function PL-FC-ENTAILs(KB, q) returns true or false
inputs: KB, the knowledge base, a set of propositional definite clauses
        q, the query, a proposition symbol
        count — a table, where count[c] is the number of symbols in c’s premise
        inferred — a table, where inferred[s] is initially false for all symbols
        agenda — a queue of symbols, initially symbols known to be true in KB

while agenda is not empty do
    p ← POP(agenda)
    if p = q then return true
    if inferred[p] = false then
        inferred[p] ← true
        for each clause c in KB where p is in c.PREmise do.
            decrement count[c]
            if count[c] = 0 then add c.CONCLUSION to agenda
    return false

Figure 7.15 The forward-chaining algorithm for propositional logic. The agenda keeps track of symbols known to be true but not yet "processed." The count table keeps track of how many premises of each implication are as yet unknown. Whenever a new symbol p from the agenda is processed, the count is reduced by one for each implication in whose premise p appears (easily identified in constant time with appropriate indexing.) If a count reaches zero, all the premises of the implication are known, so its conclusion can be added to the agenda. Finally, we need to keep track of which symbols have been processed; a symbol that is already in the set of inferred symbols need not be added to the agenda again. This avoids redundant work and prevents loops caused by implications such as $P \land \neg Q$ and $Q = P$.

It is easy to see that forward chaining is sound: every inference is essentially an application of Modus Ponens. Forward chaining is also complete: every entailed atomic sentence will be derived. The easiest way to see this is to consider the final state of the inferred table (after the algorithm reaches a fixed point where no new inferences are possible). The table contains true for each symbol inferred during the process, and false for all other symbols. We can view the table as a logical model; moreover, every definite clause in the original KB is true in this model. To see this, assume the opposite, namely that some clause at $A \land \ldots \land A \land \neg q$ is false in the model. Then $q_1 \land \ldots \land q_k$ must be true in the model and $b$ must be false in the model. But this contradicts our assumption that the algorithm has reached a fixed point! We can conclude, therefore, that the set of atomic sentences inferred at the fixed point defines a model of the original KB. Furthermore, any atomic sentence $q$ that is entailed by the KB must be true in all its models and in this model in particular. Hence, every entailed atomic sentence $q$ must be inferred by the algorithm.

Forward chaining is an example of the general concept of data-driven reasoning—that is, reasoning in which the focus of attention starts with the known data. It can be used within an agent to derive conclusions from incoming percepts, often without a specific query in mind. For example, the wumpus agent might TELL its percepts to the knowledge base using
an incremental forward-chaining algorithm in which new facts can be added to the agenda to initiate new inferences. In humans, a certain amount of data-driven reasoning occurs as new information arrives. For example, if I am indoors and hear rain starting to fall, it might occur to me that the picnic will be canceled. Yet it will probably not occur to me that the seventeenth petal on the largest rose in my neighbor's garden will get wet; humans keep forward chaining under careful control, lest they be swamped with irrelevant consequences.

The backward-chaining algorithm, as its name suggests, works backward from the query. If the query \( q \) is known to be true, then no work is needed. Otherwise, the algorithm finds those implications in the knowledge base whose conclusion is \( q \). If all the premises of one of those implications can be proved true (by backward chaining), then \( q \) is true. When applied to the query \( Q \) in Figure 7.16, it works back down the graph until it reaches a set of known facts, \( A \) and \( B \), that forms the basis for a proof. The algorithm is essentially identical to the AND-OR-GRAPH-SEARCH algorithm in Figure 4A1. As with forward chaining, an efficient implementation runs in linear time.

Backward chaining is a form of goal-directed reasoning. It is useful for answering specific questions such as "What shall I do now?" and "Where are my keys?" Often, the cost of backward chaining is much less than linear in the size of the knowledge base, because the process touches only relevant facts.

7.6 EFFECTIVE PROPOSITIONAL MODEL CHECKING

In this section, we describe two families of efficient algorithms for general propositional inference based on model checking: one approach based on backtracking search, and one on local hill-climbing search. These algorithms are part of the "technology" of propositional logic. This section can be skimmed on a first reading of the chapter.