9 INFERENCE IN FIRST-ORDER LOGIC

In which we define inference mechanisms that can efficiently answer questions posed in first-order logic.

Chapter 6 defined the notion of **inference,** and showed how sound and complete inference can be achieved for propositional logic. In this chapter, we extend these results to first-order logic. In Section 9.1 we provide some additional basic inference rules to deal with quantifiers. In Section 9.2, we show how these inference rules, along with those for propositional logic, can be chained together to make proofs. By examining these proofs, we can come up with more powerful inference rules that make proofs much shorter and more intuitive. This makes it possible to design inference procedures that are sufficiently powerful to be of use to a knowledge-based agent. These rules and procedures are discussed in Sections 9.3 and 9.4. Section 9.5 describes the problem of completeness for first-order inference, and discusses the remarkable result, obtained by Kurt Gödel, that if we extend first-order logic with additional constructs to handle mathematical induction, then there is no complete inference procedure; even though the needle is in the metaphorical haystack, no procedure can guarantee to find it. Section 9.6 describes an inference procedure called **resolution** that is complete for any set of sentences in first-order logic, and Section 9.7 proves that it is complete.

9.1 INFERENCE RULES INVOLVING QUANTIFIERS

In Section 6.4, we saw the inference rules for propositional logic: Modus Ponens, And-Elimination, And-Introduction, Or-Introduction, and Resolution. These rules hold for first-order logic as well. But we will need additional inference rules to handle first-order logic sentences with quantifiers. The three additional rules we introduce here are more complex than previous ones, because we have to talk about substituting particular individuals for the variables. We will use the notation SUBST(θ, a) to denote the result of applying the substitution (or binding list) 6 to the sentence a. For example:

$$\text{SUBST}(\{x/Sam, y/Pam\},\ Likes(x,y)) = Likes(Sam, Pam)$$

The three new inference rules are as follows:

UNIVERSAL ELIMINATION

O **Universal Elimination:** For any sentence *a,* variable *v,* and ground term[1] *g*:

$$\frac{\mathrm{V}\ v\ a}{\text{SUBST}(\{v/g\}, \alpha)}$$

For example, from $\forall x\ \ Likes(x, IceCream)$, we can use the substitution *{x/Ben}* and infer *Likes(Ben, IceCream)*.

EXISTENTIAL ELIMINATION

O **Existential Elimination:** For any sentence α, variable v, and constant symbol *k* that does not appear elsewhere in the knowledge base:

$$\frac{\exists\ v\ a}{\text{SUBST}(\{v/k\}, \alpha)}$$

For example, from $\exists x\ \ Kill(x, Victim)$, we can infer *Kill(Murderer, Victim),* as long as *Murderer* does not appear elsewhere in the knowledge base.

EXISTENTIAL INTRODUCTION

◇ **Existential Introduction:** For any sentence α, variable v that does not occur in a, and ground term *g* that does occur in α:

$$\frac{\alpha}{\exists v\ \ \text{SUBST}(\{g/v\}, \alpha)}$$

For example, from *Likes(Jerry,IceCream)* we can infer 3 *x Likes(x, IceCream).*

You can check these rules using the definition of a universal sentence as the conjunction of all its possible instantiations (so Universal Elimination is like And-Elimination) and the definition of an existential sentence as the disjunction of all its possible instantiations.

It is very important that the constant used to replace the variable in the Existential Elimination rule is new. If we disregard this requirement, it is easy to produce consequences that do not follow logically. For example, suppose we have the sentence 3 *x Father(x, John)* ("John has a father"). If we replace the variable *x* by the constant *John*, we get *Father(John, John),* which is certainly not a logical consequence of the original sentence. Basically, the existential sentence says there is some object satisfying a condition, and the elimination process is just giving a name to that object. Naturally, that name must not already belong to another object. Mathematics provides a nice example: suppose we discover that there is a number that is a little bigger than 2.71828 and that satisfies the equation $d(x^y)/dy = x^y$ for x. We can give this number a name, such as *e,* but it would be a mistake to give it the name of an existing object, like π.

9.2 AN EXAMPLE PROOF

Having defined some inference rules, we now illustrate how to use them to do a proof. From here, it is only a short step to an actual proof procedure, because the application of inference rules is simply a question of matching their premise patterns to the sentences in the KB and then adding

[1] Recall from Chapter 7 that a ground term is a term that contains no variables—that is, either a constant symbol or a function symbol applied to some ground terms.

their (suitably instantiated) conclusion patterns. We will begin with the situation as it might be described in English:

> The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

What we wish to prove is that West is a criminal. We first represent these facts in first-order logic, and then show the proof as a sequence of applications of the inference rules.[2]

"... it is a crime for an American to sell weapons to hostile nations":

$$\begin{aligned} &\text{V}\, x, y, z \;\; American(x) \text{ A } Weapon(y) \text{A } Nation(z) \text{A } Hostile(z) \\ &\qquad \text{A } Sells(x, z, y) \Rightarrow Criminal(x) \end{aligned} \tag{9.1}$$

"Nono ... has some missiles":

$$\exists x \;\; Owns(Nono, x) \wedge Missile(x) \tag{9.2}$$

"All of its missiles were sold to it by Colonel West":

$$\forall x \;\; Owns(Nono, x) \text{A } Missile(x) \Rightarrow Sells(West, Nono, x) \tag{9.3}$$

We will also need to know that missiles are weapons:

$$\forall x \;\; Missile(x) \Rightarrow Weapon(x) \tag{9.4}$$

and that an enemy of America counts as "hostile":

$$\forall x \;\; Enemy(x, America) \Rightarrow Hostile(x) \tag{9.5}$$

"West, who is American ...":

$$American(West) \tag{9.6}$$

"The country Nono ...":

$$Nation(Nono) \tag{9.7}$$

"Nono, an enemy of America ...":

$$Enemy(Nono, America) \tag{9.8}$$

$$Nation(America) \tag{9.9}$$

The proof consists of a series of applications of the inference rules:

From (9.2) and Existential Elimination:

$$Owns(Nono, M1) \text{ A } Missile(M1) \tag{9.10}$$

From (9.10) and And-Elimination:

$$Owns(Nono, M1) \tag{9.11}$$

$$Missile(M\backslash) \tag{9.12}$$

From (9.4) and Universal Elimination:

$$Missile(M1) \Rightarrow Weapon(M1) \tag{9.13}$$

[2] Our representation of the facts will not be ideal according to the standards of Chapter 8, because this is mainly an exercise in proof rather than knowledge representation.

From (9.12), (9.13), and Modus Ponens:

$$\mathit{Weapon}(M1) \tag{9.14}$$

From (9.3) and Universal Elimination:

$$\mathit{Owns}(\mathit{Nono}, M1) \text{A } \mathit{Missile}(M1) \Rightarrow \mathit{Sells}(\mathit{West}, \mathit{Nono}, M1) \tag{9.15}$$

From (9.15), (9.10), and Modus Ponens:

$$\mathit{Sells}(\mathit{West}, \mathit{Nono}, M1) \tag{9.16}$$

From (9.1) and Universal Elimination (three times):

$$\begin{gathered}\mathit{American}(\mathit{West}) \text{ A } \mathit{Weapon}(M1) \text{A } \mathit{Nation}(\mathit{Nono}) \text{A } \mathit{Hostile}(\mathit{Nono}) \\ \text{A } \mathit{Sells}(\mathit{West}, \mathit{Nono}, M1) \Rightarrow \mathit{Criminal}(\mathit{West})\end{gathered} \tag{9.17}$$

From (9.5) and Universal Elimination:

$$\mathit{Enemy}(\mathit{Nono}, \mathit{America}) \Rightarrow \mathit{Hostile}(\mathit{Nono}) \tag{9.18}$$

From (9.8), (9.18), and Modus Ponens:

$$\mathit{Hostile}(\mathit{Nono}) \tag{9.19}$$

From (9.6), (9.7), (9.14), (9.16), (9.19), and And-Introduction:

$$\begin{gathered}\mathit{American}(\mathit{West}) \text{ A } \mathit{Weapon}(M1) \text{A } \mathit{Nation}(\mathit{Nono}) \\ \text{A } \mathit{Hostile}(\mathit{Nono}) \text{ A } \mathit{Sells}(\mathit{West}, \mathit{Nono}, M1)\end{gathered} \tag{9.20}$$

From (9.17), (9.20), and Modus Ponens:

$$\mathit{Criminal}(\mathit{West}) \tag{9.21}$$

If we formulate the process of finding a proof as a search process, then obviously this proof is the solution to the search problem, and equally obviously it would have to be a pretty smart program to find the proof without following any wrong paths. As a search problem, we would have

Initial state = KB (sentences 9.1-9.9)

Operators = applicable inference rules

Goal test = KB containing *Criminal(West)*

This example illustrates some important characteristics:

- The proof is 14 steps long.
- The branching factor increases as the knowledge base grows; this is because some of the inference rules combine existing facts.
- Universal Elimination can have an enormous branching factor on its own, because we can replace the variable by any ground term.
- We spent a lot of time combining atomic sentences into conjunctions, instantiating universal rules to match, and then applying Modus Ponens.

Thus, we have a serious difficulty, in the form of a collection of operators that give long proofs and a large branching factor, and hence a potentially explosive search problem. We also have an opportunity, in the form of an identifiable pattern of application of the operators (combining atomic sentences, instantiating universal rules, and then applying Modus Ponens). If we can define a better search space in which a single operator takes these three steps, then we can find proofs more efficiently.

9.3 GENERALIZED MODUS PONENS

In this section, we introduce a generalization of the Modus Ponens inference rule that does in a single blow what required an And-Introduction, Universal Elimination, and Modus Ponens in the earlier proof. The idea is to be able to take a knowledge base containing, for example:

$$Missile(M1)$$
$$Owns(Nono, M1)$$
$$\forall x \; Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, Nono, x)$$

and infer in one step the new sentence

$$Sells(West, Nono, M1)$$

Intuitively, this inference seems quite obvious. The key is to find some x in the knowledge base such that x is a missile and Nono owns x, and then infer that West sells this missile to Nono. More generally, if there is some substitution involving x that makes the premise of the implication identical to sentences already in the knowledge base, then we can assert the conclusion of the implication, after applying the substitution. In the preceding case, the substitution $\{x/M1\}$ achieves this.

We can actually make Modus Ponens do even more work. Suppose that instead of knowing $Owns(Nono, M1)$, we knew that everyone owns M1 (a communal missile, as it were):

$$\forall y \quad Owns(y, M1)$$

Then we would still like to be able to conclude that $Sells(West, Nono, M1)$. This inference could be carried out if we first applied Universal Elimination with the substitution $\{y/Nono\}$ to get $Owns(Nono, M1)$. The generalized version of Modus Ponens can do it in one step by finding a substitution for both the variables in the implication sentence and the variables in the sentences to be matched. In this case, applying the substitution $\{x/M1, y/Nono\}$ to the premise $Owns(Nono, x)$ and the sentence $Owns(y, M1)$ will make them identical. If this can be done for all the premises of the implication, then we can infer the conclusion of the implication. The rule is as follows:

GENERALIZED MODUS PONENS

◊ **Generalized Modus Ponens:** For atomic sentences p_i, p_i', and q, where there is a substitution θ such that SUBST(θ, p_i') = SUBST(θ, p_i), for all i:

$$\frac{p_1', \quad p_2', \quad \ldots, \quad p_n', \quad (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

There are $n + 1$ premises to this rule: the n atomic sentences p_i' and the one implication. There is one conclusion: the result of applying the substitution to the consequent q. For the example with West and the missile:

p_1' is $Missile(M1)$	p_1 is $Missile(x)$
p_2' is $Owns(y, M1)$	p_2 is $Owns(Nono, x)$
θ is $\{x/M1, y/Nono\}$	q is $Sells(West, Nono, x)$
SUBST(θ, q) is $Sells(West, Nono, M1)$	

Generalized Modus Ponens is an efficient inference rule for three reasons:

1. It takes bigger steps, combining several small inferences into one.

2. It takes sensible steps—it uses substitutions that are guaranteed to help rather than randomly trying Universal Eliminations. The **unification** algorithm takes two sentences and returns a substitution that makes them look the same if such a substitution exists.

UNIFICATION

3. It makes use of a precompilation step that converts all the sentences in the knowledge base into a **canonical form.** Doing this once and for all at the start means we need not waste time trying conversions during the course of the proof.

CANONICAL FORM

We will deal with canonical form and unification in turn.

Canonical Form

We are attempting to build an inferencing mechanism with one inference rule—the generalized version of Modus Ponens. That means that all sentences in the knowledge base should be in a form that matches one of the premises of the Modus Ponens rule—otherwise, they could never be used. In other words, the canonical form for Modus Ponens mandates that each sentence in the knowledge base be either an atomic sentence or an implication with a conjunction of atomic sentences on the left hand side and a single atom on the right. As we saw on page 174, sentences of this form are called **Horn sentences,** and a knowledge base consisting of only Horn sentences is said to be in Horn Normal Form.

HORN SENTENCES

We convert sentences into Horn sentences when they are first entered into the knowledge base, using Existential Elimination and And-Elimination.[3] For example, $3 x \quad Owns(Nono, x)$A $Missile(x)$ is converted into the two atomic Horn sentences $Owns(Nono, M1)$and $Missle(M1)$. Once the existential quantifiers are all eliminated, it is traditional to drop the universal quantifiers, so that $\forall y \quad Owns(y, M1)$would be written as $Owns(y, M1)$. This is just an abbreviation—the meaning of$y$ is still a universally quantified variable, but it is simpler to write and read sentences without the quantifiers. We return to the issue of canonical form on page 278.

Unification

The job of the unification routine, UNIFY, is to take two atomic sentences p and q and return a substitution that would make p and q look the same. (If there is no such substitution, then UNIFY should return *fail.)* Formally,

$$\text{UNIFY}(p, q) = 6 \text{ where } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$$

UNIFIER

0 is called the **unifier** of the two sentences. We will illustrate unification in the context of an example, delaying discussion of detailed algorithms until Chapter 10. Suppose we have a rule

$$Knows(John, x) \Rightarrow Hates(John, x)$$

("John hates everyone he knows") and we want to use this with the Modus Ponens inference rule to find out whom he hates. In other words, we need to find those sentences in the knowledge base

[3] We will see in Section 9.5 that not all sentences can be converted into Horn form. Fortunately, the sentences in our example (and in many other problems) can be.

that unify with *Knows*(*John*, *x*), and then apply the unifier to *Hates*(*John*, *x*). Let our knowledge base contain the following sentences:

$$\begin{array}{l} Knows(John, Jane) \\ Knows(y, Leonid) \\ Knows(y, Mother(y)) \\ Knows(x, Elizabeth) \end{array}$$

(Remember that x and y are implicitly universally quantified.) Unifying the antecedent of the rule against each of the sentences in the knowledge base in turn gives us:

$$\begin{array}{ll} \text{UNIFY}(Knows(John, x), Knows(John, Jane)) = & \{x/Jane\} \\ \text{UNIFY}(Knows(John, x), Knows(y, Leonid)) = & \{x/Leonid, y/John\} \\ \text{UNIFY}(Knows(John, x), Knows(y, Mother(y))) = & \{y/John, x/Mother(John)\} \\ \text{UNIFY}(Knows(John, x), Knows(x, Elizabeth)) = & fail \end{array}$$

The last unification fails because x cannot take on the value *John* and the value *Elizabeth* at the same time. But intuitively, from the facts that John hates everyone he knows and that everyone knows Elizabeth, we should be able to infer that John hates Elizabeth. It should not matter if the sentence in the knowledge base is *Knows*(*x*, *Elizabeth*) or *Knows*(*y*, *Elizabeth*).

STANDARDIZE APART

One way to handle this problem is to **standardize apart** the two sentences being unified, which means renaming the variables of one (or both) to avoid name clashes. After standardizing apart, we would have

$$\text{UNIFY}(Knows(John, x_1), Knows(x_2, Elizabeth)) = \{x_1/Elizabeth, x_2/John\}$$

The renaming is valid because $\forall x\ Knows(x, Elizabeth)$ and $\forall x_2\ Knows(x_2, Elizabeth)$ have the same meaning. (See also Exercise 9.2.)

There is one more complication: we said that UNIFY should return a substitution that makes the two arguments look the same. But if there is one such substitution, then there are an infinite number:

$$\begin{array}{lll} \text{UNIFY}(Knows(John, x), Knows(y, z)) & = & \{y/John, x/z\} \\ & \text{or} & \{y/John, x/z, w/Freda\} \\ & \text{or} & \{y/John, x/John, z/John\} \\ & \text{or} & \cdots \end{array}$$

MOST GENERAL UNIFIER

Thus, we insist that UNIFY returns the **most general unifier** (or MGU), which is the substitution that makes the least commitment about the bindings of the variables. In this case it is $\{y/John, x/z\}$.

Sample proof revisited

Let us solve our crime problem using Generalized Modus Ponens. To do this, we first need to put the original knowledge base into Horn form. Sentences (9.1) through (9.9) become

$$American(x) \land Weapon(y) \land Nation(z) \land Hostile(z)$$
$$\land\ Sells(x, z, y) \Rightarrow Criminal(x) \qquad (9.22)$$
$$Owns(Nono, M_1) \qquad (9.23)$$

$$Missile(M1) \tag{9.24}$$
$$Owns(Nono,x) \land Missile(x) \Rightarrow Sells(West, Nono, x) \tag{9.25}$$
$$Missile(x) \Rightarrow Weapon(x) \tag{9.26}$$
$$Enemy(x,America) \Rightarrow Hostile(x) \tag{9.27}$$
$$American(West) \tag{9.28}$$
$$Nation(Nono) \tag{9.29}$$
$$Enemy(Nono,America) \tag{9.30}$$
$$Nation(America) \tag{9.31}$$

The proof involves just four steps. From (9.24) and (9.26) using Modus Ponens:

$$Weapon(M1) \tag{9.32}$$

From (9.30) and (9.27) using Modus Ponens:

$$Hostile(Nono) \tag{9.33}$$

From (9.23), (9.24), and (9.25) using Modus Ponens:

$$Sells(West, Nono, M1) \tag{9.34}$$

From (9.28), (9.32), (9.29), (9.33), (9.34) and (9.22), using Modus Ponens:

$$Criminal(West) \tag{9.35}$$

This proof shows how natural reasoning with Generalized Modus Ponens can be. (In fact, one might venture to say that it is not unlike the way in which a human might reason about the problem.) In the next section, we describe systematic reasoning algorithms using Modus Ponens. These algorithms form the basis for many large-scale applications of AI, which we describe in Chapter 10.

9.4 FORWARD AND BACKWARD CHAINING

Now that we have a reasonable language for representing knowledge, and a reasonable inference rule (Generalized Modus Ponens) for using that knowledge, we will study how a reasoning program is constructed.

The Generalized Modus Ponens rule can be used in two ways. We can start with the sentences in the knowledge base and generate new conclusions that in turn can allow more inferences to be made. This is called **forward chaining.** Forward chaining is usually used when a new fact is added to the database and we want to generate its consequences. Alternatively, we can start with something we want to prove, find implication sentences that would allow us to conclude it, and then attempt to establish their premises in turn. This is called **backward chaining,** because it uses Modus Ponens backwards. Backward chaining is normally used when there is a goal to be proved.

FORWARD CHAINING

BACKWARD CHAINING

Forward-chaining algorithm

Forward chaining is normally triggered by the addition of a new fact p to the knowledge base. It can be incorporated as part of the TELL process, for example. The idea is to find all implications that have p as a premise; then if the other premises are already known to hold, we can add the consequent of the implication to the knowledge base, triggering further inference (see Figure 9.1).

RENAMING

The FORWARD-CHAIN procedure makes use of the idea of a **renaming.** One sentence is a renaming of another if they are identical except for the names of the variables. For example, *Likes(x, IceCream)* and *Likes(y, IceCream)* are renamings of each other because they only differ in the choice of x or y, but *Likes(x, x)* and *Likes(x, y)* are not renamings of each other.

COMPOSITION

We also need the idea **of a composition** of substitutions. COMPOSE(θ_1, θ_2) is the substitution whose effect is identical to the effect of applying each substitution in turn. That is,

$$\text{SUBST}(\text{COMPOSE}(\theta_1, \theta_2), p) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, p))$$

We will use our crime problem again to illustrate how FORWARD-CHAIN works. We will begin with the knowledge base containing only the implications in Horn form:

$$American(x)\ \mathrm{A}\ Weapon(y)\mathrm{A}\ Nation(z)\mathrm{A}\ Hostile(z)$$
$$\mathrm{A}\ Sells(x,z,y) \Rightarrow Criminal(x) \quad (9.36)$$
$$Owns(Nono,x)\ \mathrm{A}\ Missile(x) \Rightarrow Sells(West,Nono,x) \quad (9.37)$$
$$Missile(x) \Rightarrow Weapon(x) \quad (9.38)$$
$$Enemy(x, America) \Rightarrow Hostile(x) \quad (9.39)$$

```
procedure FORWARD-CHAIN(KB, p)

  if there is a sentence in KB that is a renaming of p then return
  Add p to KB
  for each ($p_1$ A ... A $p_n$ ⇒ q) in KB such that for some i, UNIFY($p_i$, p) = 0 succeeds do
      FIND-AND-INFER(KB, [$p_1$, ... , $p_{i-1}$, $p_{i+1}$, ... , $p_n$], q, θ)
  end
-----------------------------------------------------------------
procedure FIND-AND-INFER(KB, premises, conclusion, θ)

  if premises = [ \ then
      FORWARD-CHAIN(KB, SUBST(θ, conclusion))
  else for each p' in KB such that UNIFY(p', SUBST(θ, FIRST(premises))) = $\theta_2$ do
      FIND-AND-INFER(KB, REST(premises), conclusion, COMPOSE(θ, $\theta_2$))
  end
```

Figure 9.1 The forward-chaining inference algorithm. It adds to *KB* all the sentences that can be inferred from the sentence p. If p is already in *KB*, it does nothing. If p is new, consider each implication that has a premise that matches p. For each such implication, if all the remaining premises are in *KB*, then infer the conclusion. If the premises can be matched several ways, then infer each corresponding conclusion. The substitution θ keeps track of the way things match.

Now we add the atomic sentences to the knowledge base one by one, forward chaining each time and showing any additional facts that are added:

FORWARD-CHAIN($KB, American(West)$)

Add to the KB. It unifies with a premise of (9.36), but the other premises of (9.36) are not known, so FORWARD-CHAIN returns without making any new inferences.

FORWARD-CHAIN($KB, Nation(Nono)$)

Add to the KB. It unifies with a premise of (9.36), but there are still missing premises, so FORWARD-CHAIN returns.

FORWARD-CHAIN($KB, Enemy(Nono, America)$)

Add to the KB. It unifies with the premise of (9.39), with unifier $\{x/Nono\}$. Call

FORWARD-CHAIN($KB, Hostile(Nono)$)

Add to the KB. It unifies with a premise of (9.36). Only two other premises are known, so processing terminates.

FORWARD-CHAIN($KB, Owns(Nono, M1)$)

Add to the KB. It unifies with a premise of (9.37), with unifier $\{x/M1\}$. The other premise, now $Missile(M1)$, is not known, so processing terminates.

FORWARD-CHAIN($KB, Missile(M1)$)

Add to the KB. It unifies with a premise of (9.37) and (9.38). We will handle them in that order.

- $Missile(M1)$ unifies with a premise of (9.37) with unifier $\{x/M1\}$. The other premise, now $Owns(Nono, M1)$, is known, so call

  FORWARD-CHAIN($KB, Sells(West, Nono, M1)$)

  Add to the KB. It unifies with a premise of (9.36), with unifier $\{x/West, y/M1, z/Nono\}$. The premise $Weapon(M1)$ is unknown, so processing terminates.
- $Missile(M1)$ unifies with a premise of (9.38) with unifier $\{x/M1\}$. Call

  FORWARD-CHAIN($KB, Weapon(M1)$)

  Add to the KB. It unifies with a premise of (9.36), with unifier $\{y/M1\}$. The other premises are all known, with accumulated unifier $\{x/West, y/M1, z/Nono\}$. Call

  FORWARD-CHAIN($KB, Criminal(West)$)

  Add to the KB. Processing terminates.

DATA-DRIVEN

As can be seen from this example, forward chaining builds up a picture of the situation gradually as new data comes in. Its inference processes are not directed toward solving any particular problem; for this reason it is called a **data-driven or data-directed** procedure. In this example, there were no rules capable of drawing irrelevant conclusions, so the lack of directedness was not a problem. In other cases (for example, if we have several rules describing the eating habits of Americans and the price of missiles), FORWARD-CHAIN will generate many irrelevant conclusions. In such cases, it is often better to use backward chaining, which directs all its effort toward the question at hand.

Backward-chaining algorithm

Backward chaining is designed to find all answers to a question posed to the knowledge base. Backward chaining therefore exhibits the functionality required for the ASK procedure. The backward-chaining algorithm BACK-CHAIN works by first checking to see if answers can be provided directly from sentences`in the knowledge base. It then finds all implications whose conclusion unifies with the query, and tries to establish the premises of those implications, also by backward chaining. If the premise is a conjunction, then BACK-CHAIN processes the conjunction conjunct by conjunct, building up the unifier for the whole premise as it goes. The algorithm is shown in Figure 9.2.

Figure 9.3 is the proof tree for deriving *Criminal*(*West*) from sentences (9.22) through (9.30). As a diagram of the backward chaining algorithm, the tree should be read depth-first, left to right. To prove *Criminal(x)* we have to prove the five conjuncts below it. Some of these are in the knowledge base, and others require further backward chaining. Each leaf node has the substitution used to obtain it written below. Note that once one branch of a conjunction succeeds, its substitution is applied to subsequent branches. Thus, by the time BACK-CHAIN gets to *Sells(x, z, y)*, all the variables are instantiated. Figure 9.3 can also be seen as a diagram of forward chaining. In this interpretation, the premises are added at the bottom, and conclusions are added once all their premises are in the *KB*. Figure 9.4 shows what can happen if an incorrect choice is made in the search—in this case, choosing America as the nation in question. There is no way to prove that America is a hostile nation, so the proof fails to go through, and we have to back up and consider another branch in the search space.

```
function BACK-CHAIN(KB, q) returns a set of substitutions
  BACK-CHAIN-LIST(KB, [q], {})

function BACK-CHAIN-LIST(KB, qlist, 9) returns a set of substitutions
  inputs: KB, a knowledge base
          qlist, a list of conjuncts forming a query ($\theta$ already applied)
          6, the current substitution
  static: answers, a set of substitutions, initially empty

  if qlist is empty then return $\{\theta\}$
  $q \leftarrow$ FIRST(qlist)
     for each $q'_i$ in KB such that $\theta_i \leftarrow$ UNIFY($q, q'_i$) succeeds do
        Add COMPOSE($\theta, \theta_i$) to answers
     end
     for each sentence ($p_1$ A ... A $p_n \Rightarrow q'_i$) in KB such that $\theta_i \leftarrow$ UNIFY($q, q'_i$) succeeds do
        answers $\leftarrow$ BACK-CHAIN-LIST(KB, SUBST($\theta_i$, [p\ . . . $p_n$]), COMPOSE($\theta, \theta_i$)) U answers
     end
  return the union of BACK-CHAIN-LIST(KB, REST(qlist), $\theta$) for each 6 $\in$ answers
```

Figure 9.2 The backward-chaining algorithm.

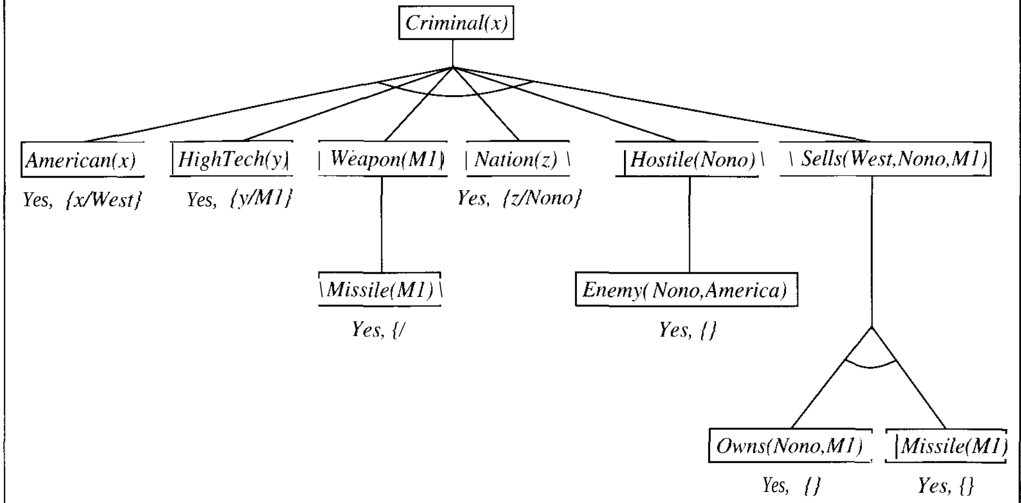


Figure 9.3 Proof tree to infer that West is a criminal.

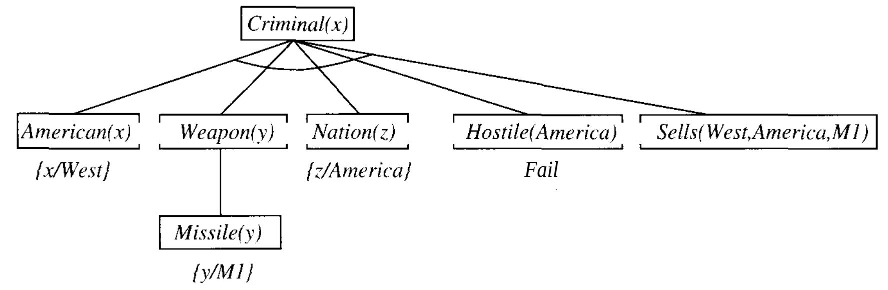


Figure 9.4 A failed proof tree: nothing can be inferred.

9.5 COMPLETENESS

Suppose we have the following knowledge base:

$$\begin{array}{l} \forall x \;\; P(x) \Rightarrow Q(x) \\ \forall x \;\; \neg P(x) \Rightarrow R(x) \\ \forall x \;\; Q(x) \Rightarrow S(x) \\ \forall x \;\; R(x) \Rightarrow S(x) \end{array} \qquad (9.40)$$

Then we certainly want to be able to conclude $S(A)$; $S(A)$ is true if $Q(A)$ or $R(A)$ is true, and one of those must be true because either P(A) is true or $\neg P(A)$ is true.

Unfortunately, chaining with Modus Ponens cannot derive $S(A)$ for us. The problem is that $\forall x\ \neg P(x) \Rightarrow R(x)$ cannot be converted to Horn form, and thus cannot be used by Modus Ponens. That means that a proof procedure using Modus Ponens is **incomplete:** there are sentences entailed by the knowledge base that the procedure cannot infer.

INCOMPLETE

The question of the existence of complete proof procedures is of direct concern to mathematicians. If a complete proof procedure can be found for mathematical statements, two things follow: first, all conjectures can be established mechanically; second, all of mathematics can be established as the logical consequence of a set of fundamental axioms. A complete proof procedure for first-order logic would also be of great value in AI: barring practical issues of computational complexity, it would enable a machine to solve any problem that can be stated in the language.

The question of completeness has therefore generated some of the most important mathematical work of the twentieth century. This work culminated in the results proved by the German mathematician Kurt Godel in 1930 and 1931. Godel has some good news for us; his **completeness theorem** showed that, for first-order logic, any sentence that is entailed by another set of sentences can be proved from that set. That is, we can find inference rules that allow a **complete** proof procedure R:

COMPLETENESS THEOREM

$$\textit{\textbf{if KB}} \models \alpha \textbf{ then } \textit{\textbf{KB}} \vdash_R \alpha$$

The completeness theorem is like saying that a procedure for finding a needle in a haystack does exist. This is not a trivial claim, because universally quantified sentences and arbitrarily nested function symbols add up to haystacks of infinite size. Godel showed that a proof procedure exists, but he did not demonstrate one; it was not until 1965 that Robinson published his **resolution algorithm,** which we discuss in the next section.

RESOLUTION ALGORITHM

There is one problem with the completeness theorem that is a real nuisance. Note that we said that *if* a sentence follows, then it can be proved. Normally, we do not know until the proof is done that the sentence *does* follow; what happens when the sentence doesn't follow? Can we tell? Well, for first-order logic, it turns out that we cannot; our proof procedure can go on and on, but we will not know if it is stuck in a hopeless loop or if the proof is just about to pop out. (This is like the halting problem for Turing machines.) Entailment in first-order logic is thus **semidecidable**, that is, we can show that sentences follow from premises, if they do, but we cannot always show it if they do not. As a corollary, consistency of sets of sentences (the question of whether there is a way to make all the sentences true) is also semidecidable.

SEMIDECIDABLE

9.6 RESOLUTION: A COMPLETE INFERENCE PROCEDURE

Recall from Chapter 6 that the simple version of the resolution inference rule for propositional logic has the following form:

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \quad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

The rule can be understood in two ways. First, we can see it as reasoning by cases. If *3* is false, then from the first disjunction, a must be true; but if β is true, then from the second disjunction ^ must be true. Hence, either *a* or - must be true. The second way to understand it is as transitivity of implication: from two implications, we derive a third that links the premise of the first to the conclusion of the second. Notice that Modus Ponens does not allow us to derive new implications; it only derives atomic conclusions. Thus, the resolution rule is more powerful than Modus Ponens. In this section, we will see that a generalization of the simple resolution rule can serve as the sole inference rule in a complete inference procedure for first-order logic.

The resolution inference rule

In the simple form of the resolution rule, the premises have exactly two disjuncts. We can extend that to get a more general rule that says that for two disjunctions of any length, if one of the disjuncts in one clause (p_j) unifies with the negation of a disjunct in the other *(qk),* then infer the disjunction of all the disjuncts except for those two:

GENERALIZED RESOLUTION (DISJUNCTIONS)

0 Generalized Resolution (disjunctions): For literals p_i and q_i, where UNIFY$(p_j, \neg q_k)$ = *0:*

$$\frac{p\backslash \vee \ldots pj \ldots \vee p_m, \quad q_1 \vee \cdots q_k \ldots \vee q_n}{\text{SUBST}(\theta, (p_1 \vee \ldots p_{j-1} \vee p_{j+1} \ldots p_m \vee q_1 \ldots q_{k-1} \vee q_{k+1} \ldots \vee q_n))}$$

Equivalently, we can rewrite this in terms of implications:

GENERALIZED RESOLUTION (IMPLICATIONS)

◇ **Generalized Resolution (implications):** For atoms p_i, q_i, r_i, s_i where UNIFY(p_j, q_k) = *0:*

$$\frac{p_1 \wedge \ldots p_j \ldots \wedge p_{n_1} \Rightarrow r_1 \vee \ldots r_{n_2} \quad s_1 \wedge \ldots \wedge s_{n_3} \Rightarrow q\backslash \vee \ldots q_k \ldots \vee q_{n_4}}{\text{SUBST}(\theta, (p_1 \wedge \ldots p_{j-1} \wedge p_{j+1} \wedge p_{n_1} \wedge s_1 \wedge \ldots s_{n_3} \Rightarrow r_1 \vee \ldots r_{n_2} \vee q_1 \vee \ldots q_{k-1} \vee q_{k+1} \vee \ldots \vee q_{n_4}))}$$

Canonical forms for resolution

In the first version of the resolution rule, every sentence is a disjunction of literals. All the disjunctions in the KB are assumed to be joined in one big, implicit conjunction (as in a normal KB), so this form is called **conjunctive normal form** (or CNF), even though each individual sentence is a disjunction (confusing, isn't it?).

CONJUNCTIVE NORMAL FORM

In the second version of the resolution rule, each sentence is an implication with a conjunction of atoms on the left and a disjunction of atoms on the right. We call this **implicative normal form** (or INF), although the name is not standard. We can transform the sentences in (9.40) into either of the two forms, as we now show. (Notice that we have standardized apart the variable names in these sentences.)

IMPLICATIVE NORMAL FORM

$$\begin{array}{ll}
\text{Conjunctive Normal Form} & \text{Implicative Normal Form} \\
\neg P(w) \vee Q(w) & P(w) \Rightarrow Q(w) \\
P(x) \vee R(x) & True \Rightarrow P(x) \vee R(x) \\
\neg Q(y) \vee S(y) & Q(y) \Rightarrow S(y) \\
\neg R(z) \vee S(z) & R(z) \Rightarrow S(z)
\end{array} \tag{9.41}$$

The two forms are notational variants of each other, and as we will see on page 281, any set of sentences can be translated to either form. Historically, conjunctive normal form is more common, but we will use implicative normal form, because we find it more natural.

It is important to recognize that *resolution is a generalization of modus ponens.* Clearly, the implicative normal form is more general than Horn form, because the right-hand side can be a disjunction, not just a single atom. But at first glance it seems that Modus Ponens has the ability to combine atoms with an implication to infer a conclusion in a way that resolution cannot do. This is just an illusion—once we realize that an atomic sentence α in implicative normal form is written as $True \Rightarrow a$, we can see that modus ponens is just a special case of resolution:

$$\frac{\alpha,\ a \Rightarrow \beta}{ft} \quad \text{is equivalent to} \quad \frac{True \Rightarrow a,\ \ \mathrm{a} \Rightarrow \beta}{True \Rightarrow 3}$$

Even though $True \Rightarrow \alpha$ is the "correct" way to write an atomic sentence in implicative normal form, we will sometimes write α as an abbreviation.

Resolution proofs

One could use resolution in a forward- or backward-chaining algorithm, just as Modus Ponens is used. Figure 9.5 shows a three-step resolution proof of *S(A)* from the KB in (9.41).

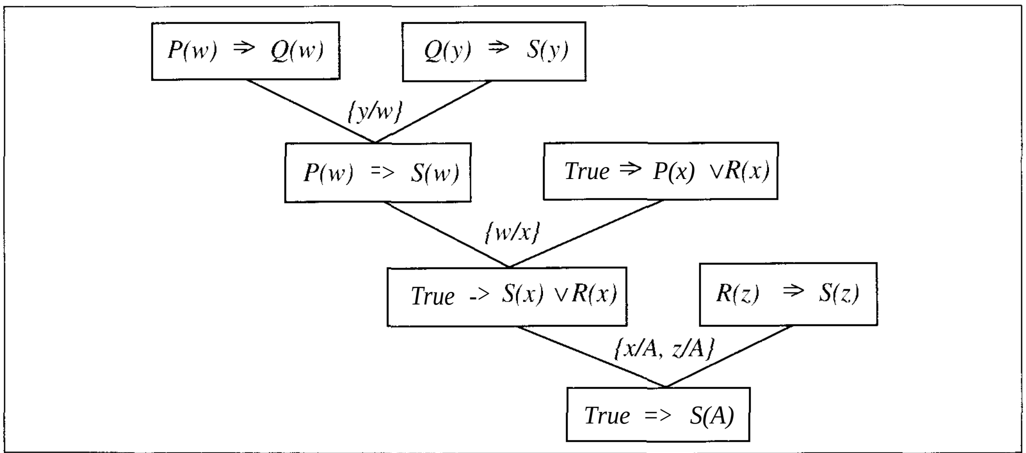


Figure 9.5 A proof that 5(A) follows from the KB in (9.41), using resolution. Each "vee" in the proof tree represents a resolution step: the two sentences at the top are the premises, and the one at the bottom is the conclusion or **resolvent.** The substitution is shown for each resolution.

Technically, the final resolvent should be $True \Rightarrow S(A) \vee S(A)$, but we have taken the liberty of removing the redundant disjunct. In some systems, there is a separate inference rule called **factoring** to do this, but it is simpler to just make it be part of the resolution rule.

FACTORING

Chaining with resolution is more powerful than chaining with Modus Ponens, but it is still not complete. To see that, consider trying to prove $P \vee \neg P$ from the empty KB. The sentence is valid, but with nothing in the KB, there is nothing for resolution to apply to, and we are unable to prove anything.

REFUTATION

One complete inference procedure using resolution is **refutation,** also known as **proof by contradiction** and **reductio ad absurdum.** The idea is that to prove P, we assume P is false (i.e., add $\neg P$ to the knowledge base) and prove a contradiction. If we can do this, then it must be that the knowledge base implies P. In other words:

$$(KB \wedge \neg P \Rightarrow False) \Leftrightarrow (KB \Rightarrow P)$$

Proof by contradiction is a powerful tool throughout mathematics, and resolution gives us a simple, sound, complete way to apply it. Figure 9.6 gives an example of the method. We start with the knowledge base of (9.41) and are attempting to prove $S(A)$. We negate this to get $\neg S(A)$, which in implicative normal form is $S(A) \Rightarrow False$, and add it to the knowledge base. Then we apply resolution until we arrive at a contradiction, which in implicative normal form is $True \Rightarrow False$. It takes one more step than in Figure 9.5, but that is a small price to pay for the security of a complete proof method.

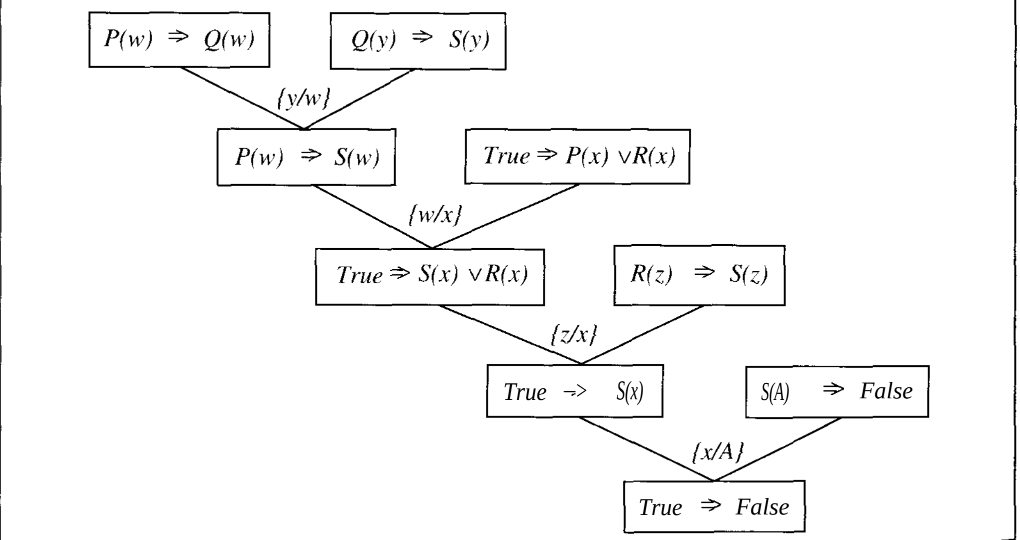


Figure 9.6 A proof that $S(A)$ follows from the KB in (9.41) using resolution with refutation.

Conversion to Normal Form

So far, we have claimed that resolution is complete, but we have not shown it. In this section we show that *any* first-order logic sentence can be put into implicative (or conjunctive) normal form, and in the following section we will show that from a set of sentences in normal form we can prove that a given sentence follows from the set.

We present the procedure for converting to normal form, step by step, showing that each step does not change the meaning; it should be fairly clear that all possible sentences are dealt with properly. You should understand why each of the steps in this procedure is valid, but few people actually manage to remember them all.

◇ **Eliminate implications:** Recall that $p \Rightarrow q$ is the same as $\neg p$ V q. So replace all implications by the corresponding disjunctions.

◇ **Move $\neg$ inwards:** Negations are allowed only on atoms in conjunctive normal form, and not at all in implicative normal form. We eliminate negations with wide scope using de Morgan's laws (see Exercise 6.2), the quantifier equivalences and double negation:

$$\begin{array}{rll}
\neg(p \mathrm{\ V\ } q) & \text{becomes} & \neg p \mathrm{\ A\ } \neg q \\
\neg(p \mathrm{\ A\ } q) & \text{becomes} & \neg p \mathrm{\ V\ } \neg q \\
\neg\forall\, x,p & \text{becomes} & \exists\, x\ \neg p \\
\neg\exists\, x,p & \text{becomes} & \forall\, x\ \neg p \\
\neg\neg p & \text{becomes} & p
\end{array}$$

0 **Standardize variables:** For sentences like $(\forall x\ \ P(x))$ V $(\exists x\ \ Q(x))$ that use the same variable name twice, change the name of one of the variables. This avoids confusion later when we drop the quantifiers.

0 **Move quantifiers left:** The sentence is now in a form in which all the quantifiers can be moved to the left, in the order in which they appear, without changing the meaning of the sentence. It is tedious to prove this properly; it involves equivalences such as

$$p \mathrm{\ V\ } \forall x\ \ q \text{ becomes } \forall x\ \ p \mathrm{\ V\ } q$$

which is true because p here is guaranteed not to contain an x.

SKOLEMIZATION

0 **Skolemize**: **Skolemization** is the process of removing existential quantifiers by elimination. In the simple case, it is just like the Existential Elimination rule of Section 9.1— translate $\exists x\ \ P(x)$ into $P(A)$, where A *is a* constant that does not appear elsewhere in the KB. But there is the added complication that some of the existential quantifiers, even though moved left, may still be nested inside a universal quantifier. Consider "Everyone has a heart":

$$\forall x\ Person(x) \Rightarrow \exists y\ \ Heart(y) \mathrm{\ A\ } Has(x,y)$$

If we just replaced y with a constant, H, we would get

$$\forall x\ \ Person(x) \Rightarrow Heart(H) \mathrm{\ A\ } Has(x,H)$$

which says that everyone has the same heart H. We need to say that the heart they have is not necessarily shared, that is, it can be found by applying to each person a function that maps from person to heart:

$$\forall x\ \ Person(x) \Rightarrow Heart(F(x)) \mathrm{\ A\ } Has(x,F(x))$$

SKOLEM FUNCTION

where F is a function name that does not appear elsewhere in the KB. F is called a **Skolem function.** In general, the existentially quantified variable is replaced by a term that consists of a Skolem function applied to all the variables universally quantified *outside* the existential quantifier in question. Skolemization eliminates all existentially quantified variables, so we are now free to drop the universal quantifiers, because any variable must be universally quantified.

◇ **Distribute A over** $\vee$: $(a$ A $b)$ V c becomes $(a$ V c) A $(b$ V $c)$.

◇ **Flatten nested conjunctions and disjunctions:** $(a$ V $b)$ V c becomes $(a$ V b V c), and $(a$ A $b)$ A c becomes $(a$ A b A $c)$.

At this point, the sentence is in conjunctive normal form (CNF): it is a conjunction where every conjunct is a disjunction of literals. This form is sufficient for resolution, but it may be difficult for us humans to understand.

0 **Convert disjunctions to implications:** Optionally, you can take one more step to convert to implicative normal form. For each conjunct, gather up the negative literals into one list, the positive literals into another, and build an implication from them:
$(\neg a$ V $\neg b$ V c V $d)$becomes $(a$ A $b \Rightarrow c$ V $d)$

Example proof

We will now show how to apply the conversion procedure and the resolution refutation procedure on a more complicated example, which is stated in English as:

Jack owns a dog.
Every dog owner is an animal lover.
No animal lover kills an animal.
Either Jack or Curiosity killed the cat, who is named Tuna.
Did Curiosity kill the cat?

First, we express the original sentences (and some background knowledge) in first-order logic:

A. $\exists x \;\; Dog(x)$ A $Owns(Jack, x)$
B. $\forall x \;\; (3\, y \;\; Dog(y)$ A $Owns(x, y)) \Rightarrow AnimalLover(x)$
C. $\forall x \;\; AnimalLover(x) \Rightarrow \forall y \;\; Animal(y) \Rightarrow \neg Kills(x, y)$
D. $Kills(Jack, Tuna)$ V $Kills(Curiosity, Tuna)$
E. $Cat(Tuna)$
F. $\forall x \;\; Cat(x) \Rightarrow Animal(x)$

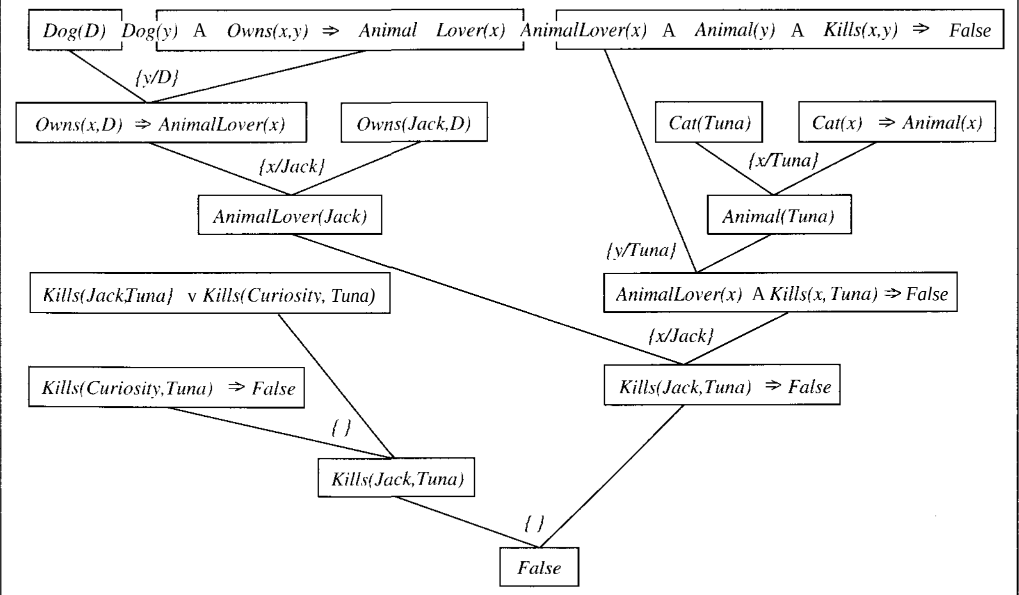
Now we have to apply the conversion procedure to convert each sentence to implicative normal form. We will use the shortcut of writing P instead of $True \Rightarrow P$:

A1. $Dog(D)$
A2. $Owns(Jack, D)$
B. $Dog(y)$ A $Owns(x, y) \Rightarrow AnimalLover(x)$
C. $AnimalLover(x)$A $Animal(y)$A $Kills(x,y) \Rightarrow False$
D. $Kills(Jack, Tuna)$ V $Kills(Curiosity, Tuna)$
E. $Cat(Tuna)$
F. $Cat(x) \Rightarrow Animal(x)$

The problem is now to show that *Kills*(*Curiosity*, *Tuna*) is true. We do that by assuming the negation, *Kills*(*Curiosity*, *Tuna*) $\Rightarrow$ *False*, and applying the resolution inference rule seven times, as shown in Figure 9.7. We eventually derive a contradiction, *False*, which means that the assumption must be false, and *Kills*(*Curiosity*, *Tuna*) is true after all. In English, the proof could be paraphrased as follows:

> Suppose Curiosity did *not* kill Tuna. We know that either Jack or Curiosity did, thus Jack must have. But Jack owns D, and D is a dog, so Jack is an animal lover. Furthermore, Tuna is a cat, and cats are animals, so Tuna is an animal. Animal lovers don't kill animals, so Jack couldn't have killed Tuna. But this is a contradiction, because we already concluded that Jack must have killed Tuna. Hence, the original supposition (that Curiosity did not kill Tuna) must be wrong, and we have proved that Curiosity *did* kill Tuna.

The proof answers the question "Did Curiosity kill the cat?" but often we want to pose more general questions, like "Who killed the cat?" Resolution can do this, but it takes a little more work to obtain the answer. The query can be expressed as $\exists\, w$ *Kills*(*w*, *Tuna*). If you repeat the proof tree in Figure 9.7, substituting the negation of this query, *Kills*(*w*, *Tuna*) $\Rightarrow$ *False* for the old query, you end up with a similar proof tree, but with the substitution $\{w/Curiosity\}$in one of the steps. So finding an answer to "Who killed the cat" is just a matter of lookingin the proof tree to find the binding of *w*. It is straightforward to maintain a composed unifier so that a solution is available as soon as a contradiction is found.



Dog(D)
Dog(y) A Owns(x,y) ⇒ Animal Lover(x)
AnimalLover(x) A Animal(y) A Kills(x,y) ⇒ False
{y/D}
Owns(x,D) ⇒ AnimalLover(x)
Owns(Jack,D)
Cat(Tuna)
Cat(x) ⇒ Animal(x)
{x/Jack}
{x/Tuna}
AnimalLover(Jack)
Animal(Tuna)
{y/Tuna}
Kills(Jack,Tuna} v Kills(Curiosity, Tuna)
AnimalLover(x) A Kills(x, Tuna) ⇒ False
{x/Jack}
Kills(Curiosity,Tuna) ⇒ False
Kills(Jack,Tuna) ⇒ False
{ }
Kills(Jack,Tuna)
{ }
False

Figure 9.7 A proof that Curiosity killed the cat, using resolution with refutation.

Dealing with equality

There is one problem that we have not dealt with so far, namely, finding appropriate inference rules for sentences containing the equality symbol. Unification does a good job of matching variables with other terms: $P(x)$ unifies with $P(A)$. But $P(A)$ and $P(B)$ fail to unify, even if the sentence $A = B$ is in the knowledge base. The problem is that unification only does a syntactic test based on the appearance of the argument terms, not a true semantic test based on the objects they represent. Of course, no semantic test is available because the inference system has no access to the objects themselves, but it should still be able to take advantage of what knowledge it has concerning the identities and differences among objects.

One way to deal with this is to axiomatize equality, by writing down its properties. We need to say that equality is reflexive, symmetric, and transitive, and we also have to say that we can substitute equals for equals in any predicate or function. So we need three basic axioms, and then one for each predicate and function:

$$
\begin{array}{l}
\forall x \quad x=x \\
\forall x,y \; x=y \Rightarrow y=x \\
\forall x,y,z \; x=y \land y=z \Rightarrow x=z \\
\forall x,y \; x=y \Rightarrow (P_1(x) \Leftrightarrow P_1(y)) \\
\forall x,y \; x=y \Rightarrow (P_2(x) \Leftrightarrow P_2(y)) \\
\qquad \vdots \\
\forall w,x,y,z \; w=y \land x=z \Rightarrow (F_1(w,x)=F_1(y,z)) \\
\forall w,x,y,z \; w=y \land x=z \Rightarrow (F_2(w,x)=F_2(y,z)) \\
\qquad \vdots
\end{array}
$$

The other way to deal with equality is with a special inference rule. The **demodulation** rule takes an equality statement $x=y$ and any sentence with a nested term that unifies with x and derives the same sentence with y substituted for the nested term. More formally, we can define the inference rule as follows:

DEMODULATION

0 **Demodulation:** For any terms x, y, and z, where UNIFY$(x,z) = 6$:

$$\frac{x=y, \quad (\ldots z \ldots)}{(\ldots \text{SUBST}(\theta, >') \ldots)}$$

If we write all our equalities so that the simpler term is on the right (e.g., $(x + 0) = 0$), then demodulation will do simplification, because it always replaces an expression on the left with

PARAMODULATION

one on the right. A more powerful rule called **paramodulation** deals with the case where we do not know $x = y$, but we do know, say, $x = y$ V $P(x)$.

Resolution strategies

We know that repeated applications of the resolution inference rule will find a proof if one exists, but we have no guarantee of the efficiency of this process. In this section we look at four of the strategies that have been used to guide the search toward a proof.

Unit preference

UNIT CLAUSE

This strategy prefers to do resolutions where one of the sentences is a single literal (also known as a **unit clause).** The idea behind the strategy is that we are trying to produce a very short sentence, *True* $\Rightarrow$ *False,* and therefore it might be a good idea to prefer inferences that produce shorter sentences. Resolving a 'unit sentence (such as *P)* with any other sentence (such as P A $Q \Rightarrow R$) always yields a sentence (in this case, $Q \Rightarrow R$) that is shorter than the other sentence. When the unit preference strategy was first tried for propositional inference in 1964, it led to a dramatic speedup, making it feasible to prove theorems that could not be handled without the preference. Unit preference by itself does not, however, reduce the branching factor in medium-sized problems enough to make them solvable by resolution. It is, nonetheless, a useful heuristic that can be combined with other strategies.

Set of support

SET OF SUPPORT

Preferences that try certain resolutions first are helpful, but in general it is more effective to try to eliminate some potential resolutions altogether. The set of support strategy does just that. It starts by identifying a subset of the sentences called the **set of support.** Every resolution combines a sentence from the set of support with another sentence, and adds the resolvent into the set of support. If the set of support is small relative to the whole knowledge base, this will cut the search space dramatically.

We have to be careful with this approach, because a bad choice for the set of support will make the algorithm incomplete. However, if we choose the set of support *S* so that the remainder of the sentences are jointly satisfiable, then set-of-support resolution will be complete. A common approach is to use the negated query as the set of support, on the assumption that the original knowledge base is consistent. (After all, if it is not consistent, then the fact that the query follows from it is vacuous.) The set-of-support strategy has the additional advantage of generating proof trees that are often easy for humans to understand, because they are goal-directed.

Input resolution

INPUT RESOLUTION

In the **input resolution** strategy, every resolution combines one of the input sentences (from the KB or the query) with some other sentence. The proofs in Figure 9.5 and Figure 9.6 use only input resolutions; they have the characteristic shape of a diagonal "spine" with single sentences combining onto the spine. Clearly, the space of proof trees of this shape is smaller than the space of all proof graphs. In Horn knowledge bases, Modus Ponens is a kind of input resolution strategy, because it combines an implication from the original KB with some other sentences. Thus, it should not be surprising that input resolution is complete for knowledge bases that are in Horn form, but incomplete in the general case.

LINEAR RESOLUTION

The **linear resolution** strategy is a slight generalization that allows *P* and *Q* to be resolved together if either *P* is in the original *KB* or if P is an ancestor of *Q* in the proof tree. Linear resolution is complete.

Subsumption

SUBSUMPTION

The **subsumption** method eliminates all sentences that are subsumed by (i.e., more specific than) an existing sentence in the KB. For example, if $P(x)$ is in the KB, then there is no sense in adding $P(A)$, and even less sense in adding $P(A)$ V $Q(B)$. Subsumption helps keep the KB small, which helps keep the search space small:

9.7 COMPLETENESS OF RESOLUTION

This section proves that resolution is complete. It can be safely skipped by those who are willing to take it on faith.

Before we show that resolution is complete, we need to be more precise about the particular flavor of completeness that we will establish. Resolution is **refutation-complete,** which means that *if a* set of sentences is unsatisfiable, then resolution will derive a contradiction. Resolution cannot be used to generate all logical consequences of a set of sentences, but it can be used to establish that a given sentence is entailed by the set. Hence, it can be used to find all answers to a given question, using the negated-goal method described before.

REFUTATION-COMPLETE

We will take it as given that any sentence in first-order logic (without equality) can be rewritten in normal form. This can be proved by induction on the form of the sentence, using atomic sentences as the base case (Davis and Putnam, 1960). Our goal therefore is to prove the following: *if S is an unsatisfiable set of sentences in clausal form, then the application of a finite number of resolution steps to S will yield a contradiction.*

Our proof sketch follows the original proof due to Robinson, with some simplifications from Genesereth and Nilsson (1987). The basic structure of the proof is shown in Figure 9.8, and is as follows:

1. We begin by observing that if S is unsatisfiable, then there exists a particular set of *ground instances* of the sentences of S, such that this set is also unsatisfiable (Herbrand's theorem).
2. We then show that resolution is complete for ground sentences (i.e., propositional logic). This is easy because the set of consequences of a propositional theory is always finite.
3. We then use a **lifting lemma** to show that, for any resolution proof using the set of ground sentences, there is a corresponding proof using the first-order sentences from which the ground sentences were obtained.

To carry out the first step, we will need three new concepts:

HERBRAND UNIVERSE

0 **Herbrand universe:** If S is a set of clauses, then H_S, the Herbrand universe of S, is the set of all ground terms constructible from the following:

   a. The function symbols in S, if any.
   b. The constant symbols in S, if any; if none, then the constant symbol A.

   For example, if S contains just the clause $P(x, F(x, A))$ A $Q(x, A) \Rightarrow R(x, B)$, then H_S is the following infinite set of ground sentences:

$$\{A, B, F(A, A), F(A, B), F(B, A), F(B, B), F(A, F(A, A)), F(A, F(A, B)), \ldots\}$$

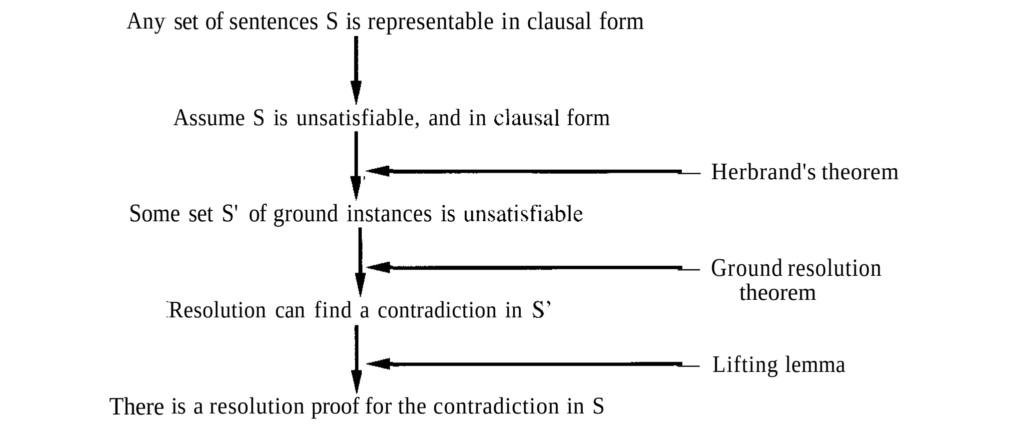

Any set of sentences S is representable in clausal form
Assume S is unsatisfiable, and in clausal form
Herbrand's theorem
Some set S' of ground instances is unsatisfiable
Ground resolution theorem
Resolution can find a contradiction in S'
Lifting lemma
There is a resolution proof for the contradiction in S

Figure 9.8 Structure of a completeness proof for resolution.

SATURATION

◇ **Saturation:** If S is a set of clauses, and P is a set of ground terms, then $P(S)$, the saturation of S with respect to P, is the set of all ground clauses obtained by applying all possible consistent substitutions for variables in S with ground terms in P.

HERBRAND BASE

◇ **Herbrand base:** The saturation of a set of clauses S with respect to its Herbrand universe is called the Herbrand base of S, written as $H_S(S)$. For example, if 5 contains just the clause given above, then

$$\begin{aligned}H_S(S) = \{ & P(A,F(A,A)) \wedge Q(A,A) \Rightarrow R(A,B),\\ & P(B,F(B,A)) \wedge Q(B,A) \Rightarrow R(B,B),\\ & P(F(A,A),F(F(A,A),A)) \wedge Q(F(A,A),A) \Rightarrow R(F(A,A),B),\\ & P(F(A,B),F(F(A,B),A)) \wedge Q(F(A,B),A) \Rightarrow R(F(A,B),B),\\ & \dots \}\end{aligned}$$

Notice that both the Herbrand universe and the Herbrand base can be infinite even if the original set of sentences S is finite.

HERBRAND'S THEOREM

These definitions allow us to state a form of **Herbrand's theorem** (Herbrand, 1930):

If a set of clauses S is unsatisfiable, then there exists a finite subset of $H_S(S)$ that is also unsatisfiable.

RESOLUTION CLOSURE

Let S' be this finite subset. Now it will be useful to introduce the **resolution closure** of S', which is the set of all clauses derivable by repeated application of the resolution inference step to clauses in S' or their derivatives. (To construct this closure, we could run a breadth-first search to completion using the resolution inference rule as the successor generator.) Let T be the resolution closure of S', and let $A_{S'} = \{A_1, A_2, \dots, A_k\}$ be the set of atomic sentences occurring in S'. Notice that because S' is finite, $A_{S'}$ must also be finite. And because the clauses in T are constructed

GÖDEL'S INCOMPLETENESS THEOREM

By slightly extending the language of first-order logic to allow for the **mathematical induction schema** in arithmetic, Godel was able to show, in his **incompleteness theorem,** that there are true aYithmetic sentences that cannot be proved.

The proof of the incompleteness theorem is somewhat beyond the scope of this book, occupying, as it does, at least 30 pages, but we can give a hint here. We begin with the logical theory of numbers. In this theory, there is a single constant, 0, and a single function, S (the successor function). In the intended model, 5(0) denotes 1, $S(S(0))$ denotes 2, and so on; the language therefore has names for all the natural numbers. The vocabulary also includes the function symbols +, $\times$, and *Expt* (exponentiation), and the usual set of logical connectives and quantifiers. The first step is to notice that the set of sentences that we can write in this language can be enumerated. (Imagine defining an alphabetical order on the symbols and then arranging in alphabetical order each of the sets of sentences of length 1, 2, and so on.) We can then number each sentence a with a unique natural number $\#\alpha$ (the **Godel number).** This is crucial: number theory contains a name for each of its own sentences. Similarly, we can number each possible proof P with a Godel number $G(P)$, because a proof is simply a finite sequence of sentences.

Now suppose we have a set A of sentences that are true statements about the natural numbers. Recalling that A can be named by a given set of integers, we can imagine writing in our language a sentence $\alpha(j, A)$ of the following sort:

> V i i is not the Godel number of a proof of the sentence whose Godel number is j, where the proof uses only premises in A.

Then let σ be the sentence $\alpha(\#\sigma, \text{A})$, that is, a sentence that states its own unprovability from A. (That this sentence always exists is true but not entirely obvious.)

Now we make the following ingenious argument. Suppose that a *is* provable from A; then a is false (because a says it cannot be proved). But then we have a false sentence that is provable from A, so A cannot consist of only true sentences—a violation of our premise. Therefore a is *not* provable from A. But this is exactly what σ itself claims; hence σ is a true sentence.

So, we have shown (barring 29 and a half pages) that for any set of true sentences of number theory, and in particular any set of basic axioms, there are other true sentences that *cannot* be proved from those axioms. This establishes, among other things, that we can never prove all the theorems of mathematics *within any given system of axioms.* Clearly, this was an important discovery for mathematics. Its significance for AI has been widely debated, beginning with speculations by Godel himself. We take up the debate in Chapter 26.

entirely from members of $A_{S'}$, T must be finite because only a finite number of distinct clauses can be constructed from a finite vocabulary of atomic sentences. To illustrate these definitions, we will use a slimmed-down example:

$$\begin{aligned} S' &= \{\ P(A),\ P(A) \Rightarrow Q(A),\ Q(A) \Rightarrow \mathit{False}\ \} \\ A_{S'} &= \{P(A), Q(A), \mathit{False}\} \\ T &= \{\ P(A), P(A) \Rightarrow Q(A),\ Q(A) \Rightarrow \mathit{False},\ Q(A),\ P(A) \Rightarrow \mathit{False}, \mathit{False}\ \} \end{aligned}$$

GROUND RESOLUTION THEOREM

Now we can state a completeness theorem for resolution on ground clauses. This is called the **ground resolution theorem:**

If a set of ground clauses is unsatisfiable, then the resolution closure of those clauses contains the clause *False.*

We prove this theorem by showing its contrapositive: if the closure T does *not* contain *False,* then S' is satisfiable; in fact, we can construct a satisfying assignment for the atomic sentences in S'. The construction procedure is as follows:

Pick an assignment *(True* or *False)* for each atomic sentence in Ay in some fixed order $A_1, \ldots, A_k$:

- If there is a clause in T containing the literal $\neg A_i$, such that all its other literals are false under the assignment chosen for $A_1, \ldots, A_{i-1}$, then assign A_i to be *False.*
- Otherwise, assign A_i to be *True.*

It is easy to show that the assignment so constructed will satisfy S', provided T is closed under resolution and does not contain the clause *False* (Exercise 9.10).

Now we have established that there is always a resolution proof involving some finite subset of the Herbrand base of *S.* The next step is to show that there is a resolution proof using the clauses of S itself, which are not necessarily ground clauses. We start by considering a single application of the resolution rule. Robinson's basic lemma implies the following fact:

Let C_1 and C_2 be two clauses with no shared variables, and let C'_1 and C'_2 be ground instances of $C\backslash$ and C_2. If C' is a resolvent of C'_1 and C'_2, then there exists a clause C such that (1) C is a resolvent of $C\backslash$ and C_2, and (2) C' is a ground instance of C.

LIFTING LEMMA

This is called a **lifting lemma,** because it lifts a proof step from ground clauses up to general first-order clauses. In order to prove his basic lifting lemma, Robinson had to invent unification and derive all of the properties of most general unifiers. Rather than repeat the proof here, we simply illustrate the lemma:

$$\begin{aligned} C_1 &= P(x, F(x,A)) \wedge Q(x,A) \Rightarrow R(x,B) \\ C_2 &= N(G(y),z) \Rightarrow P(H(y),z) \\ C'_1 &= P(H(B), F(H(B),A)) \wedge Q(H(B),A) \Rightarrow R(H(B),B) \\ C'_2 &= N(G(B), F(H(B),A)) \Rightarrow P(H(B), F(H(B),A)) \\ C' &= N(G(B), F(H(B),A)) \wedge Q(H(B),A) \Rightarrow R(H(B),B) \\ C &= N(G(y), F(H(y),A)) \wedge Q(H(y),A) \Rightarrow R(H(y),B) \end{aligned}$$

We see that indeed C' is a ground instance of C. In general, for C'_1 and C'_2 to have any resolvents, they must be constructed by first applying to C_1 and C_2 the most general unifier of a pair of complementary literals in $C\backslash$ and C_2. From the lifting lemma, it is easy to derive a similar statement about any sequence of applications of the resolution rule:

> For any clause C' in the resolution closure of S', there is a clause C in the resolution closure of S, such that C' is a ground instance of C and the derivation of C is the same length as the derivation of C'.

From this fact, it follows that if the clause *False* appears in the resolution closure of S', it must also appear in the resolution closure of S. This is because *False* cannot be a ground instance of any other clause. To recap: we have shown that if 5 is unsatisfiable, then there is a finite derivation of the clause *False* using the resolution rule.

The lifting of theorem proving from ground clauses to first-order clauses provided a vast increase in power. This derives from the fact that the first-order proof need instantiate variables only as far as necessary for the proof, whereas the ground-clause methods were required to examine a huge number of arbitrary instantiations.

9.8 SUMMARY

We have presented an analysis of logical inference in first-order logic, and a number of algorithms for doing it.

- A simple extension of the propositional inference rules allows the construction of proofs for first-order logic. Unfortunately, the branching factor for the quantifier is huge.
- **The** use of **unification** to identify appropriate substitutions for variables eliminates the 5 instantiation step in first-order proofs, making the process much more efficient.
- A generalized version **of Modus Ponens** uses unification to provide a natural and powerful inference rule, which can be used in a backward-chaining or forward-chaining algorithm.
- The canonical form for Modus Ponens is **Horn form:**

  $p\backslash \text{ A } \ldots \text{ A} p_n \Rightarrow q$, where p_i and q are atoms.

  This form cannot represent all sentences, and Modus Ponens is not a complete proof system.
- The generalized **resolution** inference rule provides a complete system for proof by refutation. It requires a normal form, but *any* sentence can be put into the form.
- Resolution can work with either **conjunctive normal form**—each sentence is a disjunction of literals—or **implicative normal form**—each sentence is of the form

  $p\backslash \text{ A } \ldots \text{ A} p_n \Rightarrow q\backslash \text{ V } \ldots \text{ V } q_m$, where p_i and q_i are atoms.
- Several strategies exist for reducing the search space of a resolution system without compromising completeness.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

SYLLOGISM

Logical inference was studied extensively in Greek mathematics. The type of inference most carefully studied by Aristotle was the **syllogism.** The syllogism is divided into "figures" and "moods," depending on the order of the terms (which we would call predicates) in the sentences, the degree of generality (which we would today interpret through quantifiers) applied to each term, and whether each term is negated. The most fundamental syllogism is that of the first mood of the first figure:

All *S* are *M*.
All *M* are *P*.
Therefore, all *S* are *P*.

Aristotle tried to prove the validity of other syllogisms by "reducing" them to those of the first figure. He was much less precise in describing what this "reduction" should involve than he was in characterizing the syllogistic figures and moods themselves.

Rigorous and explicit analysis of inference rules was one of the strong points of Megarian and Stoic propositional logic. The Stoics took five basic inference rules as valid without proof and then rigorously derived all others from these five. The first and most important of the five rules was the one known today as Modus Ponens. The Stoics were much more precise about what was meant by derivation than Aristotle had been about what was meant by reduction of one syllogism to another. They also used the "Principle of Conditionalization," which states that if q can be inferred validly from p, then the conditional "$p \Rightarrow q$" is logically true. Both these principles figure prominently in contemporary logic.

Inference rules, other than logically valid schemas, were not a major focus either for Boole or for Frege. Boole's logic was closely modeled on the algebra of numbers. It relied mainly on the **equality substitution** inference rule, which allows one to conclude $P(t)$ given $P(s)$ and $s = t$. Logically valid schemas were used to obtain equations to which equality substitution could be applied. Whereas Frege's logic was much more general than Boole's, it too relied on an abundance of logically valid schemas plus a single inference rule that had premises—in Frege's case, this rule was Modus Ponens. Frege took advantage of the fact that the effect of an inference rule of the form "From p infer q" can be simulated by applying Modus Ponens to p along with a logically valid schema $p \Rightarrow q$. This "axiomatic" style of exposition, using Modus Ponens plus a number of logically valid schemas, was employed by a number of logicians after Frege; most notably, it was used in *Principia Mathematica* (Whitehead and Russell, 1910).

One of the earliest types of systems (after Frege) to focus prominently on inference rules was **natural deduction,** introduced by Gerhard Gentzen (1934) and by Stanisław Jáskowski (1934). Natural deduction is called "natural" because it does not require sentences to be subjected to extensive preprocessing before it can be applied to them (as many other proof procedures do) and because its inference rules are thought to be more intuitive than, say, the resolution rule. Natural deduction makes frequent use of the Principle of Conditionalization. The objects manipulated by Gentzen's inference rules are called *sequents.* A sequent can be regarded either as a logical argument (a pair of a set of premises and a set of alternative conclusions, intended to be an instance of some valid rule of inference) or as a sentence in implicative normal form. Prawitz (1965) offers a book-length treatment of natural deduction. Gallier (1986) uses Gentzen sequents to expound the theoretical underpinnings of automated deduction.

Conjunctive normal form and disjunctive normal form for propositional formulas were known to Schröder (1877), although the principles underlying the construction of these forms go back at least to Boole. The use of clausal form (conjunctive normal form for first-order logic) depends upon certain techniques for manipulating quantifiers, in particular skolemization. Whitehead and Russell (1910) expounded the so-called *rules ofpassage* (the actual term is from Herbrand (1930)) that are used to move quantifiers to the front of formulas. (Moving all the quantifiers to the front of a formula is called *prenexing* the formula, and a formula with all the quantifiers in front is said to be in **prenex form.)** Horn form was introduced by Alfred Horn (1951). What we have called "implicative normal form" was used (with a right-to-left implication symbol) in Robert Kowalski's (1979b) *Logic for Problem Solving,* and this way of writing clauses is sometimes called "Kowalski form."

Skolem constants and Skolem functions were introduced, appropriately enough, by Thoralf Skolem (1920). The general procedure for skolemization is given in (Skolem, 1928), along with the important notion of the Herbrand universe.

Herbrand's theorem, named after the French logician Jacques Herbrand (1930), has played a vital role in the development of automated reasoning methods, both before and after Robinson's introduction of resolution. This is reflected in our reference to the "Herbrand universe" rather than the "Skolem universe," even though Skolem really invented this concept (and indeed Herbrand does not explicitly use Skolem functions and constants, but a less elegant although roughly equivalent device). Herbrand can also be regarded as the inventor of unification, because a variant of the unification algorithm occurs in (Herbrand, 1930).

First-order logic was shown to have complete proof procedures by Gödel (1930), using methods similar to those of Skolem and Herbrand. Alan Turing (1936) and Alonzo Church (1936) simultaneously showed, using very different proofs, that validity in first-order logic was not decidable. The excellent text by Enderton (1972) explains all of these results in a rigorous yet moderately understandable fashion.

The first mechanical device to carry out logical inferences was constructed by the third Earl of Stanhope (1753-1816). The Stanhope Demonstrator could handle syllogisms and certain inferences in the theory of probability. The first mechanical device to carry out inferences in *mathematical* logic was William Stanley Jevons's "logical piano," constructed in 1869. Jevons was one of the nineteenth-century logicians who expanded and improved Boole's work; the logical piano carried out reasoning in Boolean logic. An entertaining and instructive history of these and other early mechanical devices for reasoning is given by Martin Gardner (1968).

The first published results from research on automated deduction using electronic computers were those of Newell, Shaw, and Simon (1957) on the Logic Theorist. This program was based on an attempt to model human thought processes. Martin Davis (1957) had actually designed a program that came up with a proof in 1954, but the Logic Theorist's results were published slightly earlier. Both Davis's 1954 program and the Logic Theorist were based on somewhat ad hoc methods that did not strongly influence later automated deduction.

It was Abraham Robinson who suggested attempting to use Herbrand's Theorem to generate proofs mechanically. Gilmore (1960) wrote a program that uses Robinson's suggestion in a way influenced by the "Beth tableaux" method of proof (Beth, 1955). Davis and Putnam (1960) introduced clausal form, and produced a program that attempted to find refutations by substituting members of the Herbrand universe for variables to produce ground clauses and then looking for propositional inconsistencies among the ground clauses. Prawitz (1960) developed the key idea of letting the quest for propositional inconsistency drive the search process, and generating terms from the Herbrand universe only when it was necessary to do so in order to establish propositional inconsistency. After further development by other researchers, this idea led J. A. Robinson (no relation) to develop the resolution method (Robinson, 1965), which used unification in its modern form to allow the demonstration of propositional inconsistency without necessarily making explicit use of terms from the Herbrand universe. The so-called "inverse method" developed at about the same time by the Soviet researcher S. Maslov (1964; 1967) is based on principles somewhat different from Robinson's resolution method but offers similar computational advantages in avoiding the unnecessary generation of terms in the Herbrand universe. The relations between resolution and the inverse method are explored by Maslov (1971) and by Kuehner (1971).

The demodulation rule described in the chapter was intended to eliminate equality axioms by combining equality substitution with resolution, and was introduced by Wos (1967). Term rewriting systems such as the Knuth-Bendix algorithm (Knuth and Bendix, 1970) are based on demodulation, and have found wide application in programming languages. The paramodulation inference rule (Wos and Robinson, 1968) is a more general version of demodulation that provides a complete proof procedure for first-order logic with equality.

In addition to demodulation and paramodulation for equality reasoning, other special-purpose inference rules have been introduced to aid reasoning of other kinds. Boyer and Moore (1979) provide powerful methods for the use of mathematical induction in automated reasoning, although their logic unfortunately lacks quantifiers and does not quite have the full power of first-order logic. Stickel's (1985) "theory resolution" and Manna and Waldinger's (1986) method of "special relations" provide general ways of incorporating special-purpose inference rules into a resolution-style framework.

A number of control strategies have been proposed for resolution, beginning with the unit preference strategy (Wos *et al.*, 1964). The set of support strategy was proposed by Wos *et al.* (1965), to provide a degree of goal-directedness in resolution. *Linear resolution* first appeared in (Loveland, 1968). Wolfgang Bibel (1981) developed the **connection method** which allows complex deductions to be recognized efficiently. Developments in resolution control strategies, and the accompanying growth in the understanding of the relationship between completeness and syntactic restrictions on clauses, contributed significantly to the development of **logic programming** (see Chapter 10). Genesereth and Nilsson (1987, Chapter 5) provide a short but thorough analysis of a wide variety of control strategies.

There are a number of good general-purpose introductions to automated deduction and to the theory of proof and inference; some were mentioned in Chapter 7. Additional textbooks on matters related to completeness and undecidability include *Computability and Logic* (Boolos and Jeffrey, 1989), *Metalogic* (Hunter, 1971), and (for an entertaining and unconventional, yet highly rigorous approach) A *Course in Mathematical Logic* (Manin, 1977). Many of the most important papers from the turn-of-the-century development of mathematical logic are to be found in *From Frege to Gödel: A Source Book in Mathematical Logic* (van Heijenoort, 1967). The journal of record for the field of pure mathematical logic (as opposed to automated deduction) is *The Journal of Symbolic Logic.*

Textbooks geared toward automated deduction include (in addition to those mentioned in Chapter 7) the classic *Symbolic Logic and Mechanical Theorem Proving* (Chang and Lee, 1973) and, more recently, *Automated Reasoning: Introduction and Applications* (Wos *et al.*, 1992). The two-volume anthology *Automation of Reasoning* (Siekmann and Wrightson, 1983) includes many important papers on automated deduction from 1957 to 1970. The historical summaries prefacing each volume provide a concise yet thorough overview of the history of the field. Further important historical information is available from Loveland's "Automated Theorem Proving: A Quarter-Century Review" (1984) and from the bibliography of (Wos *et al.*, 1992).

The principal journal for the field of theorem proving is the *Journal of Automated Reasoning*; important results are also frequently reported in the proceedings of the annual Conferences on Automated Deduction (CADE). Research in theorem proving is also strongly related to the use of logic in analyzing programs and programming languages, for which the principal conference is Logic in Computer Science.

EXERCISES

9.1 For each of the following pairs of atomic sentences, give the most general unifier, if it exists.

a. $P(A,B,B)$, $P(x,y,z)$.
b. $Q(y,G(A,B))$, $Q(G(x,x),y)$.
c. $Older(Father(y),y), Older(Father(x),John)$.
d. $Knows(Father(y),y), Knows(x,x)$.

9.2 One might suppose that we can avoid the problem of variable conflict in unification by standardizing apart all of the sentences in the knowledge base once and for all. Show that for some sentences, this approach cannot work. *(Hint:* Consider a sentence, one part of which unifies with another.)

9.3 Show that the final state of the knowledge base after a series of calls to FORWARD-CHAIN is independent of the order of the calls. Does the number of inference steps required depend on the order in which sentences are added? Suggest a useful heuristic for choosing an order.

9.4 Write down logical representations for the following sentences, suitable for use with Generalized Modus Ponens:

a. Horses, cows, and pigs are mammals.
b. An offspring of a horse is a horse.
c. Bluebeard is a horse.
d. Bluebeard is Charlie's parent.
e. Offspring and parent are inverse relations.
f. Every mammal has a parent.

9.5 In this question we will use the sentences you wrote in Exercise 9.4 to answer a question using a backward-chaining algorithm.

a. Draw the proof tree generated by an exhaustive backward-chaining algorithm for the query $\exists h \;\; Horse(h)$.
b. What do you notice about this domain?
c. How many solutions for h actually follow from your sentences?
d. Can you think of a way to find all of them? *(Hint:* You might want to consult (Smith *et al.*, 1986).)

9.6 A popular children's riddle is "Brothers and sisters have I none, but that man's father is my father's son." Use the rules of the family domain (Chapter 7) to show who that man is. You may use any of the inference methods described in this chapter.

9.7 How can resolution be used to show that a sentence is

a. Valid?
b. Unsatisfiable?

9.8 From "Horses are animals," it follows that "The head of a horse is the head of an animal." Demonstrate that this inference is valid by carrying out the following steps:

a. Translate the premise and the conclusion into the language of first-order logic. Use three predicates: $HeadOf(h, x)$, $Horse(x)$, and $Animal(x)$.
b. Negate the conclusion, and convert the premise and the negated conclusion into conjunctive normal form.
c. Use resolution to show that the conclusion follows from the premise.

9.9 Here are two sentences in the language of first-order logic:

(A): $\forall x \;\; \exists y \;\; (x \geq y)$
(B): $\exists y \;\; \forall x \;\; (x \geq y)$

a. Assume that the variables range over all the natural numbers $0, 1, 2, \ldots, \infty$, and that the "$>$" predicate means "greater than or equal to." Under this interpretation, translate these sentences into English.
b. Is (A) true under this interpretation?
c. Is (B) true under this interpretation?
d. Does (A) logically entail (B)?
e. Does (B) logically entail (A)?
f. Try to prove that (A) follows from (B) using resolution. Do this even if you think that (B) does not logically entail (A); continue until the proof breaks down and you cannot proceed (if it does break down). Show the unifying substitution for each resolution step. If the proof fails, explain exactly where, how, and why it breaks down.
g. Now try to prove that (B) follows from (A).

9.10 In this exercise, you will complete the proof of the ground resolution theorem given in the chapter. The proof rests on the claim that if T is the resolution closure of a set of ground clauses S', and T does not contain the clause *False,* then a satisfying assignment can be constructed for S' using the construction given in the chapter. Show that the assignment does indeed satisfy S', as claimed.