

# Chapter 1

## Preliminaries

### 1.1 Logic Formulas

When describing some state of affairs in the real world we often use *declarative*<sup>1</sup> sentences like:

- (i) “Every mother loves her children”
- (ii) “Mary is a mother and Tom is Mary’s child”

By applying some general rules of reasoning such descriptions can be used to draw new conclusions. For example, knowing (i) and (ii) it is possible to conclude that:

- (iii) “Mary loves Tom”

A closer inspection reveals that (i) and (ii) describe some *universe* of persons and some *relations* between these individuals — like “... is a mother”, “... is a child of ...” or the relation “... loves ...” — which may or may not hold between the persons.<sup>2</sup> This example reflects the principal idea of *logic programming* — to describe possibly infinite relations on objects and to apply the programming system in order to draw conclusions like (iii).

For a computer to deal with sentences like (i)–(iii) the *syntax* of the sentences must be precisely defined. What is even more important, the rules of reasoning — like the one

---

<sup>1</sup>The notion of declarative sentence has its roots in linguistics. A declarative sentence is a complete expression of natural language which is either true or false, as opposed to e.g. imperative or interrogative sentences (commands and questions). Only declarative sentences can be expressed in predicate logic.

<sup>2</sup>Some people would probably argue that “being a mother” is not a relation but rather a property. However, for the sake of uniformity properties will be called relations and so will statements which relate more than two objects (like “... is the sum of ... and ...”).

which permits inferring (iii) from (i) and (ii) — must be carefully formalized. Such problems have been studied in the field of mathematical logic. This chapter surveys basic logical concepts that are used later on in the book to relate logic programming and logic. (For basic set theoretic notions see Appendix B.)

The first concept considered is that of *logic formulas* which provide a formalized syntax for writing sentences like (i)–(iii). Such sentences refer to *individuals* in some *world* and to *relations* between those individuals. Therefore the starting point is an assumption about the alphabet of the language. It must include:

- symbols for denoting individuals (e.g. the symbol *tom* may be used to denote the person Tom of our example). Such symbols will be called *constants*;
- symbols for denoting relations (*loves*, *mother*, *child\_of*). Such symbols are called *predicate symbols*.

Every predicate symbol has an associated natural number, called its arity. The relation named by an  $n$ -ary predicate symbol is a set of  $n$ -tuples of individuals; in the example above the predicate symbol *loves* denotes a set of pairs of persons, including the pair Mary and Tom, denoted by the constants *mary* and *tom*.

With the alphabet of constants, predicate symbols and some auxiliary characters, sentences of natural language like “Mary loves Tom” can be formalized as formulas like *loves(mary, tom)*.

The formal language should also provide the possibility of expressing sentences like (i) which refers to *all* elements of the described “world”. This sentence says that “for all individuals X and Y, if X is a mother and Y is a child of X then X loves Y”. For this purpose, the language of logic introduces the symbol of *universal quantifier* “ $\forall$ ” (to be read “for every” or “for all”) and the alphabet of *variables*. A variable is a symbol that refers to an unspecified individual, like X and Y above. Now the sentences (i)–(iii) can be formalized accordingly:

$$\forall X (\forall Y ((mother(X) \wedge child\_of(Y, X)) \supset loves(X, Y))) \quad (1)$$

$$mother(mary) \wedge child\_of(tom, mary) \quad (2)$$

$$loves(mary, tom) \quad (3)$$

The symbols “ $\wedge$ ” and “ $\supset$ ” are examples of *logical connectives* which are used to combine logic formulas — “ $\wedge$ ” reads “and” and is called *conjunction* whereas “ $\supset$ ” is called *implication* and corresponds to the “if-then” construction above. Parentheses are used to disambiguate the language.

Another connective which will be used frequently is that for expressing negation. It is denoted by “ $\neg$ ” (with reading “not”). For example the sentence “Tom does not love Mary” can be formalized as the formula:

$$\neg loves(tom, mary)$$

In what follows the symbol “ $\exists$ ” is also sometimes used. It is called the *existential quantifier* and reads “there exists”. The existential quantifier makes it possible to express the fact that, in the world under consideration, there exists at least one individual

which is in a certain relation with some other individuals. For example the sentence “Mary has a child” can be formalized as the formula:

$$\exists X \textit{ child\_of}(X, \textit{ mary})$$

On occasion the logical connectives “ $\vee$ ” and “ $\leftrightarrow$ ” are used. They formalize the connectives “or” and “if and only if” (“iff”).

So far individuals have been represented only by constants. However it is often the case that in the world under consideration, some “individuals” are “composed objects”. For instance, in some world it may be necessary to discuss relations between families as well as relations between persons. In this case it would be desirable to refer to a given family by a construction composed of the constants identifying the members of the family (actually what is needed is a *function* that constructs a family from its members). The language of logic offers means of solving this problem. It is assumed that its alphabet contains symbols called *functors* that represent functions over object domains. Every functor has assigned a natural number called its arity, which determines the number of arguments of the function. The constants can be seen as 0-ary functors. Assume now that there is a ternary<sup>3</sup> functor *family*, a binary functor *child* and a constant *none*. The family consisting of the parents Bill and Mary and children Tom and Alice can now be represented by the construction:

$$\textit{ family}(\textit{ bill}, \textit{ mary}, \textit{ child}(\textit{ tom}, \textit{ child}(\textit{ alice}, \textit{ none})))$$

Such a construction is called a *compound term*.

The above informal discussion based on examples of simple declarative sentences gives motivation for introducing basic constructs of the language of symbolic logic. The kind of logic used here is called *predicate logic*. Next a formal definition of this language is given. For the moment we specify only the form of allowed sentences, while the meaning of the language will be discussed separately. Thus the definition covers only the *syntax* of the language separated from its *semantics*.

From the syntactic point of view logic formulas are finite sequences of symbols such as variables, functors and predicate symbols. There are infinitely many of them and therefore the symbols are usually represented by finite strings of primitive characters. The representation employed in this book usually conforms to that specified in the ISO standard of the programming language Prolog (1995). Thus, the *alphabet* of the language of predicate logic consists of the following classes of symbols:

- *variables* which will be written as alphanumeric identifiers beginning with capital letters (sometimes subscripted). Examples of variables are  $X, X_s, Y, X_7, \dots$ ;
- *constants* which are numerals or alphanumeric identifiers beginning with lower-case letters. Examples of constants are  $x, \textit{ alf}, \textit{ none}, 17, \dots$ ;
- *functors* which are alphanumeric identifiers beginning with lower-case letters and with an associated arity  $> 0$ . To emphasize the arity  $n$  of a functor  $f$  it is sometimes written in the form  $f/n$ ;

---

<sup>3</sup>Usually the terms *nullary*, *unary*, *binary* and *ternary* are used instead of 0-ary, 1-ary, 2-ary and 3-ary.

- *predicate symbols* which are usually alphanumeric identifiers starting with lower-case letters and with an associated arity  $\geq 0$ . The notation  $p/n$  is used also for predicate symbols;
- *logical connectives* which are  $\wedge$  (conjunction),  $\neg$  (negation),  $\leftrightarrow$  (logical equivalence),  $\supset$  (implication) and  $\vee$  (disjunction);
- *quantifiers* —  $\forall$  (universal) and  $\exists$  (existential);
- *auxiliary* symbols like parentheses and commas.

No syntactic distinction will be imposed between constants, functors and predicate symbols. However, as a notational convention we use  $a, b, c, \dots$  (with or without adornments) to denote constants and  $X, Y, Z, \dots$  to denote variables. Functors are denoted  $f, g, h, \dots$  and  $p, q, r, \dots$  are used to denote predicate symbols. Constants are sometimes viewed as nullary functors. Notice also that the sets of functors and predicate symbols may contain identical identifiers with different arities.

Sentences of natural language consist of words where objects of the described world are represented by nouns. In the formalized language of predicate logic objects will be represented by strings called *terms* whose syntax is defined as follows:

**Definition 1.1 (Terms)** The set  $\mathcal{T}$  of *terms* over a given alphabet  $\mathcal{A}$  is the smallest set such that:

- any constant in  $\mathcal{A}$  is in  $\mathcal{T}$ ;
- any variable in  $\mathcal{A}$  is in  $\mathcal{T}$ ;
- if  $f/n$  is a functor in  $\mathcal{A}$  and  $t_1, \dots, t_n \in \mathcal{T}$  then  $f(t_1, \dots, t_n) \in \mathcal{T}$ .

In this book terms are typically denoted by  $s$  and  $t$ .

In natural language only certain combinations of words are meaningful sentences. The counterpart of sentences in predicate logic are special constructs built from terms. These are called *formulas* or well-formed formulas (*wff*) and their syntax is defined as follows:

**Definition 1.2 (Formulas)** Let  $\mathcal{T}$  be the set of terms over the alphabet  $\mathcal{A}$ . The set  $\mathcal{F}$  of *wff* (with respect to  $\mathcal{A}$ ) is the smallest set such that:

- if  $p/n$  is a predicate symbol in  $\mathcal{A}$  and  $t_1, \dots, t_n \in \mathcal{T}$  then  $p(t_1, \dots, t_n) \in \mathcal{F}$ ;
- if  $F$  and  $G \in \mathcal{F}$  then so are  $(\neg F)$ ,  $(F \wedge G)$ ,  $(F \vee G)$ ,  $(F \supset G)$  and  $(F \leftrightarrow G)$ ;
- if  $F \in \mathcal{F}$  and  $X$  is a variable in  $\mathcal{A}$  then  $(\forall X F)$  and  $(\exists X F) \in \mathcal{F}$ .

Formulas of the form  $p(t_1, \dots, t_n)$  are called *atomic formulas* (or simply *atoms*).

In order to adopt a syntax similar to that of Prolog, formulas in the form  $(F \supset G)$  are instead written in the form  $(G \leftarrow F)$ . To simplify the notation parentheses will be removed whenever possible. To avoid ambiguity it will be assumed that the connectives

have a binding-order where  $\neg$ ,  $\forall$  and  $\exists$  bind stronger than  $\vee$ , which in turn binds stronger than  $\wedge$  followed by  $\supset$  (i.e.  $\leftarrow$ ) and finally  $\leftrightarrow$ . Thus  $(a \leftarrow ((\neg b) \wedge c))$  will be simplified into  $a \leftarrow \neg b \wedge c$ . Sometimes binary functors and predicate symbols are written in infix notation (e.g.  $2 \leq 3$ ).

Let  $F$  be a formula. An occurrence of the variable  $X$  in  $F$  is said to be *bound* either if the occurrence follows directly after a quantifier or if it appears inside the subformula which follows directly after “ $\forall X$ ” or “ $\exists X$ ”. Otherwise the occurrence is said to be *free*. A formula with no free occurrences of variables is said to be *closed*. A formula/term which contains no variables is called *ground*.

Let  $X_1, \dots, X_n$  be all variables that occur free in a formula  $F$ . The closed formula of the form  $\forall X_1(\dots(\forall X_n F)\dots)$  is called the *universal closure* of  $F$  and is denoted  $\forall F$ . Similarly,  $\exists F$  is called the *existential closure* of  $F$  and denotes the formula  $F$  closed under existential quantification.

## 1.2 Semantics of Formulas

The previous section introduced the language of formulas as a formalization of a class of declarative statements of natural language. Such sentences refer to some “world” and may be true or false in this world. The meaning of a logic formula is also defined relative to an “abstract world” called an (algebraic) *structure* and is also either true or false. In other words, to define the meaning of formulas, a formal connection between the language and a structure must be established. This section discusses the notions underlying this idea.

As stated above declarative statements refer to individuals, and concern relations and functions on individuals. Thus the mathematical abstraction of the “world”, called a structure, is a nonempty set of individuals (called the *domain*) with a number of relations and functions defined on this domain. For example the structure referred to by the sentences (i)–(iii) may be an abstraction of the world shown in Figure 1.1. Its domain consists of three individuals — Mary, John and Tom. Moreover, three relations will be considered on this set: a unary relation, “... is a mother”, and two binary relations, “... is a child of ...” and “... loves ...”. For the sake of simplicity it is assumed that there are no functions in the structure.

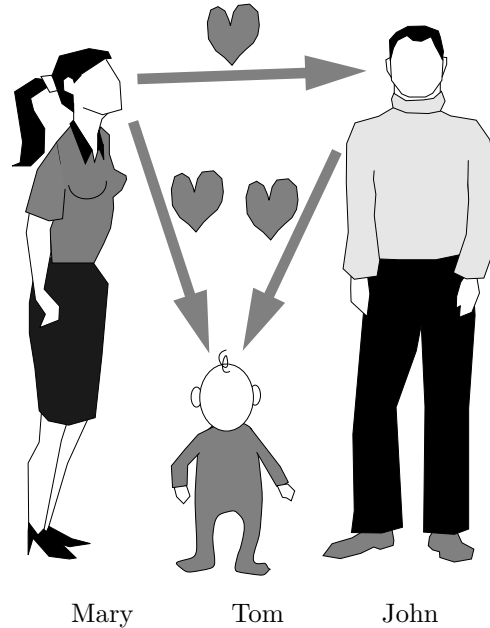
The building blocks of the language of formulas are constants, functors and predicate symbols. The link between the language and the structure is established as follows:

**Definition 1.3 (Interpretation)** An interpretation  $\mathfrak{I}$  of an alphabet  $\mathcal{A}$  is a non-empty domain  $\mathcal{D}$  (sometimes denoted  $|\mathfrak{I}|$ ) and a mapping that associates:

- each constant  $c \in \mathcal{A}$  with an element  $c_{\mathfrak{I}} \in \mathcal{D}$ ;
- each  $n$ -ary functor  $f \in \mathcal{A}$  with a function  $f_{\mathfrak{I}}: \mathcal{D}^n \rightarrow \mathcal{D}$ ;
- each  $n$ -ary predicate symbol  $p \in \mathcal{A}$  with a relation  $p_{\mathfrak{I}} \subseteq \underbrace{\mathcal{D} \times \dots \times \mathcal{D}}_n$ .

■

The interpretation of constants, functors and predicate symbols provides a basis for assigning truth values to formulas of the language. The meaning of a formula will be



**Figure 1.1: A family structure**

defined as a function on meanings of its components. First the meaning of terms will be defined since they are components of formulas. Since terms may contain variables the auxiliary notion of *valuation* is needed. A valuation  $\varphi$  is a mapping from variables of the alphabet to the domain of an interpretation. Thus, it is a function which assigns objects of an interpretation to variables of the language. By the notation  $\varphi[X \mapsto t]$  we denote the valuation which is identical to  $\varphi$  except that  $\varphi[X \mapsto t]$  maps  $X$  to  $t$ .

**Definition 1.4 (Semantics of terms)** Let  $\mathfrak{I}$  be an interpretation,  $\varphi$  a valuation and  $t$  a term. Then the *meaning*  $\varphi_{\mathfrak{I}}(t)$  of  $t$  is an element in  $|\mathfrak{I}|$  defined as follows:

- if  $t$  is a constant  $c$  then  $\varphi_{\mathfrak{I}}(t) := c_{\mathfrak{I}}$ ;
- if  $t$  is a variable  $X$  then  $\varphi_{\mathfrak{I}}(t) := \varphi(X)$ ;
- if  $t$  is of the form  $f(t_1, \dots, t_n)$ , then  $\varphi_{\mathfrak{I}}(t) := f_{\mathfrak{I}}(\varphi_{\mathfrak{I}}(t_1), \dots, \varphi_{\mathfrak{I}}(t_n))$ .

■

Notice that the meaning of a compound term is obtained by applying the function denoted by its main functor to the meanings of its principal subterms, which are obtained by recursive application of this definition.

**Example 1.5** Consider a language which includes the constant *zero*, the unary functor *s* and the binary functor *plus*. Assume that the domain of  $\mathfrak{I}$  is the set of the natural numbers ( $\mathbb{N}$ ) and that:

$$zero_{\mathfrak{I}} := 0$$

$$\begin{aligned} s_{\mathfrak{S}}(x) &:= 1 + x \\ plus_{\mathfrak{S}}(x, y) &:= x + y \end{aligned}$$

That is, *zero* denotes the *natural number* 0, *s* denotes the successor function and *plus* denotes the addition function. For the interpretation  $\mathfrak{S}$  and a valuation  $\varphi$  such that  $\varphi(X) := 0$  the meaning of the term  $plus(s(zero), X)$  is obtained as follows:

$$\begin{aligned} \varphi_{\mathfrak{S}}(plus(s(zero), X)) &= \varphi_{\mathfrak{S}}(s(zero)) + \varphi_{\mathfrak{S}}(X) \\ &= (1 + \varphi_{\mathfrak{S}}(zero)) + \varphi(X) \\ &= (1 + 0) + 0 \\ &= 1 \end{aligned}$$

■

The meaning of a formula is a truth value. The meaning depends on the components of the formula which are either (sub-) formulas or terms. As a consequence the meanings of formulas also rely on valuations. In the following definition the notation  $\mathfrak{S} \models_{\varphi} Q$  is used as a shorthand for the statement “ $Q$  is true with respect to  $\mathfrak{S}$  and  $\varphi$ ” and  $\mathfrak{S} \not\models_{\varphi} Q$  is to be read “ $Q$  is false w.r.t.  $\mathfrak{S}$  and  $\varphi$ ”.

**Definition 1.6 (Semantics of wff’s)** Let  $\mathfrak{S}$  be an interpretation,  $\varphi$  a valuation and  $Q$  a formula. The meaning of  $Q$  w.r.t.  $\mathfrak{S}$  and  $\varphi$  is defined as follows:

- $\mathfrak{S} \models_{\varphi} p(t_1, \dots, t_n)$  iff  $\langle \varphi_{\mathfrak{S}}(t_1), \dots, \varphi_{\mathfrak{S}}(t_n) \rangle \in p_{\mathfrak{S}}$ ;
- $\mathfrak{S} \models_{\varphi} (\neg F)$  iff  $\mathfrak{S} \not\models_{\varphi} F$ ;
- $\mathfrak{S} \models_{\varphi} (F \wedge G)$  iff  $\mathfrak{S} \models_{\varphi} F$  and  $\mathfrak{S} \models_{\varphi} G$ ;
- $\mathfrak{S} \models_{\varphi} (F \vee G)$  iff  $\mathfrak{S} \models_{\varphi} F$  or  $\mathfrak{S} \models_{\varphi} G$  (or both);
- $\mathfrak{S} \models_{\varphi} (F \supset G)$  iff  $\mathfrak{S} \models_{\varphi} G$  whenever  $\mathfrak{S} \models_{\varphi} F$ ;
- $\mathfrak{S} \models_{\varphi} (F \leftrightarrow G)$  iff  $\mathfrak{S} \models_{\varphi} (F \supset G)$  and  $\mathfrak{S} \models_{\varphi} (G \supset F)$ ;
- $\mathfrak{S} \models_{\varphi} (\forall X F)$  iff  $\mathfrak{S} \models_{\varphi[X \mapsto t]} F$  for every  $t \in |\mathfrak{S}|$ ;
- $\mathfrak{S} \models_{\varphi} (\exists X F)$  iff  $\mathfrak{S} \models_{\varphi[X \mapsto t]} F$  for some  $t \in |\mathfrak{S}|$ .

■

The semantics of formulas as defined above relies on the auxiliary concept of valuation that associates variables of the formula with elements of the domain of the interpretation. It is easy to see that the truth value of a closed formula depends only on the interpretation. It is therefore common practice in logic programming to consider all formulas as being implicitly universally quantified. That is, whenever there are free occurrences of variables in a formula its universal closure is considered instead. Since the valuation is of no importance for closed formulas it will be omitted when considering the meaning of such formulas.

**Example 1.7** Consider Example 1.5 again. Assume that the language contains also a unary predicate symbol  $p$  and that:

$$p_{\mathfrak{S}} := \{\langle 1 \rangle, \langle 3 \rangle, \langle 5 \rangle, \langle 7 \rangle, \dots\}$$

Then the meaning of the formula  $p(\text{zero}) \wedge p(s(\text{zero}))$  in the interpretation  $\mathfrak{S}$  is determined as follows:

$$\begin{aligned} \mathfrak{S} \models p(\text{zero}) \wedge p(s(\text{zero})) &\text{ iff } \mathfrak{S} \models p(\text{zero}) \text{ and } \mathfrak{S} \models p(s(\text{zero})) \\ &\text{ iff } \langle \varphi_{\mathfrak{S}}(\text{zero}) \rangle \in p_{\mathfrak{S}} \text{ and } \langle \varphi_{\mathfrak{S}}(s(\text{zero})) \rangle \in p_{\mathfrak{S}} \\ &\text{ iff } \langle \varphi_{\mathfrak{S}}(\text{zero}) \rangle \in p_{\mathfrak{S}} \text{ and } \langle 1 + \varphi_{\mathfrak{S}}(\text{zero}) \rangle \in p_{\mathfrak{S}} \\ &\text{ iff } \langle 0 \rangle \in p_{\mathfrak{S}} \text{ and } \langle 1 \rangle \in p_{\mathfrak{S}} \end{aligned}$$

Now  $\langle 1 \rangle \in p_{\mathfrak{S}}$  but  $\langle 0 \rangle \notin p_{\mathfrak{S}}$  so the whole formula is false in  $\mathfrak{S}$ . ■

**Example 1.8** Consider the interpretation  $\mathfrak{S}$  that assigns:

- the persons Tom, John and Mary of the structure in Figure 1.1 to the constants *tom*, *john* and *mary*;
- the relations “... is a mother”, “... is a child of ...” and “... loves ...” of the structure in Figure 1.1 to the predicate symbols *mother/1*, *child\_of/2* and *loves/2*.

Using the definition above it is easy to show that the meaning of the formula:

$$\forall X \exists Y \text{ loves}(X, Y)$$

is false in  $\mathfrak{S}$  (since Tom does not love anyone), while the meaning of formula:

$$\exists X \forall Y \neg \text{loves}(Y, X)$$

is true in  $\mathfrak{S}$  (since Mary is not loved by anyone). ■

### 1.3 Models and Logical Consequence

The motivation for introducing the language of formulas was to give a tool for describing “worlds” — that is, algebraic structures. Given a set of closed formulas  $P$  and an interpretation  $\mathfrak{S}$  it is natural to ask whether the formulas of  $P$  give a proper account of this world. This is the case if all formulas of  $P$  are true in  $\mathfrak{S}$ .

**Definition 1.9 (Model)** An interpretation  $\mathfrak{S}$  is said to be a *model* of  $P$  iff every formula of  $P$  is true in  $\mathfrak{S}$ . ■

Clearly  $P$  has infinitely many interpretations. However, it may happen that none of them is a model of  $P$ . A trivial example is any  $P$  that includes the formula  $(F \wedge \neg F)$  where  $F$  is an arbitrary (closed) formula. Such sets of formulas are called *unsatisfiable*. When using formulas for describing “worlds” it is necessary to make sure that every description produced is *satisfiable* (that is, has at least one model), and in particular that the world being described is a model of  $P$ .

Generally, a satisfiable set of formulas has (infinitely) many models. This means that the formulas which properly describe a particular “world” of interest at the same time describe many other worlds.

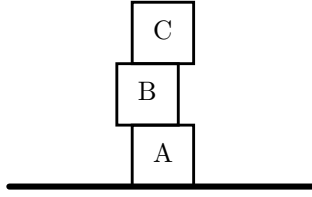


Figure 1.2: An alternative structure

**Example 1.10** Figure 1.2 shows another structure which can be used as a model of the formulas (1) and (2) of Section 1.1 which were originally used to describe the world of Figure 1.1. In order for the structure to be a model the constants *tom*, *john* and *mary* are interpreted as the boxes ‘A’, ‘B’ and ‘C’ respectively — the predicate symbols *loves*, *child\_of* and *mother* are interpreted as the relations “... is above ...”, “... is below ...” and “... is on top”. ■

Our intention is to use the description of the world of interest to obtain more information about this world. This new information is to be represented by new formulas not explicitly included in the original description. An example is the formula (3) of Section 1.1 which is obtained from (1) and (2). In other words, for a given set  $P$  of formulas other formulas (say  $F$ ) which are also true in the world described by  $P$  are searched for. Unfortunately,  $P$  itself has many models and does not uniquely identify the “intended model” which was described by  $P$ . Therefore it must be required that  $F$  is true in every model of  $P$  to guarantee that it is also true in the particular world of interest. This leads to the fundamental concept of *logical consequence*.

**Definition 1.11 (Logical consequence)** Let  $P$  be a set of closed formulas. A closed formula  $F$  is called a logical consequence of  $P$  (denoted  $P \models F$ ) iff  $F$  is true in every model of  $P$ . ■

**Example 1.12** To illustrate this notion by an example it is shown that (3) is a logical consequence of (1) and (2). Let  $\mathfrak{S}$  be an arbitrary interpretation. If  $\mathfrak{S}$  is a model of (1) and (2) then:

$$\mathfrak{S} \models \forall X(\forall Y((mother(X) \wedge child\_of(Y, X)) \supset loves(X, Y))) \quad (4)$$

$$\mathfrak{S} \models mother(mary) \wedge child\_of(tom, mary) \quad (5)$$

For (4) to be true it is necessary that:

$$\mathfrak{S} \models_{\varphi} mother(X) \wedge child\_of(Y, X) \supset loves(X, Y) \quad (6)$$

for any valuation  $\varphi$  — specifically for  $\varphi(X) = mary_{\mathfrak{S}}$  and  $\varphi(Y) = tom_{\mathfrak{S}}$ . However, since these individuals are denoted by the constants *mary* and *tom* it must also hold that:

$$\mathfrak{S} \models mother(mary) \wedge child\_of(tom, mary) \supset loves(mary, tom) \quad (7)$$

Finally, for this to hold it follows that *loves(mary, tom)* must be true in  $\mathfrak{S}$  (by Definition 1.6 and since (5) holds by assumption). Hence, any model of (1) and (2) is also a model of (3). ■

This example shows that it may be rather difficult to prove that a formula is a logical consequence of a set of formulas. The reason is that one has to use the semantics of the language of formulas and to deal with all models of the formulas.

One possible way to prove  $P \models F$  is to show that  $\neg F$  is false in every model of  $P$ , or put alternatively, that the set of formulas  $P \cup \{\neg F\}$  is unsatisfiable (has no model). The proof of the following proposition is left as an exercise.

**Proposition 1.13 (Unsatisfiability)** Let  $P$  be a set of closed formulas and  $F$  a closed formula. Then  $P \models F$  iff  $P \cup \{\neg F\}$  is unsatisfiable. ■

It is often straightforward to show that a formula  $F$  is not a logical consequence of the set  $P$  of formulas. For this, it suffices to give a model of  $P$  which is not a model of  $F$ .

**Example 1.14** Let  $P$  be the formulas:

$$\forall X(r(X) \supset (p(X) \vee q(X))) \quad (8)$$

$$r(a) \wedge r(b) \quad (9)$$

To prove that  $p(a)$  is not a logical consequence of  $P$  it suffices to consider an interpretation  $\mathfrak{S}$  where  $|\mathfrak{S}|$  is the set consisting of the two persons “Adam” and “Eve” and where:

$$a_{\mathfrak{S}} := \text{Adam}$$

$$b_{\mathfrak{S}} := \text{Eve}$$

$$p_{\mathfrak{S}} := \{\langle \text{Eve} \rangle\} \quad \% \text{ the property of being female}$$

$$q_{\mathfrak{S}} := \{\langle \text{Adam} \rangle\} \quad \% \text{ the property of being male}$$

$$r_{\mathfrak{S}} := \{\langle \text{Adam} \rangle, \langle \text{Eve} \rangle\} \quad \% \text{ the property of being a person}$$

Clearly, (8) is true in  $\mathfrak{S}$  since “any person is either female or male”. Similarly (9) is true since “both Adam and Eve are persons”. However,  $p(a)$  is false in  $\mathfrak{S}$  since Adam is not a female. ■

Another important concept based on the semantics of formulas is the notion of *logical equivalence*.

**Definition 1.15 (Logical equivalence)** Two formulas  $F$  and  $G$  are said to be logically equivalent (denoted  $F \equiv G$ ) iff  $F$  and  $G$  have the same truth value for all interpretations  $\mathfrak{S}$  and valuations  $\varphi$ . ■

Next a number of well-known facts concerning equivalences of formulas are given. Let  $F$  and  $G$  be arbitrary formulas and  $H(X)$  a formula with zero or more free occurrences of  $X$ . Then:

$$\begin{aligned} \neg\neg F &\equiv F \\ F \supset G &\equiv \neg F \vee G \\ F \supset G &\equiv \neg G \supset \neg F \\ F \leftrightarrow G &\equiv (F \supset G) \wedge (G \supset F) \\ \neg(F \vee G) &\equiv \neg F \wedge \neg G && \text{DeMorgan's law} \\ \neg(F \wedge G) &\equiv \neg F \vee \neg G && \text{DeMorgan's law} \\ \neg\forall X H(X) &\equiv \exists X \neg H(X) && \text{DeMorgan's law} \\ \neg\exists X H(X) &\equiv \forall X \neg H(X) && \text{DeMorgan's law} \end{aligned}$$

and if there are no free occurrences of  $X$  in  $F$  then:

$$\forall X(F \vee H(X)) \equiv F \vee \forall XH(X)$$

Proofs of these equivalences are left as an exercise to the reader.

## 1.4 Logical Inference

In Section 1.1 the sentence (iii) was obtained by reasoning about the sentences (i) and (ii). The language was then formalized and the sentences were expressed as the logical formulas (1), (2) and (3). With this formalization, reasoning can be seen as a process of manipulation of formulas, which from a given set of formulas, like (1) and (2), called the *premises*, produces a new formula called the *conclusion*, for instance (3). One of the objectives of the symbolic logic is to formalize “reasoning principles” as formal re-write rules that can be used to generate new formulas from given ones. These rules are called *inference rules*. It is required that the inference rules correspond to correct ways of reasoning — whenever the premises are true in any world under consideration, any conclusion obtained by application of an inference rule should also be true in this world. In other words it is required that the inference rules produce only logical consequences of the premises to which they can be applied. An inference rule satisfying this requirement is said to be *sound*.

Among well-known inference rules of predicate logic the following are frequently used:

- *Modus ponens* or elimination rule for implication: This rule says that whenever formulas of the form  $F$  and  $(F \supset G)$  belong to or are concluded from a set of premises,  $G$  can be inferred. This rule is often presented as follows:

$$\frac{F \quad F \supset G}{G} \quad (\supset E)$$

- Elimination rule for universal quantifier: This rule says that whenever a formula of the form  $(\forall X F)$  belongs to or is concluded from the premises a new formula can be concluded by replacing all free occurrences of  $X$  in  $F$  by some term  $t$  which is *free for X* (that is, all variables in  $t$  remain free when  $X$  is replaced by  $t$ : for details see e.g. van Dalen (1983) page 68). This rule is often presented as follows:

$$\frac{\forall X F(X)}{F(t)} \quad (\forall E)$$

- Introduction rule for conjunction: This rule states that if formulas  $F$  and  $G$  belong to or are concluded from the premises then the conclusion  $F \wedge G$  can be inferred. This is often stated as follows:

$$\frac{F \quad G}{F \wedge G} \quad (\wedge I)$$

Soundness of these rules can be proved directly from the definition of the semantics of the language of formulas.

Their use can be illustrated by considering the example above. The premises are:

$$\forall X (\forall Y (mother(X) \wedge child\_of(Y, X) \supset loves(X, Y))) \quad (10)$$

$$mother(mary) \wedge child\_of(tom, mary) \quad (11)$$

Elimination of the universal quantifier in (10) yields:

$$\forall Y (mother(mary) \wedge child\_of(Y, mary) \supset loves(mary, Y)) \quad (12)$$

Elimination of the universal quantifier in (12) yields:

$$mother(mary) \wedge child\_of(tom, mary) \supset loves(mary, tom) \quad (13)$$

Finally *modus ponens* applied to (11) and (13) yields:

$$loves(mary, tom) \quad (14)$$

Thus the conclusion (14) has been produced in a formal way by application of the inference rules. The example illustrates the concept of *derivability*. As observed, (14) is obtained from (10) and (11) not directly, but in a number of inference steps, each of them adding a new formula to the initial set of premises. Any formula  $F$  that can be obtained in that way from a given set  $P$  of premises is said to be *derivable* from  $P$ . This is denoted by  $P \vdash F$ . If the inference rules are sound it follows that whenever  $P \vdash F$ , then  $P \models F$ . That is, whatever can be derived from  $P$  is also a logical consequence of  $P$ . An important question related to the use of inference rules is the problem of whether all logical consequences of an arbitrary set of premises  $P$  can also be derived from  $P$ . In this case the set of inference rules is said to be *complete*.

**Definition 1.16 (Soundness and Completeness)** A set of inference rules are said to be *sound* if, for every set of closed formulas  $P$  and every closed formula  $F$ , whenever  $P \vdash F$  it holds that  $P \models F$ . The inference rules are *complete* if  $P \vdash F$  whenever  $P \models F$ . ■

A set of premises is said to be *inconsistent* if any formula can be derived from the set. Inconsistency is the proof-theoretic counterpart of unsatisfiability, and when the inference system is both sound and complete the two are frequently used as synonyms.

## 1.5 Substitutions

The chapter is concluded with a brief discussion on *substitutions* — a concept fundamental to forthcoming chapters. Formally a substitution is a mapping from variables of a given alphabet to terms in this alphabet. The following syntactic definition is often used instead:

**Definition 1.17 (Substitutions)** A *substitution* is a finite set of pairs of terms  $\{X_1/t_1, \dots, X_n/t_n\}$  where each  $t_i$  is a term and each  $X_i$  a variable such that  $X_i \neq t_i$  and  $X_i \neq X_j$  if  $i \neq j$ . The *empty substitution* is denoted  $\epsilon$ . ■

The *application*  $X\theta$  of a substitution  $\theta$  to a variable  $X$  is defined as follows:

$$X\theta := \begin{cases} t & \text{if } X/t \in \theta. \\ X & \text{otherwise} \end{cases}$$

In what follows let  $Dom(\{X_1/t_1, \dots, X_n/t_n\})$  denote the set  $\{X_1, \dots, X_n\}$ . Also let  $Range(\{X_1/t_1, \dots, X_n/t_n\})$  be the set of all variables in  $t_1, \dots, t_n$ . Thus, for variables not included in  $Dom(\theta)$ ,  $\theta$  behaves as the identity mapping. It is natural to extend the domain of substitutions to include also terms and formulas. In other words, it is possible to *apply* a substitution to an arbitrary term or formula in the following way:

**Definition 1.18 (Application)** Let  $\theta$  be a substitution  $\{X_1/t_1, \dots, X_n/t_n\}$  and  $E$  a term or a formula. The application  $E\theta$  of  $\theta$  to  $E$  is the term/formula obtained by simultaneously replacing  $t_i$  for every free occurrence of  $X_i$  in  $E$  ( $1 \leq i \leq n$ ).  $E\theta$  is called an *instance* of  $E$ . ■

**Example 1.19**

$$\begin{aligned} p(f(X, Z), f(Y, a))\{X/a, Y/Z, W/b\} &= p(f(a, Z), f(Z, a)) \\ p(X, Y)\{X/f(Y), Y/b\} &= p(f(Y), b) \end{aligned}$$

It is also possible to compose substitutions:

**Definition 1.20 (Composition)** Let  $\theta$  and  $\sigma$  be two substitutions:

$$\begin{aligned} \theta &:= \{X_1/s_1, \dots, X_m/s_m\} \\ \sigma &:= \{Y_1/t_1, \dots, Y_n/t_n\} \end{aligned}$$

The *composition*  $\theta\sigma$  of  $\theta$  and  $\sigma$  is obtained from the set:

$$\{X_1/s_1\sigma, \dots, X_m/s_m\sigma, Y_1/t_1, \dots, Y_n/t_n\}$$

by removing all  $X_i/s_i\sigma$  for which  $X_i = s_i\sigma$  ( $1 \leq i \leq m$ ) and by removing those  $Y_j/t_j$  for which  $Y_j \in \{X_1, \dots, X_m\}$  ( $1 \leq j \leq n$ ). ■

It is left as an exercise to prove that the above syntactic definition of composition actually coincides with function composition (see exercise 1.13).

**Example 1.21**

$$\{X/f(Z), Y/W\}\{X/a, Z/a, W/Y\} = \{X/f(a), Z/a, W/Y\}$$

A kind of substitution that will be of special interest are the so-called idempotent substitutions:

**Definition 1.22 (Idempotent substitution)** A substitution  $\theta$  is said to be *idempotent* iff  $\theta = \theta\theta$ . ■

It can be shown that a substitution  $\theta$  is *idempotent* iff  $Dom(\theta) \cap Range(\theta) = \emptyset$ . The proof of this is left as an exercise and so are the proofs of the following properties:

**Proposition 1.23 (Properties of substitutions)** Let  $\theta$ ,  $\sigma$  and  $\gamma$  be substitutions and let  $E$  be a term or a formula. Then:

- $E(\theta\sigma) = (E\theta)\sigma$
- $(\theta\sigma)\gamma = \theta(\sigma\gamma)$
- $\epsilon\theta = \theta\epsilon = \theta$

■

Notice that composition of substitutions is not commutative as illustrated by the following example:

$$\{X/f(Y)\}\{Y/a\} = \{X/f(a), Y/a\} \neq \{Y/a\}\{X/f(Y)\} = \{Y/a, X/f(Y)\}$$

## Exercises

**1.1** Formalize the following sentences of natural language as formulas of predicate logic:

- a) Every natural number has a successor.
- b) Nothing is better than taking a nap.
- c) There is no such thing as negative integers.
- d) The names have been changed to protect the innocent.
- e) Logic plays an important role in all areas of computer science.
- f) The renter of a car pays the deductible in case of an accident.

**1.2** Formalize the following sentences of natural language into predicate logic:

- a) A bronze medal is better than nothing.
- b) Nothing is better than a gold medal.
- c) A bronze medal is better than a gold medal.

**1.3** Prove Proposition 1.13.

**1.4** Prove the equivalences in connection with Definition 1.15.

**1.5** Let  $F := \forall X \exists Y p(X, Y)$  and  $G := \exists Y \forall X p(X, Y)$ . State for each of the following four formulas whether it is satisfiable or not. If it is, give a model with the natural numbers as domain, if it is not, explain why.

$$(F \wedge G) \quad (F \wedge \neg G) \quad (\neg F \wedge \neg G) \quad (\neg F \wedge G)$$

**1.6** Let  $F$  and  $G$  be closed formulas. Show that  $F \equiv G$  iff  $\{F\} \models G$  and  $\{G\} \models F$ .

**1.7** Show that  $P$  is unsatisfiable iff there is some closed formula  $F$  such that  $P \models F$  and  $P \models \neg F$ .

- 1.8 Show that the following three formulas are satisfiable only if the interpretation has an infinite domain

$$\begin{aligned} & \forall X \neg p(X, X) \\ & \forall X \forall Y \forall Z (p(X, Y) \wedge p(Y, Z) \supset p(X, Z)) \\ & \forall X \exists Y p(X, Y) \end{aligned}$$

- 1.9 Let  $F$  be a formula and  $\theta$  a substitution. Show that  $\forall F \models \forall(F\theta)$ .
- 1.10 Let  $P_1, P_2$  and  $P_3$  be sets of closed formulas. Redefine  $\models$  in such a way that  $P_1 \models P_2$  iff every formula in  $P_2$  is a logical consequence of  $P_1$ . Then show that  $\models$  is transitive — that is, if  $P_1 \models P_2$  and  $P_2 \models P_3$  then  $P_1 \models P_3$ .
- 1.11 Let  $P_1$  and  $P_2$  be sets of closed formulas. Show that if  $P_1 \subseteq P_2$  and  $P_1 \models F$  then  $P_2 \models F$ .
- 1.12 Prove Proposition 1.23.
- 1.13 Let  $\theta$  and  $\sigma$  be substitutions. Show that the composition  $\theta\sigma$  is equivalent to function composition of the mappings denoted by  $\theta$  and  $\sigma$ .
- 1.14 Show that a substitution  $\theta$  is idempotent iff  $Dom(\theta) \cap Range(\theta) = \emptyset$ .
- 1.15 Which of the following statements are true?
- if  $\sigma\theta = \delta\theta$  then  $\sigma = \delta$
  - if  $\theta\sigma = \theta\delta$  then  $\sigma = \delta$
  - if  $\sigma = \delta$  then  $\sigma\theta = \delta\theta$



# 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:

$$\text{child}(\text{tom}, \text{john}) \quad (1)$$

$$\text{child}(\text{ann}, \text{tom}) \quad (2)$$

$$\text{child}(\text{john}, \text{mark}) \quad (3)$$

$$\text{child}(\text{alice}, \text{john}) \quad (4)$$

Applying this notation to the final sentence yields:

$$\begin{aligned} &\text{“For all } X \text{ and } Y, \text{ grandchild}(X, Y) \text{ if} \\ &\quad \text{there exists a } Z \text{ such that } \text{child}(X, Z) \text{ and } \text{child}(Z, Y)\text{”} \end{aligned} \quad (5)$$

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 “ $\leftarrow$ ”:

$$\forall X \forall Y (\text{grandchild}(X, Y) \leftarrow \exists Z (\text{child}(X, Z) \wedge \text{child}(Z, Y))) \quad (6)$$

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

$$\forall X \forall Y (\text{grandchild}(X, Y) \vee \neg \exists Z (\text{child}(X, Z) \wedge \text{child}(Z, Y)))$$

$$\forall X \forall Y (\text{grandchild}(X, Y) \vee \forall Z \neg (\text{child}(X, Z) \wedge \text{child}(Z, Y)))$$

$$\forall X \forall Y \forall Z (\text{grandchild}(X, Y) \vee \neg (\text{child}(X, Z) \wedge \text{child}(Z, Y)))$$

$$\forall X \forall Y \forall Z (\text{grandchild}(X, Y) \leftarrow (\text{child}(X, Z) \wedge \text{child}(Z, Y)))$$

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  $A_0, \dots, A_n$  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:

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

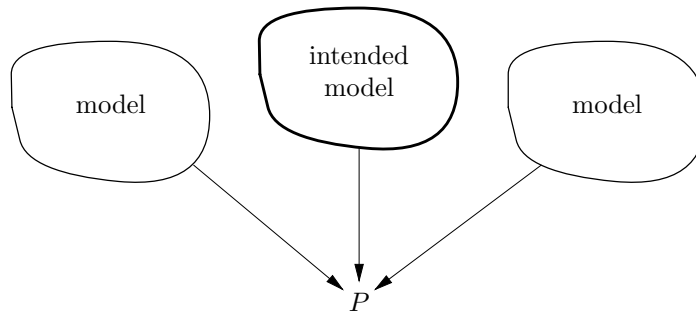
If the body is empty (i.e. if  $n = 0$ ) the implication arrow is usually omitted. Alternatively the empty body can be seen as a nullary connective ■ which is true in every interpretation. (Symmetrically there is also a nullary connective □ which is false in every interpretation.) The first kind of logic 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.

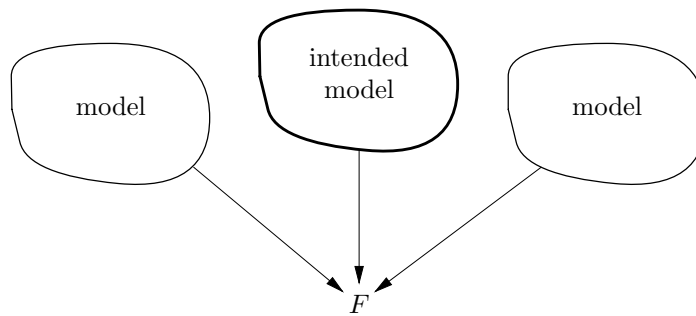
The set of logical consequences of a program is infinite. Therefore the user is expected to *query* the program selectively for various aspects of the intended model. There is an analogy with relational databases — facts explicitly describe elements of the relations while rules give intensional characterization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

$$\forall(\neg(A_1 \wedge \dots \wedge A_m))$$

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

$$\leftarrow A_1, \dots, A_m$$

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

$$\forall X_1 \dots \forall X_n \neg(A_1 \wedge \dots \wedge A_m)$$

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

$$\neg \exists X_1 \dots \exists X_n (A_1 \wedge \dots \wedge A_m)$$

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  $t_1, \dots, t_n$  such that the formula obtained from  $A_1 \wedge \dots \wedge A_m$  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?”	$\leftarrow \text{child}(\text{ann}, \text{tom})$
“Who is a grandchild of Ann?”	$\leftarrow \text{grandchild}(X, \text{ann})$
“Whose grandchild is Tom?”	$\leftarrow \text{grandchild}(\text{tom}, X)$
“Who is a grandchild of whom?”	$\leftarrow \text{grandchild}(X, Y)$

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”;

<sup>1</sup>Of course, formally it is not correct to write  $\leftarrow A_1, \dots, A_m$  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  $\forall(\neg(A_1 \wedge \dots \wedge A_m))$  which is equivalent to  $\forall(\square \vee \neg(A_1 \wedge \dots \wedge A_m \wedge \blacksquare))$ . A nonempty goal can thus be viewed as the formula  $\forall(\square \leftarrow (A_1 \wedge \dots \wedge A_m))$ . The empty goal can be viewed as the formula  $\square \leftarrow \blacksquare$  which is equivalent to  $\square$ .

- 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  $P$ :

$$\begin{aligned} & \text{odd}(s(0)). \\ & \text{odd}(s(s(X))) \leftarrow \text{odd}(X). \end{aligned}$$

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

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

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

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

**Example 2.6** Consider the following definite program  $P$ :

$$\begin{aligned} & \text{owns}(\text{owner}(\text{corvette}), \text{corvette}). \\ & \text{happy}(X) \leftarrow \text{owns}(X, \text{corvette}). \end{aligned}$$

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

$$\{\text{corvette}, \text{owner}(\text{corvette}), \text{owner}(\text{owner}(\text{corvette})), \dots\}$$

and the Herbrand base  $B_P$  of the set:

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

**Definition 2.7 (Herbrand interpretations)** A Herbrand interpretation of  $P$  is an interpretation  $\mathfrak{S}$  such that:

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

$$f_{\mathfrak{S}}(x_1, \dots, x_n) := f(x_1, \dots, x_n)$$

That is, the function  $f_{\mathfrak{S}}$  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  $p_{\mathfrak{S}}$  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  $\mathfrak{S}$  the meaning  $p_{\mathfrak{S}}$  of  $p$  consists of the following set of  $n$ -tuples:  $\{\langle t_1, \dots, t_n \rangle \in U_P^n \mid \mathfrak{S} \models p(t_1, \dots, t_n)\}$ .

**Example 2.8** One possible interpretation of the program  $P$  in Example 2.5 is  $odd_{\mathfrak{S}} = \{\langle s(0) \rangle, \langle s(s(0)) \rangle\}$ . 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  $\mathfrak{S}$  can be seen as a subset of the Herbrand base (or a possibly infinite relational database), namely  $\{A \in B_P \mid \mathfrak{S} \models A\}$ .

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

$$\begin{aligned} \mathfrak{S}_1 &:= \emptyset \\ \mathfrak{S}_2 &:= \{odd(s(0))\} \\ \mathfrak{S}_3 &:= \{odd(s(0)), odd(s(s(0)))\} \\ \mathfrak{S}_4 &:= \{odd(s^n(0)) \mid n \in \{1, 3, 5, 7, \dots\}\} \\ &= \{odd(s(0)), odd(s(s(0))), \dots\} \\ \mathfrak{S}_5 &:= B_P \end{aligned}$$

**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  $\mathfrak{S}$  is a model of a universally quantified formula  $\forall F$  it suffices to check if all ground instances of  $F$  are true in  $\mathfrak{S}$ . For instance, to check if  $A_0 \leftarrow A_1, \dots, A_n$  is true in  $\mathfrak{S}$  it suffices to show that if  $(A_0 \leftarrow A_1, \dots, A_n)\theta$  is a ground instance of  $A_0 \leftarrow A_1, \dots, A_n$  and  $A_1\theta, \dots, A_n\theta \in \mathfrak{S}$  then  $A_0\theta \in \mathfrak{S}$ .

**Example 2.11** Clearly  $\mathfrak{S}_1$  cannot be a model of  $P$  in Example 2.5 as it is not a Herbrand model of  $odd(s(0))$ . However,  $\mathfrak{S}_2, \mathfrak{S}_3, \mathfrak{S}_4, \mathfrak{S}_5$  are all models of  $odd(s(0))$  since  $odd(s(0)) \in \mathfrak{S}_i$ , ( $2 \leq i \leq 5$ ).

Now,  $\mathfrak{S}_2$  is not a model of  $odd(s(s(X))) \leftarrow odd(X)$  since there is a ground instance of the rule — namely  $odd(s(s(0))) \leftarrow odd(s(0))$  — such that all premises are true:  $odd(s(0)) \in \mathfrak{S}_2$ , but the conclusion is false:  $odd(s(s(0))) \notin \mathfrak{S}_2$ . By a similar reasoning it follows that  $\mathfrak{S}_3$  is not a model of the rule.

However,  $\mathfrak{S}_4$  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  $odd(t) \notin \mathfrak{S}_4$  (check with Definition 1.6). Furthermore, if  $odd(t) \in \mathfrak{S}_4$  then it must also hold that  $odd(s(s(t))) \in \mathfrak{S}_4$  (cf. the the definition of  $\mathfrak{S}_4$  above) and hence  $odd(s(s(t))) \leftarrow odd(t)$  is true in  $\mathfrak{S}_4$ . Similar reasoning proves that  $\mathfrak{S}_5$  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  $\mathfrak{S}'$  is a model of  $P \cup \{G\}$  then  $\mathfrak{S} := \{A \in B_P \mid \mathfrak{S}' \models A\}$  is a Herbrand model of  $P \cup \{G\}$ . ■

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

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

which is not true in  $\mathfrak{S}$  ( $A_0 = \square$  in case of a goal).

Since this clause is false in  $\mathfrak{S}$  then  $A_1, \dots, A_m$  are all true and  $A_0$  is false in  $\mathfrak{S}$ . Hence, by the definition of  $\mathfrak{S}$  we conclude that  $A_1, \dots, A_m$  are true and  $A_0$  is false in  $\mathfrak{S}'$ . This contradicts the assumption that  $\mathfrak{S}'$  is a model. Hence  $\mathfrak{S}$  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.<sup>2</sup>

**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  $\{\langle 1 \rangle, \langle 3 \rangle, \langle 5 \rangle, \dots\}$  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  $A_0 \leftarrow A_1, \dots, A_m$  in  $P$ . Clearly, all  $A_0, \dots, A_m$  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.

<sup>2</sup>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  $\mathfrak{S} := \bigcap M$  is a Herbrand model of  $P$ . ■

*Proof:* Assume that  $\mathfrak{S}$  is not a model of  $P$ . Then there exists a ground instance of a clause of  $P$ :

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

which is not true in  $\mathfrak{S}$ . This implies that  $\mathfrak{S}$  contains  $A_1, \dots, A_m$  but not  $A_0$ . Then  $A_1, \dots, A_m$  are elements of every interpretation of the family  $M$ . Moreover there must be at least one model  $\mathfrak{S}_i \in M$  such that  $A_0 \notin \mathfrak{S}_i$ . Thus  $A_0 \leftarrow A_1, \dots, A_m$  is not true in this  $\mathfrak{S}_i$ . Hence  $\mathfrak{S}_i$  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:

$$\begin{aligned} &\{male(adam), female(eve)\} \\ &\{male(adam), male(eve), female(eve)\} \\ &\{male(adam), female(eve), female(adam)\} \\ &\{male(adam), male(eve), female(eve), female(adam)\} \end{aligned}$$

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  $p(t_1, \dots, t_n)$  is a logical consequence of the program iff, in the intended interpretation  $\mathfrak{S}$ ,  $t_i$  denotes the individual  $x_i$  and  $\langle x_1, \dots, x_n \rangle \in p_{\mathfrak{S}}$ . 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  $\mathfrak{S}'$  of  $P$ . But we know (see Theorem 2.12) that the set  $\mathfrak{S}$  of all ground atomic formulas which are true in  $\mathfrak{S}'$  is a Herbrand model of  $P$ . Hence  $A$  cannot be an element of  $\mathfrak{S}$ . 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  $\{p(a)\} \cap \{q(b)\} = \emptyset$  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  $f : \mathcal{D} \rightarrow \mathcal{D}$  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  $\mathfrak{S}$  does not include a ground instance of a fact  $A$  of the program then  $A$  is not true in  $\mathfrak{S}$  and  $\mathfrak{S}$  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  $\mathfrak{S}$  includes  $A_1\theta, \dots, A_m\theta$  it must also include  $A_0\theta$  in order to be a model.

Consider the set  $\mathfrak{S}_1$  of all ground instances of facts in the program. It is now possible to use every instance of each rule to augment  $\mathfrak{S}_1$  with new elements which necessarily must belong to every model. In that way a new set  $\mathfrak{S}_2$  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  $\mathfrak{S}_{i+1}$  are those which *must follow immediately* from  $\mathfrak{S}_i$ .

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:

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

■

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

$$\emptyset, T_P(\emptyset), T_P(T_P(\emptyset)), T_P(T_P(T_P(\emptyset))), \dots$$

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

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

The following example illustrates the construction:

**Example 2.19** Consider again the program of Example 2.5.

$$\begin{aligned} T_P \uparrow 0 &= \emptyset \\ T_P \uparrow 1 &= \{\text{odd}(s(0))\} \\ T_P \uparrow 2 &= \{\text{odd}(s(s(s(0))))\}, \text{odd}(s(0))\} \\ &\vdots \\ T_P \uparrow \omega &= \{\text{odd}(s^n(0)) \mid n \in \{1, 3, 5, \dots\}\} \end{aligned}$$

■

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  $A_0 \leftarrow A_1, \dots, A_m$ :

$$\begin{aligned} & \forall X(p(X) \vee \neg q(X)) \\ & \forall X(p(X) \vee \neg \exists Y(q(X, Y) \wedge r(X))) \\ & \forall X(\neg p(X) \vee (q(X) \supset r(X))) \\ & \forall X(r(X) \supset (q(X) \supset p(X))) \end{aligned}$$

**2.2** Formalize the following scenario as a definite program:

Basil owns Fawly Towers. Basil and Sybil are married. Polly and Manuel are employees at Fawly Towers. Smith and Jones are guests at Fawly 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  $\mathfrak{S}$  be the Herbrand interpretation:

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

Which of the following formulas are true in  $\mathfrak{S}$ ?

- (1)  $\forall X p(X)$
- (2)  $\forall X q(X)$
- (3)  $\exists X (q(X) \wedge p(X))$
- (4)  $\forall X (q(X) \supset p(X))$
- (5)  $\forall X (p(X) \supset q(X))$

**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:

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

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

**2.9** Let  $P$  be a definite program and  $\mathfrak{S}$  a Herbrand interpretation. Show that  $\mathfrak{S}$  is a model of  $P$  iff  $T_P(\mathfrak{S}) \subseteq \mathfrak{S}$ .