is old.

W: The house is a lot of work to maintain.

C: The house comes with a housecleaner.

G: The house has a garden.

Following the steps outlined above, we now demonstrate how to convert this formula of propositional logic into a database in normal form:

1. Eliminate implications:

$$\neg(B \wedge O) \vee W \vee (C \wedge \neg G)$$

2. Move negations inwards:

$$\neg B \vee \neg O \vee W \vee (C \wedge \neg G)$$

3. Distribute \vee over \wedge :

$$\begin{aligned} &\neg B \vee \neg O \vee (W \vee C) \wedge (W \vee \neg G) \\ &\neg B \vee (\neg O \vee W \vee C) \wedge (\neg O \vee W \vee \neg G) \\ &(\neg B \vee \neg O \vee W \vee C) \wedge (\neg B \vee \neg O \vee W \vee \neg G) \end{aligned}$$

4. Split conjunctions:

$$\begin{aligned} &\neg B \vee \neg O \vee W \vee C \\ &\neg B \vee \neg O \vee W \vee \neg G \end{aligned}$$

5. Flatten nested disjunctions. (We've been doing that along the way.)

6. Eliminate negations by reintroducing implications:

$$\begin{aligned} &B \wedge O \rightarrow W \vee C \\ &B \wedge O \wedge G \rightarrow W \end{aligned}$$

3.2 Proof-by-Refutation

Recall that a proof theory Π is said to be *complete* iff $\text{KB} \models A$ implies $\text{KB} \vdash_{\Pi} A$, for all formulas A . A proof theory Π is *refutation-complete* iff $\text{KB} \models \perp$ implies $\text{KB} \vdash_{\Pi} \perp$. The inference rule IGR is refutation-complete for knowledge bases in normal form. Therefore, since $\text{KB} \models A$ iff $\text{KB} \cup \{\neg A\} \models \perp$, by soundness and refutation-completeness, $\text{KB} \models A$ iff $\text{KB}, \neg A \vdash_{\text{IGR}} \perp$.

This final observation motivates the solution to the logical entailment problem known as proof-by-refutation: (i) convert knowledge base $\text{KB} \cup \{\neg A\}$ to normal form; (ii) apply resolution until

$$\frac{\frac{\frac{\frac{\frac{\frac{\top \rightarrow C \quad C \rightarrow T \vee H}{\top \rightarrow T \vee H} \quad H \rightarrow R}{\top \rightarrow T \vee R} \quad T \rightarrow R \vee S}{\top \rightarrow R \vee S} \quad S \rightarrow R}{\top \rightarrow R} \quad R \rightarrow \perp}{\top \rightarrow \perp}}$$

$$\frac{\frac{\frac{\frac{\frac{\frac{\frac{S \rightarrow R \quad R \rightarrow \perp}{S \rightarrow \perp} \quad T \rightarrow R \vee S}{T \rightarrow R} \quad C \rightarrow T \vee H}{C \rightarrow R \vee H} \quad H \rightarrow R}{C \rightarrow R} \quad \top \rightarrow C}{\top \rightarrow R} \quad R \rightarrow \perp}{\top \rightarrow \perp}}$$

resolution is no longer applicable. The following are two proofs-by-refutation that $KB \cup \{\neg R\} \vdash \perp$, which implies that $KB \models R$, given $KB = \{\top \rightarrow C, C \rightarrow T \vee H, H \rightarrow R, T \rightarrow R \vee S, S \rightarrow R\}$.

Resolution pseudocode for propositional logic² appears in Table 3. All pairs of formulas in turn,

One way to implement this pseudocode would be to represent each formula as a pair of sets (e.g., $\phi = \langle \{T\}, \{R, S\} \rangle$). That way, resolvents that look like $C \rightarrow R \vee R$ would automatically be simplified to $C \rightarrow R$.

RESOLUTION(KB, A)	
Inputs	a knowledge base KB in normal form a goal A
Initialize	$\Delta = \emptyset$ $\Phi = KB \cup \{A \rightarrow \perp\}$
Outputs	if $KB, A \rightarrow \perp \vdash \top \rightarrow \perp$ (i.e., $KB \models A$), return true
LOOP	
1. for all pairs of formulas $\phi, \psi \in \Phi$	
(a) let R be the set of resolvents of ϕ and ψ	
(b) if $\top \rightarrow \perp \in R$, then return true	
(c) else $\Delta = \Delta \cup R$	
2. if $\Delta \subseteq \Phi$, then return false	
3. else $\Phi = \Phi \cup \Delta$	
FOREVER	

Table 3: Resolution.

²Adopted from Russell and Norvig, p. 216.

3.3 Resolution Strategies

Resolution is sound and refutation-complete: *i.e.*, $KB \models A$ iff $KB, \neg A \vdash_{\text{IGR}} \perp$. In other words, the resolution inference rule guarantees the existence of a proof (by contradiction) whenever a knowledge base logically entails a fact, but it is not always straightforward to find such a proof.

Exercise: Let $KB = \{\top \rightarrow A \vee B, A \rightarrow C, B \rightarrow C, C \rightarrow P, P \rightarrow Q, \top \rightarrow P\}$. Derive Q with and without explicitly deriving C .

There are a number of strategies that guide the application of resolution and aim to increase the efficiency of resolution theorem provers: (i) *unit resolution*: at least one of the formulas is a unit clause (*i.e.*, consists of one predicate symbol); (ii) *input resolution*: at least one of the formulas is in the knowledge base; (iii) *linear resolution*: one of the formulas is the most recent conclusion; (iv) *set of support resolution*: at least one of the formulas is in the set of support, which includes the negated query and all derived formulas; and (v) *ordered resolution*: set of support resolution in which the first clause is always the resolvent.

A control strategy is complete if it preserves refutation-completeness. Neither unit nor input resolution is complete, but linear resolution, set of support, and ordered resolution are complete. For example, it is not possible to derive a contradiction from $KB = \{\top \rightarrow P \vee Q, P \rightarrow Q, Q \rightarrow P, P \wedge Q \rightarrow \perp\}$ using unit or input resolution, although the following reasoning is sound:

$$\boxed{\frac{\frac{T \rightarrow P \vee Q \quad Q \rightarrow P}{\top \rightarrow P} \quad \frac{P \wedge Q \rightarrow \perp \quad P \rightarrow Q}{P \rightarrow \perp}}{\top \rightarrow \perp}}$$

In addition, the following is a linear derivation of a contradiction, given KB:

$$\boxed{\frac{\frac{\frac{\frac{\top \rightarrow P \vee Q \quad P \rightarrow Q}{\top \rightarrow Q}}{Q \rightarrow P} \quad \frac{P \wedge Q \rightarrow \perp}{\top \rightarrow P}}{Q \rightarrow \perp}}{\top \rightarrow Q}}{\top \rightarrow \perp}}$$