9.3 FORWARD CHAINING

A forward-chaining algorithm for propositional definite clauses was given in Section 7.5. The idea is simple: start with the atomic sentences in the knowledge base and apply Modus Ponens in the forward direction, adding new atomic sentences, until no further inferences can be made. Here, we explain how the algorithm is applied to first-order definite clauses. Definite clauses such as Situation = Response are especially useful for systems that make inferences in response to newly arrived information. Many systems can be defined this way, and forward chaining can be implemented very efficiently.

9.3.1 First order definite clauses

First-order definite clauses closely resemble propositional definite clauses (page 256): they are disjunctions of literals of which exactly one is positive. A definite clause either is atomic or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal. The following are first-order definite clauses:

\[ \text{King}(x) \lor \text{Greedy}(x) \lor \text{Evil}(x) \]
\[ \text{King}(\text{John}) \]
\[ \text{Greedy}(\text{Bob}) \]

Unlike propositional literals, first-order literals can include variables, in which case those variables are assumed to be universally quantified. (Typically, we omit universal quantifiers when writing definite clauses.) Not every knowledge base can be converted into a set of definite clauses because of the single-positive-literal restriction, but many can. Consider the following problem:

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.

We will prove that West is a criminal, First, we will represent these facts as first-order definite clauses. The next section shows how the forward-chaining algorithm solves the problem.

\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x) \]  \hspace{1cm} (9.3)

"Novo has some missiles." The sentence \( \exists x \text{ Owns}(\text{Novo}, x) \land \text{Missile}(x) \) is transformed into two definite clauses by Existential Instantiation, introducing a new constant \( M_1 \):

\[ \text{Owns}(\text{Novo}, M_1) \]  \hspace{1cm} (9.4)
\[ \text{Missile}(M_1) \]  \hspace{1cm} (9.5)

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

\[ \text{Missile}(x) \land \text{Owns}(\text{Novo}, x) = 5' \land \text{West}, x \text{ Nono} \]  \hspace{1cm} (9.6)

We will also need to know that missiles are weapons

\[ \text{Missile}(x) \land \text{Weapon}(x) \]  \hspace{1cm} (9.7)
and we must know that an enemy of America counts as "hostile:

\[ \text{Enemy}(x, \text{America}) = \text{Hostile}(x) \]  

(9.8)

"West, who is American . . .":

\[ \text{American}(\text{West}) \]  

(9.9)

"The country None, an enemy of America . . .

\[ \text{Enemy}(\text{None}, \text{America}) \]  

(9.10)

This knowledge base contains no function symbols and is therefore an instance of the class of Datalog knowledge bases. Datalog is a language that is restricted to first-order definite clauses with no function symbols. Datalog gets its name because it can represent the type of statements typically made in relational databases. We will see that the absence of function symbols makes inference much easier.

### 9.3.2 A simple forward-chaining algorithm

The first forward-chaining algorithm we consider is a simple one, shown in Figure 9.3. Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts. The process repeats until the query is answered (assuming that just one answer is required) or no new facts are added. Notice that a fact is not "new" if it is just a renaming of a known fact. One sentence is a renaming of another if they are identical except for the names of the variables. For example, \( \text{Likes}(x, \text{Ice Cream}) \) and \( \text{Likes}(y, \text{Ice Cream}) \) are renamings of each other because they differ only in the choice of \( x \) or \( y \); their meanings are identical: everyone likes ice cream.

We use our crime problem to illustrate how FOL-FC-Ask works. The implication sentences are (9.3), (9.6), (9.7), and (9.8). Two iterations are required:

- On the first iteration, rule (9.3) has unsatisfied premises. Rule (9.6) is satisfied with \( \{x/M1\} \), and \( \text{Sells}(\text{West}, M1, \text{Nano}) \) is added. Rule (9.7) is satisfied with \( \{x/M1\} \), and \( \text{Weapon}(M1) \) is added. Rule (9.8) is satisfied with \( \{x/\text{None}\} \), and \( \text{Hostile}(\text{None}) \) is added.

- On the second iteration, rule (9.3) is satisfied with \( \{a/\text{West}, y/M1, z/\text{None}\} \), and \( \text{Criminal}(\text{West}) \) is added.

Figure 9.4 shows the proof tree that is generated. Notice that no new inferences are possible at this point because every sentence that could be concluded by forward chaining is already contained explicitly in the KB. Such a knowledge base is called a fixed point of the inference process. Fixed points reached by forward chaining with first-order definite clauses are similar to those for propositional forward chaining (page 258); the principal difference is that a first-order fixed point can include universally quantified atomic sentences.

FOL-FC-Ask is easy to analyze. First, it is sound, because every inference is just an application of Generalized Modus Ponens, which is sound. Second, it is complete for definite clause knowledge bases; that is, it answers every query whose answers are entailed by any knowledge base of definite clauses. For Datalog knowledge bases, which contain no function symbols, the proof of completeness is fairly easy. We begin by counting the number of