Chapter 2

Definite Logic Programs

2.1 Definite Clauses

The idea of logic programming is to use a computer for drawing conclusions from declarative descriptions. Such descriptions — called logic programs — consist of finite sets of logic formulas. Thus, the idea has its roots in the research on *automatic theorem proving*. However, the transition from experimental theorem proving to applied logic programming requires improved efficiency of the system. This is achieved by introducing restrictions on the language of formulas — restrictions that make it possible to use the relatively simple and powerful inference rule called the *SLD-resolution principle*. This chapter introduces a restricted language of *definite logic programs* and in the next chapter their computational principles are discussed. In subsequent chapters a more unrestrictive language of so-called *general* programs is introduced. In this way the foundations of the programming language Prolog are presented.

To start with, attention will be restricted to a special type of *declarative* sentences of natural language that describe positive *facts* and *rules*. A sentence of this type either states that a relation holds between individuals (in case of a fact), or that a relation holds between individuals *provided* that some other relations hold (in case of a rule). For example, consider the sentences:

(i) "Tom is John's child"

(ii) "Ann is Tom's child"

(iii) "John is Mark's child"

(iv) "Alice is John's child"

(v) "The grandchild of a person is a child of a child of this person"

These sentences may be formalized in two steps. First atomic formulas describing facts are introduced:

$$\begin{aligned} &child(tom, john) && (1)\\ &child(ann, tom) && (2)\\ &child(john, mark) && (3)\\ &child(alice, john) && (4) \end{aligned}$$

Applying this notation to the final sentence yields:

"For all X and Y, $grandchild(X, Y)$ if
there exists a Z such that $child(X, Z)$ and $child(Z, Y)$" (5)

This can be further formalized using quantifiers and the logical connectives "$\supset$" and "$\wedge$", but to preserve the natural order of expression the implication is reversed and written "$\leftarrow$":

$$\forall X\, \forall Y\, (grandchild(X, Y) \leftarrow \exists\, Z\, (child(X, Z) \wedge child(Z, Y))) \qquad (6)$$

This formula can be transformed into the following equivalent forms using the equivalences given in connection with Definition 1.15:

$$\begin{aligned} &\forall X\, \forall Y\, (grandchild(X, Y) \vee \neg\, \exists\, Z\, (child(X, Z) \wedge child(Z, Y)))\\ &\forall X\, \forall Y\, (grandchild(X, Y) \vee \forall\, Z\, \neg\, (child(X, Z) \wedge child(Z, Y)))\\ &\forall X\, \forall Y\, \forall Z\, (grandchild(X, Y) \vee \neg\, (child(X, Z) \wedge child(Z, Y)))\\ &\forall X\, \forall Y\, \forall Z\, (grandchild(X, Y) \leftarrow (child(X, Z) \wedge child(Z, Y))) \end{aligned}$$

We now focus attention on the language of formulas exemplified by the example above. It consists of formulas of the form:

$$A_0 \leftarrow A_1 \wedge \cdots \wedge A_n \quad \text{(where } n \geq 0)$$

or equivalently:

$$A_0 \vee \neg A_1 \vee \cdots \vee \neg A_n$$

where $A_0, \ldots, A_n$ are atomic formulas and all variables occurring in a formula are (implicitly) universally quantified over the whole formula. The formulas of this form are called *definite clauses.* Facts are definite clauses where $n = 0$. (Facts are sometimes called unit-clauses.) The atomic formula A_0 is called the *head* of the clause whereas $A_1 \wedge \cdots \wedge A_n$ is called its *body.*

The initial example shows that definite clauses use a restricted form of existential quantification — the variables that occur only in body literals are existentially quantified over the body (though formally this is equivalent to universal quantification on the level of clauses).

2.2 Definite Programs and Goals

The logic formulas derived above are special cases of a more general form, called *clausal form.*

Definition 2.1 (Clause) A *clause* is a formula $\forall(L_1 \vee \cdots \vee L_n)$ where each L_i is an atomic formula (a positive literal) or the negation of an atomic formula (a negative literal). ▮

As seen above, a *definite clause* is a clause that contains exactly one positive literal. That is, a formula of the form:

$$\forall(A_0 \vee \neg A_1 \vee \cdots \vee \neg A_n)$$

The notational convention is to write such a definite clause thus:

$$A_0 \leftarrow A_1, \ldots, A_n \quad (n \geq 0)$$

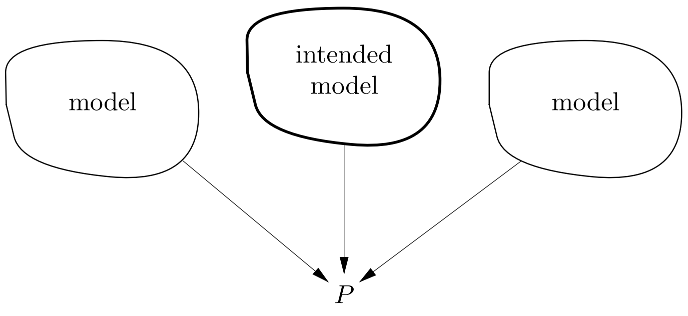
If the body is empty (i.e. if $n = 0$) the implication arrow is usually omitted. Alternatively the empty body can be seen as a nullary connective $\blacksquare$ which is true in every interpretation. (Symmetrically there is also a nullary connective $\square$ which is false in every interpretation.) The first kind of logic program to be discussed are programs consisting of a finite number of definite clauses:

Definition 2.2 (Definite programs) A *definite program* is a finite set of definite clauses. ▮

To explain the use of logic formulas as programs, a general view of logic programming is presented in Figure 2.1. The programmer attempts to describe the *intended model* by means of declarative sentences (i.e. when writing a program he has in mind an algebraic structure, usually infinite, whose relations are to interpret the predicate symbols of the program). These sentences are definite clauses — facts and rules. The program is a set of logic formulas and it may have many models, including the intended model (Figure 2.1(a)). The concept of intended model makes it possible to discuss correctness of logic programs — a program P is incorrect iff the intended model is not a model of P. (Notice that in order to prove programs to be correct or to test programs it is necessary to have an alternative description of the intended model, independent of P.)

The program will be used by the computer to draw conclusions about the intended model (Figure 2.1(b)). However, the only information available to the computer about the intended model is the program itself. So the conclusions drawn must be true in *any* model of the program to guarantee that they are true in the intended model (Figure 2.1(c)). In other words — the soundness of the system is a necessary condition. This will be discussed in Chapter 3. Before that, attention will be focused on the practical question of how a logic program is to be used.

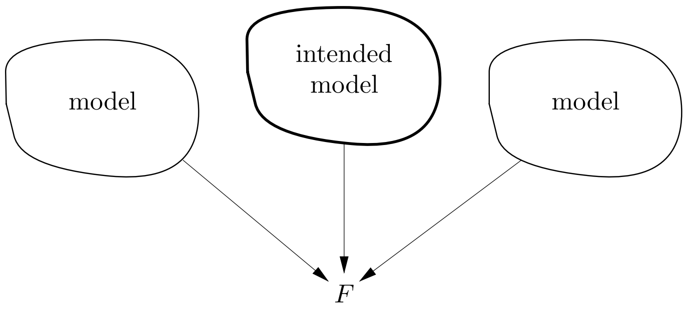
The set of logical consequences of a program is infinite. Therefore the user is expected to *query* the program selectively for various aspects of the intended model. There is an analogy with relational databases — facts explicitly describe elements of the relations while rules give intensional characterization of some other elements.



(a)

$$P \vdash F$$

(b)



(c)

Figure 2.1: General view of logic programming

Since the rules may be recursive, the relation described may be infinite in contrast to the traditional relational databases. Another difference is the use of variables and compound terms. This chapter considers only "queries" of the form:

$$\forall(\neg(A_1 \land \cdots \land A_m))$$

Such formulas are called *definite goals* and are usually written as:

$$\leftarrow A_1, \ldots, A_m$$

where A_i's are atomic formulas called *subgoals*. The goal where $m = 0$ is denoted $\Box$[1] and called the *empty* goal. The logical meaning of a goal can be explained by referring to the equivalent universally quantified formula:

$$\forall X_1 \cdots \forall X_n \, \neg(A_1 \land \cdots \land A_m)$$

where $X_1, \ldots, X_n$ are all variables that occur in the goal. This is equivalent to:

$$\neg \, \exists X_1 \cdots \exists X_n \, (A_1 \land \cdots \land A_m)$$

This, in turn, can be seen as an existential question and the system attempts to deny it by constructing a counter-example. That is, it attempts to find terms $t_1, \ldots, t_n$ such that the formula obtained from $A_1 \land \cdots \land A_m$ when replacing the variable X_i by t_i $(1 \leq i \leq n)$, is true in any model of the program, i.e. to construct a logical consequence of the program which is an instance of a conjunction of all subgoals in the goal.

By giving a definite goal the user selects the set of conclusions to be constructed. This set may be finite or infinite. The problem of how the machine constructs it will be discussed in Chapter 3. The section is concluded with some examples of queries and the answers obtained to the corresponding goals in a typical Prolog system.

Example 2.3 Referring to the family-example in Section 2.1 the user may ask the following queries (with the corresponding goal):

Query	Goal
"Is Ann a child of Tom?"	$\leftarrow child(ann, tom)$
"Who is a grandchild of Ann?"	$\leftarrow grandchild(X, ann)$
"Whose grandchild is Tom?"	$\leftarrow grandchild(tom, X)$
"Who is a grandchild of whom?"	$\leftarrow grandchild(X, Y)$

The following answers are obtained:

- Since there are no variables in the first goal the answer is simply "yes";
- Since the program contains no information about grandchildren of Ann the answer to the second goal is "no one" (although most Prolog implementations would answer simply "no";

[1]Of course, formally it is not correct to write $\leftarrow A_1, \ldots, A_m$ since "$\leftarrow$" should have a formula also on the left-hand side. The problem becomes even more evident when $m = 0$ because then the right-hand side disappears as well. However, formally the problem can be viewed as follows — a definite goal has the form $\forall(\neg(A_1 \land \cdots \land A_m))$ which is equivalent to $\forall(\Box \lor \neg(A_1 \land \cdots \land A_m \land \blacksquare))$. A nonempty goal can thus be viewed as the formula $\forall(\Box \leftarrow (A_1 \land \cdots \land A_m))$. The empty goal can be viewed as the formula $\Box \leftarrow \blacksquare$ which is equivalent to $\Box$.

- Since Tom is the grandchild of Mark the answer is $X = mark$ in reply to the third goal;
- The final goal yields three answers:

$$\begin{array}{ll} X = tom & Y = mark \\ X = alice & Y = mark \\ X = ann & Y = john \end{array}$$

It is also possible to ask more complicated queries, for example "Is there a person whose grandchildren are Tom and Alice?", expressed formally as:

$$\leftarrow grandchild(tom, X), grandchild(alice, X)$$

whose (expected) answer is $X = mark$. ▮

2.3 The Least Herbrand Model

Definite programs can only express positive knowledge — both facts and rules say which elements of a structure are in a relation, but they do not say when the relations do not hold. Therefore, using the language of definite programs, it is not possible to construct contradictory descriptions, i.e. unsatisfiable sets of formulas. In other words, every definite program has a model. This section discusses this matter in more detail. It shows also that every definite program has a well defined *least* model. Intuitively this model reflects all information expressed by the program and nothing more.

We first focus attention on models of a special kind, called *Herbrand models.* The idea is to abstract from the actual meanings of the functors (here, constants are treated as 0-ary functors) of the language. More precisely, attention is restricted to the interpretations where the domain is the set of variable-free terms and the meaning of every ground term is the term itself. After all, it is a common practice in databases — the constants *tom* and *ann* may represent persons but the database describes relations between the persons by handling relations between the terms (symbols) no matter whom they represent.

The formal definition of such domains follows and is illustrated by two simple examples.

Definition 2.4 (Herbrand universe, Herbrand base) Let $\mathcal{A}$ be an alphabet containing at least one constant symbol. The set $U_{\mathcal{A}}$ of all ground terms constructed from functors and constants in $\mathcal{A}$ is called the *Herbrand universe* of $\mathcal{A}$. The set $B_{\mathcal{A}}$ of all ground, atomic formulas over $\mathcal{A}$ is called the *Herbrand base* of $\mathcal{A}$. ▮

The Herbrand universe and Herbrand base are often defined for a given *program.* In this case it is assumed that the alphabet of the program consists of exactly those symbols which appear in the program. It is also assumed that the program contains at least one constant (since otherwise, the domain would be empty).

Example 2.5 Consider the following definite program P:

$$\begin{array}{l} odd(s(0)). \\ odd(s(s(X))) \leftarrow odd(X). \end{array}$$

The program contains one constant (0) and one unary functor (s). Consequently the Herbrand universe looks as follows:

$$U_P = \{0, s(0), s(s(0)), s(s(s(0))), \ldots\}$$

Since the program contains only one (unary) predicate symbol (*odd*) it has the following Herbrand base:

$$B_P = \{odd(0), odd(s(0)), odd(s(s(0))), \ldots\}$$

∎

Example 2.6 Consider the following definite program P:

$$\begin{array}{l} owns(owner(corvette), corvette). \\ happy(X) \leftarrow owns(X, corvette). \end{array}$$

In this case the Herbrand universe U_P consists of the set:

$$\{corvette, owner(corvette), owner(owner(corvette)), \ldots\}$$

and the Herbrand base B_P of the set:

$$\{owns(s,t) \mid s,t \in U_P\} \cup \{happy(s) \mid s \in U_P\}$$

∎

Definition 2.7 (Herbrand interpretations) A Herbrand interpretation of P is an interpretation $\Im$ such that:

- the domain of $\Im$ is U_P;
- for every constant c, $c_\Im$ is defined to be c itself;
- for every n-ary functor f the function $f_\Im$ is defined as follows

  $$f_\Im(x_1, \ldots, x_n) := f(x_1, \ldots, x_n)$$

  That is, the function $f_\Im$ applied to n ground terms composes them into the ground term with the principal functor f;
- for every n-ary predicate symbol p the relation $p_\Im$ is a subset of U_P^n (the set of all n-tuples of ground terms).

∎

Thus Herbrand interpretations have predefined meanings of functors and constants and in order to specify a Herbrand interpretation it suffices to list the relations associated with the predicate symbol. Hence, for an n-ary predicate symbol p and a Herbrand interpretation $\Im$ the meaning $p_\Im$ of p consists of the following set of n-tuples: $\{\langle t_1, \ldots, t_n\rangle \in U_P^n \mid \Im \models p(t_1, \ldots, t_n)\}$.

Example 2.8 One possible interpretation of the program P in Example 2.5 is $odd_{\Im} = \{\langle s(0)\rangle, \langle s(s(s(0)))\rangle\}$. A Herbrand interpretation can be specified by giving a family of such relations (one for every predicate symbol). ∎

Since the domain of a Herbrand interpretation is the Herbrand universe the relations are sets of tuples of ground terms. One can define all of them at once by specifying a set of *labelled* tuples, where the labels are predicate symbols. In other words: A Herbrand interpretation $\Im$ can be seen as a subset of the Herbrand base (or a possibly infinite relational database), namely $\{A \in B_P \mid \Im \models A\}$.

Example 2.9 Consider some alternative Herbrand interpretations for P of Example 2.5.

$$\begin{array}{rcl}
\Im_1 & := & \varnothing \\
\Im_2 & := & \{odd(s(0))\} \\
\Im_3 & := & \{odd(s(0)), odd(s(s(0)))\} \\
\Im_4 & := & \{odd(s^n(0)) \mid n \in \{1,3,5,7,\ldots\}\} \\
 & = & \{odd(s(0)), odd(s(s(s(0)))), \ldots\} \\
\Im_5 & := & B_P
\end{array}$$

∎

Definition 2.10 (Herbrand model) A Herbrand model of a set of (closed) formulas is a Herbrand interpretation which is a model of every formula in the set. ∎

It turns out that Herbrand interpretations and Herbrand models have two attractive properties. The first is pragmatic: In order to determine if a Herbrand interpretation $\Im$ is a model of a universally quantified formula $\forall F$ it suffices to check if all ground instances of F are true in $\Im$. For instance, to check if $A_0 \leftarrow A_1, \ldots, A_n$ is true in $\Im$ it suffices to show that if $(A_0 \leftarrow A_1, \ldots, A_n)\theta$ is a ground instance of $A_0 \leftarrow A_1, \ldots, A_n$ and $A_1\theta, \ldots, A_n\theta \in \Im$ then $A_0\theta \in \Im$.

Example 2.11 Clearly $\Im_1$ cannot be a model of P in Example 2.5 as it is not a Herbrand model of $odd(s(0))$. However, $\Im_2, \Im_3, \Im_4, \Im_5$ are all models of $odd(s(0))$ since $odd(s(0)) \in \Im_i$, $(2 \leq i \leq 5)$.

Now, $\Im_2$ is not a model of $odd(s(s(X))) \leftarrow odd(X)$ since there is a ground instance of the rule — namely $odd(s(s(s(0)))) \leftarrow odd(s(0))$ — such that all premises are true: $odd(s(0)) \in \Im_2$, but the conclusion is false: $odd(s(s(s(0)))) \notin \Im_2$. By a similar reasoning it follows that $\Im_3$ is not a model of the rule.

However, $\Im_4$ is a model also of the rule; let $odd(s(s(t))) \leftarrow odd(t)$ be any ground instance of the rule where $t \in U_P$. Clearly, $odd(s(s(t))) \leftarrow odd(t)$ is true if $odd(t) \notin \Im_4$ (check with Definition 1.6). Furthermore, if $odd(t) \in \Im_4$ then it must also hold that $odd(s(s(t))) \in \Im_4$ (cf. the the definition of $\Im_4$ above) and hence $odd(s(s(t))) \leftarrow odd(t)$ is true in $\Im_4$. Similar reasoning proves that $\Im_5$ is also a model of the program. ∎

The second reason for focusing on Herbrand interpretations is more theoretical. For the restricted language of definite programs, it turns out that in order to determine whether an atomic formula A is a logical consequence of a definite program P it suffices to check that every Herbrand model of P is also a Herbrand model of A.

Theorem 2.12 Let P be a definite program and G a definite goal. If $\Im'$ is a model of $P \cup \{G\}$ then $\Im := \{A \in B_P \mid \Im' \models A\}$ is a Herbrand model of $P \cup \{G\}$. ▮

Proof: Clearly, $\Im$ is a Herbrand interpretation. Now assume that $\Im'$ is a model and that $\Im$ is not a model of $P \cup \{G\}$. In other words, there exists a ground instance of a clause or a goal in $P \cup \{G\}$:

$$A_0 \leftarrow A_1, \ldots, A_m \quad (m \geq 0)$$

which is not true in $\Im$ ($A_0 = \Box$ in case of a goal).

Since this clause is false in $\Im$ then $A_1, \ldots, A_m$ are all true and A_0 is false in $\Im$. Hence, by the definition of $\Im$ we conclude that $A_1, \ldots, A_m$ are true and A_0 is false in $\Im'$. This contradicts the assumption that $\Im'$ is a model. Hence $\Im$ is a model of $P \cup \{G\}$. ▮

Notice that the form of P in Theorem 2.12 is restricted to definite programs. In the general case, nonexistence of a Herbrand model of a set of formulas P does not mean that P is unsatisfiable. That is, there are sets of formulas P which do not have a Herbrand model but which have other models.[2]

Example 2.13 Consider the formulas $\{\neg p(a), \exists X p(X)\}$ where $U_P := \{a\}$ and $B_P := \{p(a)\}$. Clearly, there are only two Herbrand interpretations — the empty set and B_P itself. The former is not a model of the second formula. The latter is a model of the second formula but not of the first.

However, it is not very hard to find a model of the formulas — let the domain be the natural numbers, assign 0 to the constant a and the relation $\{\langle 1\rangle, \langle 3\rangle, \langle 5\rangle, \ldots\}$ to the predicate symbol p (i.e. let p denote the "odd"-relation). Clearly this is a model since "0 is not odd" and "there exists a natural number which is odd, e.g. 1". ▮

Notice that the Herbrand base of a definite program P always *is* a Herbrand model of the program. To check that this is so, simply take an arbitrary ground instance of any clause $A_0 \leftarrow A_1, \ldots, A_m$ in P. Clearly, all $A_0, \ldots, A_m$ are in the Herbrand base. Hence the formula is true. However, this model is rather uninteresting — every n-ary predicate of the program is interpreted as the full n-ary relation over the domain of ground terms. More important is of course the question — what are the *interesting* models of the program? Intuitively there is no reason to expect that the model includes more ground atoms than those which follow from the program. By the analogy to databases — if John is not in the telephone directory he probably has no telephone. However, the directory gives only positive facts and if John has a telephone it is not a contradiction to what is said in the directory.

The rest of this section is organized as follows. First it is shown that there exists a *unique* minimal model called the *least Herbrand model* of a definite program. Then it is shown that this model really contains all positive information present in the program.

The Herbrand models of a definite program are subsets of its Herbrand base. Thus the set-inclusion is a natural ordering of such models. In order to show the existence of least models with respect to set-inclusion it suffices to show that the intersection of all Herbrand models is also a (Herbrand) model.

[2]More generally the result of Theorem 2.12 would hold for any set of *clauses*.

Theorem 2.14 (Model intersection property) Let M be a non-empty family of Herbrand models of a definite program P. Then the intersection $\Im := \bigcap M$ is a Herbrand model of P. ▮

Proof: Assume that $\Im$ is not a model of P. Then there exists a ground instance of a clause of P:

$$A_0 \leftarrow A_1, \ldots, A_m \quad (m \geq 0)$$

which is not true in $\Im$. This implies that $\Im$ contains $A_1, \ldots, A_m$ but not A_0. Then $A_1, \ldots, A_m$ are elements of every interpretation of the family M. Moreover there must be at least one model $\Im_i \in M$ such that $A_0 \notin \Im_i$. Thus $A_0 \leftarrow A_1, \ldots, A_m$ is not true in this $\Im_i$. Hence $\Im_i$ is not a model of the program, which contradicts the assumption. This concludes the proof that the intersection of any set of Herbrand models of a program is also a Herbrand model. ▮

Thus by taking the intersection of all Herbrand models (it is known that every definite program P has at least one Herbrand model — namely B_P) the least Herbrand model of the definite program is obtained.

Example 2.15 Let P be the definite program $\{male(adam), female(eve)\}$ with obvious intended interpretation. P has the following four Herbrand models:

$$\{male(adam), female(eve)\}$$
$$\{male(adam), male(eve), female(eve)\}$$
$$\{male(adam), female(eve), female(adam)\}$$
$$\{male(adam), male(eve), female(eve), female(adam)\}$$

It is not very hard to see that any intersection of these yields a Herbrand model. However, all but the first model contain atoms incompatible with the intended one. Notice also that the intersection of all four models yields a model which corresponds to the intended model. ▮

This example indicates a connection between the least Herbrand model and the intended model of a definite program. The intended model is an abstraction of the world to be described by the program. The world may be richer than the least Herbrand model. For instance, there may be more female individuals than just Eve. However, the information not included explicitly (via facts) or implicitly (via rules) in the program cannot be obtained as an answer to a goal. The answers correspond to *logical consequences* of the program. Ideally, a ground atomic formula $p(t_1, \ldots, t_n)$ is a logical consequence of the program iff, in the intended interpretation $\Im$, t_i denotes the individual x_i and $\langle x_1, \ldots, x_n \rangle \in p_\Im$. The set of all such ground atoms can be seen as a "coded" version of the intended model. The following theorem relates this set to the least Herbrand model.

Theorem 2.16 The least Herbrand model M_P of a definite program P is the set of all ground atomic logical consequences of the program. That is, $M_P = \{A \in B_P \mid P \models A\}$. ▮

Proof: Show first $M_P \supseteq \{A \in B_P \mid P \models A\}$: It is easy to see that every ground atom A which is a logical consequence of P is an element of M_P. Indeed, by the definition of logical consequence A must be true in M_P. On the other hand, the definition of Herbrand interpretation states that A is true in M_P iff A is an element of M_P.

Then show that $M_P \subseteq \{A \in B_P \mid P \models A\}$: Assume that A is in M_P. Hence it is true in every Herbrand model of P. Assume that it is not true in some non-Herbrand model $\Im'$ of P. But we know (see Theorem 2.12) that the set $\Im$ of all ground atomic formulas which are true in $\Im'$ is a Herbrand model of P. Hence A cannot be an element of $\Im$. This contradicts the assumption that there exists a model of P where A is false. Hence A is true in every model of P, that is $P \models A$, which concludes the proof. ∎

The model intersection property expressed by Theorem 2.14 does not hold for arbitrary formulas as illustrated by the following example.

Example 2.17 Consider the formula $p(a) \vee q(b)$. Clearly, both $\{p(a)\}$ and $\{q(b)\}$ are Herbrand models of the formula. However, the intersection $\{p(a)\} \cap \{q(b)\} = \varnothing$ is not a model. The two models are examples of *minimal* models — that is, one cannot remove any element from the model and still have a model. However, there is no *least* model — that is, a unique minimal model. ∎

2.4 Construction of Least Herbrand Models

The question arises how the least Herbrand model can be constructed, or approximated by successive enumeration of its elements. The answer to this question is given by a *fixed point* approach to the semantics of definite programs. (A fixpoint of a function $f : \mathcal{D} \to \mathcal{D}$ is an element $x \in \mathcal{D}$ such that $f(x) = x$.) This section gives only a sketch of the construction. The discussion of the relevant theory is outside of the scope of this book. However, the intuition behind the construction is the following:

A definite program consists of facts and rules. Clearly, all ground instances of the facts must be included in every Herbrand model. If a Herbrand interpretation $\Im$ does not include a ground instance of a fact A of the program then A is not true in $\Im$ and $\Im$ is not a model.

Next, consider a rule $A_0 \leftarrow A_1, \dots, A_m$ where $(m > 0)$. This rule states that whenever $A_1, \dots, A_m$ are true then so is A_0. In other words, take any ground instance $(A_0 \leftarrow A_1, \dots, A_m)\theta$ of the rule. If $\Im$ includes $A_1\theta, \dots, A_m\theta$ it must also include $A_0\theta$ in order to be a model.

Consider the set $\Im_1$ of all ground instances of facts in the program. It is now possible to use every instance of each rule to augment $\Im_1$ with new elements which necessarily must belong to every model. In that way a new set $\Im_2$ is obtained which can be used again to generate more elements which must belong to the model. This process is repeated as long as new elements are generated. The new elements added to $\Im_{i+1}$ are those which *must follow immediately* from $\Im_i$.

The construction outlined above can be formally defined as an iteration of a transformation T_P on Herbrand interpretations of the program P. The operation is called the *immediate consequence operator* and is defined as follows:

Definition 2.18 (Immediate consequence operator) Let *ground*(P) be the set of all ground instances of clauses in P. T_P is a function on Herbrand interpretations of P defined as follows:

$$T_P(I) := \{A_0 \mid A_0 \leftarrow A_1, \ldots, A_m \in \mathit{ground}(P) \wedge \{A_1, \ldots, A_m\} \subseteq I\}$$

∎

For definite programs it can be shown that there exists a least interpretation $\Im$ such that $T_P(\Im) = \Im$ and that $\Im$ is identical with the least Herbrand model M_P. Moreover, M_P is the limit of the increasing, possibly infinite sequence of iterations:

$$\varnothing, \quad T_P(\varnothing), \quad T_P(T_P(\varnothing)), \quad T_P(T_P(T_P(\varnothing))), \quad \ldots$$

There is a standard notation used to denote elements of the sequence of interpretations constructed for P. Namely:

$$\begin{array}{rcl} T_P \uparrow 0 & := & \varnothing \\ T_P \uparrow (i+1) & := & T_P(T_P \uparrow i) \\ T_P \uparrow \omega & := & \bigcup_{i=0}^{\infty} T_P \uparrow i \end{array}$$

The following example illustrates the construction:

Example 2.19 Consider again the program of Example 2.5.

$$\begin{array}{rcl} T_P \uparrow 0 & = & \varnothing \\ T_P \uparrow 1 & = & \{\mathit{odd}(s(0))\} \\ T_P \uparrow 2 & = & \{\mathit{odd}(s(s(s(0)))), \mathit{odd}(s(0))\} \\ & \vdots & \\ T_P \uparrow \omega & = & \{\mathit{odd}(s^n(0)) \mid n \in \{1, 3, 5, \ldots\}\} \end{array}$$

∎

As already mentioned above it has been established that the set constructed in this way is identical to the least Herbrand model.

Theorem 2.20 Let P be a definite program and M_P its least Herbrand model. Then:

- M_P is the least Herbrand interpretation such that $T_P(M_P) = M_P$ (i.e. it is the least fixpoint of T_P).
- $M_P = T_P \uparrow \omega$.

∎

For additional details and proofs see for example Apt (1990), Lloyd (1987) or van Emden and Kowalski (1976).

Exercises

2.1 Rewrite the following formulas in the form $A_0 \leftarrow A_1, \ldots, A_m$:

$$\begin{gathered}\forall X(p(X) \lor \neg q(X))\\ \forall X(p(X) \lor \neg\exists\, Y\,(q(X,Y) \land r(X)))\\ \forall X(\neg p(X) \lor (q(X) \supset r(X)))\\ \forall X(r(X) \supset (q(X) \supset p(X)))\end{gathered}$$

2.2 Formalize the following scenario as a definite program:

> Basil owns Fawlty Towers. Basil and Sybil are married. Polly and Manuel are employees at Fawlty Towers. Smith and Jones are guests at Fawlty Towers. All hotel-owners and their spouses serve all guests at the hotel. All employees at a hotel serve all guests at the hotel. All employees dislike the owner of the workplace. Basil dislikes Manuel.

Then ask the queries "Who serves who?" and "Who dislikes who?".

2.3 Give the Herbrand universe and Herbrand base of the following definite program:

$$\begin{aligned}&p(f(X)) \leftarrow q(X, g(X)).\\ &q(a, g(b)).\\ &q(b, g(b)).\end{aligned}$$

2.4 Give the Herbrand universe and Herbrand base of the following definite program:

$$\begin{aligned}&p(s(X), Y, s(Z)) \leftarrow p(X, Y, Z).\\ &p(0, X, X).\end{aligned}$$

2.5 Consider the Herbrand universe consisting of the constants a, b, c and d. Let $\Im$ be the Herbrand interpretation:

$$\{p(a), p(b), q(a), q(b), q(c), q(d)\}$$

Which of the following formulas are true in $\Im$?

$$\begin{array}{ll}(1) & \forall X p(X)\\ (2) & \forall X q(X)\\ (3) & \exists X(q(X) \land p(X))\\ (4) & \forall X(q(X) \supset p(X))\\ (5) & \forall X(p(X) \supset q(X))\end{array}$$

2.6 Give the least Herbrand model of the program in exercise 2.3.

2.7 Give the least Herbrand model of the program in exercise 2.4. *Hint*: the model is infinite, but a certain pattern can be spotted when using the T_P-operator.

2.8 Consider the following program:

$$\begin{array}{l} p(0). \\ p(s(X)) \leftarrow p(X). \end{array}$$

Show that $p(s^n(0)) \in T_P \uparrow m$ iff $n < m$.

2.9 Let P be a definite program and $\Im$ a Herbrand interpretation. Show that $\Im$ is a model of P iff $T_P(\Im) \subseteq \Im$.