contain the empty clause, then S is satisfiable. In fact, we can construct a model for S with suitable truth values for P₁, . . . , Pk. The construction procedure is as follows:

For i from 1 to k,
- If a clause in RC(S) contains the literal and all its other literals are false under the assignment chosen for P₁, . . . , P_{i-1}, then assign false to Pi.
- Otherwise, assign true to P_i.

This assignment to P₁, . . . , Pk is a model of S. To see this, assume the opposite—that, at some stage i in the sequence, assigning symbol Pi causes some clause C to become false. For this to happen, it must be the case that all the other literals in C must already have been falsified by assignments to P_i. Thus, C must now look like either (false ∨ false ∨ . . . ∨ false ∨ P_i) or like (false ∨ false ∨ . . . false) ∨ ¬P_i. If just one of these two is in RC(S), then the algorithm will assign the appropriate truth value to PQ to make C true, so C can only be falsified if both of these clauses are in RC(S). Now, since RC(S) is closed under resolution, it will contain the resolvent of these two clauses, and that resolvent will have all of its literals already falsified by the assignments to Pi’s. This contradicts our assumption that the first falsified clause appears at stage i. Hence, we have proved that the construction never falsifies a clause in RC(S), that is, it produces a model of RC(S) and thus a model of S itself (since S is contained in RC(S)).

7.5.3 Horn clauses and definite clauses

The completeness of resolution makes it a very important inference method. In many practical situations, however, the full power of resolution is not needed. Some real-world knowledge bases satisfy certain restrictions on the form of sentences they contain, which enables them to use a more restricted and efficient inference algorithm.

One such restricted form is the definite clause, which is a disjunction of literals of which exactly one is positive. For example, the clause \( \neg \text{Breeze} \lor \text{B} \lor \text{P}_1 \lor \text{P}_2 \lor \text{P}_3 \) is a definite clause, whereas \( \neg \text{Breeze} \lor \text{P}_1 \lor \text{P}_2 \lor \text{P}_3 \) is not.

Slightly more general is the Horn clause, which is a disjunction of literals of which at most one is positive. So all definite clauses are Horn clauses, as are clauses with no positive literals; these are called goal clauses. Horn clauses are closed under resolution. If you resolve two Horn clauses, you get back a Horn clause.

Knowledge bases containing only definite clauses are interesting for three reasons:

1. Every definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal. (See Exercise 7.13.) For example, the definite clause \( \neg \text{Breeze} \lor \text{B} \lor \text{P}_1 \lor \text{P}_2 \lor \text{P}_3 \) can be written as the implication \( \text{L}_{1,1} \lor \text{A Breeze} \Rightarrow \text{B}_{1,1} \). In this implication form, the sentence is easier to understand: it says that if the agent is in \([1,1]\) and there is a breeze, then \([1,1]\) is breezy. In Horn form, the premise is called the body and the conclusion is called the head. A sentence consisting of a single positive literal, such as \( \text{F}_{1,1} \), is called a fact. It too can be written in implication form as \( \text{True} \Rightarrow \text{L}_{1,1} \), but it is simpler to write just \( \text{L}_{1,1} \).
Section 7.5. Propositional Theorem Proving

2. Inference with Horn clauses can be done through the forward-chaining and backward-chaining algorithms, which we explain next. Both of these algorithms are natural, in that the inference steps are obvious and easy for humans to follow. This type of inference is the basis for logic programming, which is discussed in Chapter 9.

1 Deciding entailment with Horn clauses can be done in time that is linear in the size of the knowledge base—a pleasant surprise.

7.5.4 Forward and backward chaining

The forward-chaining algorithm \( \text{PL-FC-ENTAIL?}(KB,q) \) determines if a single proposition symbol \( q \)—the query—is entailed by a knowledge base of definite clauses. It begins from known facts (positive literals) in the knowledge base. If all the premises of an implication are known, then its conclusion is added to the set of known facts. For example, if \( L_{1:1} \) and Breeze are known and \( (L_{1:1} \land \text{Breeze}) \rightarrow B_{1:1} \) is in the knowledge base, then \( B_{1:1} \) can be added. This process continues until the query \( q \) is added or until no further inferences can be made. The detailed algorithm is shown in Figure 7.15; the main point to remember is that it runs in linear time.

The best way to understand the algorithm is through an example and a picture. Figure 7.16(a) shows a simple knowledge base of Horn clauses with \( A \) and \( B \) as known facts. Figure 7.16(b) shows the same knowledge base drawn as an AND–OR graph (see Chapter 4). In AND–OR graphs, multiple links joined by an arc indicate a conjunction—every link must be proved—while multiple links without an arc indicate a disjunction—any link can be proved. It is easy to see how forward chaining works in the graph. The known leaves (here, \( A \) and \( B \)) are set, and inference propagates up the graph as far as possible. Whenever a conjunction appears, the propagation waits until all the conjuncts are known before proceeding. The reader is encouraged to work through the example in detail.