3
Roots of Answer Set Prolog

Answer Set Prolog is a comparatively new knowledge representation (KR) language with roots in older nonmonotonic logics and the logic programming language Prolog. Early proponents of the logical approach to artificial intelligence believed that the classical logical formalism called *first-order logic* would serve as the basis for the application of the axiomatic method to the development of intelligent agents. In this chapter we briefly describe some important developments that forced them to question this belief and to work instead on the development of nonclassical knowledge representation languages including ASP. To make the chapter easier for people not familiar with mathematical logic, we give a very short introduction to one of its basic logical tools – first-order logic.

3.1 First-Order Logic (FOL)

First-order logic is a formal logical system that consists of a formal language, an entailment or consequence relation for this language, and a collection of inference rules that can be used to obtain these consequences. The language of FOL is parametrized with respect to a signature Σ. The notions of term and atom over Σ are the same as those defined in Section 2.1. The statements of FOL (called FOL formulas) are built from atoms using boolean logical connectives and quantifiers $\forall$ (for all) and $\exists$ (there exists). Atoms are formulas. If A and B are formulas and X is a variable, then $(A \wedge B)$, $(A \vee B)$, $(A \supset B)$, $\neg A$, $\forall X\, A$, $\exists X\, A$ are formulas. An occurrence of a variable X in a formula A is called *bound* if it belongs to a subformula of A that has the form $\forall X\, F$ or $\exists X\, F$; otherwise it is free. Formulas without free occurrences of variables are called *sentences*. $\forall X\, p(X)$ is a sentence; $p(X)$ is an FOL formula that is not a sentence. Sometimes, the latter are referred to as *conditions* on X. To simplify further presentation, we assume that, for every variable X, all occurrences of X in a formula are either free or bound, exclusively. This assumption eliminates formulas such as $(\forall X(p(X) \lor q(X)) \lor r(X))$ where the last occurrence of X is free and the first two are bound. A set of FOL sentences is often referred to as an FOL *theory*.

To illustrate, let's take our family example from Chapter 1. Its FOL signature, atoms, and terms are identical to the those given in Section 2.1. Here are some formulas made from the signature's symbols:

$$(father(john, sam) \land (mother(alice, sam) \lor mother(alice, john)))$$

$$\forall X, Y\ (parent(Y, X) \supset child(X, Y))$$

$$\neg\exists X\ gender\_of(X, male)$$

$$gender\_of(X, male)$$

The first three are sentences. The last one is a condition on X.

Note that so far we have only talked about the syntax of FOL. Nothing was said about the semantics (meaning) of these formulas and their truth or falsity. The meaning of a first-order formula depends on the interpretation of its nonlogical symbols. Such an interpretation is normally defined by specifying a nonempty set U called the *universe* and a mapping I that maps

- every object constant c of Σ into an element $I(c)$ of U,
- every function constant f of arity n into a function $I(f) : U^n \rightarrow U$,
- every predicate constant p of arity n into a subset $I(p)$ of U^n.

Using the first two clauses of this definition, function I can be naturally extended to arbitrary terms. Whenever convenient, we refer to an interpretation by its second element I.

To illustrate this notion let us consider the family example above, and define an interpretation, say I_1, of its symbols.

Suppose that our universe U_1 of I_1 consists of a group of four people and a group of two genders. Let us denote elements of U_1 by $d_1, \ldots, d_4, m, f$ where the ds stand for people and m and f for genders. (Note that these symbols do not belong to the signature of our FOL theory. Sometimes they are referred to as *meta-symbols*.) Mapping of object constants into elements of the universe can be viewed as naming these elements within the language. Making the value of $I_1(alice)$ to be d_1 simply means that, in our interpretation, the first person in the group has the name *alice*. The same goes for other constants. Let us assume that $I_1(john) = d_2$, $I_1(sam) = d_3$, $I_1(female) = f$, and $I_1(male) = m$. Let us also assume that our interpretation I_1 maps predicate *father* of arity 2 into the set $\{\langle d_2, d_3\rangle\}$. Intuitively this means the only people in the group that satisfy the father-child relation are d_2 and d_3. The same goes for $mother$ and $gender\_of$.

I_1 is, of course, not the only possible interpretation. Interpretation I_2 may have the same universe and differ from I_1 only in its mapping of $father$: $I_2(father) = \{\langle d_4, d_3 \rangle\}$. Other interpretations may have completely different universes and mappings.[1]

Now we can define the truth value of an FOL sentence in interpretation I with universe U. First we expand the signature Σ by a collection of constant symbols, one for each element of the universe U; let's say that for each $d \in U$ the constant symbol c_d is added. The interpretation is extended so that each new constant symbol is assigned to its corresponding element of the domain.

Next we define the truth of sentences by induction on the definition of a sentence. A ground atom $p(t_1, \ldots, t_n)$ of the new signature is said to be true in I if $\langle I(t_1), \ldots, I(t_n) \rangle \in I(p)$. If A and B are FOL sentences then $\neg A$ is true in I if A is not true in I; $A \wedge B$ is true in I if both A and B are true in I; $\forall\ X\ A$ is true in I if for every $d \in U$, $A(c_d)$ is true in I. (Note that here by $A(c_d)$ we denote the result of replacing all occurrences of X in A by c_d.) An interpretation I satisfies a sentence F or is a *model* of F ($I \models F$) if F is true in I. An FOL theory T is *satisfiable* (or *consistent*) if it has a model. Finally, a sentence F is a *consequence* of theory T ($T \models F$) if F is true in all the models of T; $T \models F$ is often referred to as the *FOL entailment relation* and is read as T entails F. In what follows we often use a version of first-order logic whose syntax includes a special relation $=$. An interpretation I satisfies $t_1 = t_2$ iff I maps t_1 and t_2 to the same element of U.

By this definition, under interpretation I_1, $father(john, sam)$ is true, $father(sam, john)$ is false, $father(john, sam) \vee father(sam, john)$ is true, $\exists X\ father(X, sam)$ is true, $\forall X\ father(X, sam)$ is false, and so on. Under I_2, these formulas' truth values would become false, false, false, true, false, respectively.

Now suppose we have the following theory:

$$father(john, sam)$$
$$\neg father(sam, john)$$

[1] Note that this view of the universe is quite different from that which we intuitively use in logic programming where we normally have one (intended) universe. In our example the intended universe contains exactly three people and two genders (each uniquely named by the corresponding constant in the language). In some cases the ability to have arbitrary universes is very important. This is especially true for representing mathematical knowledge.

You can see that this theory is satisfiable because there exists an interpretation, for example I_1, in which all its sentences are true. Note that this theory entails $\neg(father(john, sam) \wedge father(sam, john))$, but does not entail $\neg(father(alice, sam))$ because there is an interpretation in which the parameters of predicate $father$ are assigned this way.

In addition to the definition of formula and consequence relation, first-order logic normally includes a collection of inference rules – syntactic rules that usually consist of premises and a conclusion. The inference rules of first-order logic are *sound* (i.e., they preserve the truth of first-order sentences). Probably the most famous example of such a rule is *modus ponens*:

$$\frac{A,\ A \supset B}{B}$$

which says that if A is true and $A \supset B$ is true then B is also true. We say that a sentence F is derived from a theory T using a collection of inference rules R ($T \vdash_R F$) if there is a sequence of sentences $F_1, \ldots, F_n$ such that $F_n = F$ and every F_i is an axiom of T or is obtained from some previous elements of the sequence by an inference rule from R. We refer to such a sequence as a *derivation* of F from T in R. A collection R of first-order inference rules is called *complete* if for every theory T and formula $F, T \models F$ iff $T \vdash_R F$. There are a number of complete collections of inference rules of first-order logic. In Chapter 12 we discuss a logic programming version of an inference rule, called *resolution*, which is often used in automated reasoning systems.

This theoretical machinery allows us to refine our notion of the axiomatic method. We can view axioms as a theory T – a collection of FOL sentences. A sentence F is a consequence of T if $T \models F$. A mathematical proof of F can be viewed as shorthand for a derivation of F from T using some complete collection of inference rules. First-order logic provides a powerful tool for knowledge representation and reasoning. It has been shown to be sufficient for formalization of a very large part of contemporary mathematical knowledge and reasoning.

3.2 Nonmonotonic Logics

Surprisingly, further experience of applying the logic-based approach to the design of intelligent agents showed that first-order logic may not be a fully adequate tool for representing nonmathematical (especially commonsense) knowledge. The main problem is the difficulty of dealing with defeasible (or nonmonotonic) reasoning. Many researchers agree with the *Stanford Encyclopedia of Philosophy*, which states,

One of the most significant developments both in logic and artificial intelligence is the emergence of a number of non-monotonic formalisms, which were devised expressly for the purpose of capturing defeasible reasoning in a mathematically precise manner.

Among the pioneers of the field in the late 1970s were John McCarthy, Drew McDermott and Jon Doyle, and Raymond Reiter. An influential logical system developed by McCarthy is called **circumscription**. It is based on classical second-order logic and allows elegant model-theoretic characterization. McDermott and Doyle based their *nonmonotonic logics* on modal logics capturing the notion of belief. (A few years later Robert Moore introduced a particular logic of this type, called **autoepistemic logic**, which, among many other important things, served as a starting point for the development of the stable model semantics of the original Prolog. This semantics was the precursor of answer set semantics for more general logic programs we use in this book.) Reiter's formalism called **default logic** expands classical logic by allowing defeasible rules somewhat similar to those of ASP. The papers on circumscription, McDermott and Doyle's nonmonotonic logics, and default logic appeared in the same issue of the *Artificial Intelligence Journal* (vol. 13, 1980) dedicated to these new formalisms. Many people view this publication as the birth of nonmonotonic logic. In what follows we give a brief introduction to these formalisms.

3.2.1 Circumscription

Circumscription is a nonmonotonic logic based on the notion of *minimal entailment*. The basic idea is as follows. Let Σ be a first-order signature and $<$ be a partial order defined on interpretations of Σ. A model M is a *$<$-minimal model* of a first-order theory T if there is no other model M_0 of T such that $M_0 < M$. We say that T minimally entails formula F with respect to $<$ (or that $<$-circumscription of T entails F), and write $T \models_{min(<)} F$ or $circ(T, <) \models F$, if F is true in all *$<$-minimal* models of T.

Here is a simple example. Let Σ be a signature containing two object constants a and b and predicate constants p and q, and let T be a theory consisting of axioms

$$a \neq b$$
$$\forall X(X = a \lor X = b)$$
$$p(a)$$
$$q(a)$$

and $<_p$ be a partial order on interpretations of Σ defined as follows: $I_1 <_p I_2$ if I_1 and I_2 have the same universe and the same mapping of terms, and $I_1(p) \subset I_2(p)$. We show that

$$T \models_{<_p} \neg p(b)$$
$$T \not\models_{<_p} \neg q(b)$$
$$T \models_{<_p} \forall X\, X \neq b \supset q(X).$$

We start by finding $<_p$-minimal models of T. To satisfy the first two axioms of T, the universe of model M should consist of exactly two elements. To satisfy the third axiom, $M(p)$ should be equal to $\{M(a)\}$ or $\{M(a), M(b)\}$. However, the latter model is not minimal with respect to $<_p$ and should be discarded. $M(q)$ should be equal to $\{M(a)\}$ or $\{M(a), M(b)\}$. Since minimization of q is not required by relation $<_p$, both models are minimal. In more detailed notation a model of T has the universe consisting of two distinct elements, say, e_1 and e_2, with constants a and b mapped into these elements; e.g., $U = \{e_1, e_2\}$, $M(a) = e_1$ and $M(b) = e_2$. The corresponding $<_p$-minimal models M_1 and M_2 are obtained from M as follows:

$$M_1(p) = \{e_1\}$$
$$M_1(q) = \{e_1\}$$

and

$$M_2(p) = \{e_1\}$$
$$M_2(q) = \{e_1, e_2\}.$$

Note that strictly speaking we have an infinite collection of such models since $M(a)$ can be equal to e_2 and $M(b)$ to e_1, and moreover, elements of U can be arbitrary pairs. But it can be easily shown that such models are isomorphic to M_1 and M_2 and can be ignored.

This proves the above entailments. Of course, if we expand our theory by $p(b)$ the new theory entails $p(b)$. One can check that

$$T \cup \{p(b)\} \models_{<_p} p(b);$$

i.e., $p(b)$ is a consequence of the new theory. As expected, the new entailment relation is nonmonotonic.

It may be instructive to see what happens if we remove from T some of its equality axioms. Let T_1 be obtained from T by removing the first axiom. The new theory has a model M whose universe consists of one element, say e; the model maps both constants of the language into this element – $M(a) = M(b) = e$; both p and q are mapped into $\{e\}$; i.e., $M(p) = M(q) = \{e\}$. This implies that

$$T_1 \not\models_{<_p} \neg p(b).$$

Now let T_2 be obtained from the original theory T by removing the second axiom. T_2 has a $<_p$-minimal model M with the universe $U = \{a, b, c\}$, $M(a) = a$, $M(b) = b$, $M(p) = \{a\}$, and $M(q) = \{a, b\}$. This time we have

$$T_2 \not\models_{<_p} \forall X \, (X \neq b \supset q(X)).$$

Circumscription is a powerful nonmonotonic formalism that remains the language of choice for a number of researchers interested in knowledge representation and reasoning. Its minimality idea clearly influenced the development of ASP as well as other nonmonotonic formalisms. Recent work established a close mathematical connection between circumscription and some powerful generalizations of ASP, but full comprehension of the meaning of these results may require some additional nontrivial insights.

3.2.2 Autoepistemic Logic

Formulas of **autoepistemic logic** are built from propositional atoms using propositional connectives and the modal operator B. For instance, formula $Bp \supset p$ says that if p is believed then p is true, and so on. The semantics of autoepistemic logic is given via a notion of stable expansion.

Definition 3.2.1. *(Stable Expansion)*
For any sets T and E of autoepistemic formulas, E is said to be a **stable expansion** *of T iff*

$$E = Cn(T \cup \{B\phi : \phi \in E\} \cup \{\neg B\psi : \psi \notin E\})$$

where Cn is a propositional consequence operator.

Intuitively, T is a set of axioms and E is a possible collection of a reasoner's beliefs determined by T. A formula F is said to be *true* in T (or entailed by T) if F belongs to all stable expansions of T. If T does not contain the modal operator B, T has a unique stable expansion denoted by $Th(T)$. For instance, if $T = \{p, q \vee r\}$ then $Th(T)$ contains T together with an infinite collection of other formulas, including Bp, $\neg Bq$, $\neg Br$, $B(q \vee r)$, $B\neg Bq$, etc. There is a close connection between classes of autoepistemic theories and classes of theories of ASP. We describe one such connection that served as a starting point for the development of stable model semantics of logic programs.

Consider a class G of programs of Answer Set Prolog that consists of rules of the form:

$$\begin{array}{ll} (i) & p_0 \leftarrow p_1, \ldots, p_m,\ not\ p_{m+1}, \ldots,\ not\ p_n \\ (ii) & \neg p \leftarrow\ not\ p \quad \text{(for every atom } p) \end{array} \tag{3.1}$$

where $0 \leq m \leq n$ and the ps are atoms. Let α be a mapping that maps rules (i) and (ii) into autoepistemic formulas:

$$\begin{array}{ll} \alpha(i) & p_1 \wedge \cdots \wedge p_m \wedge \neg B\, p_{m+1} \wedge \cdots \wedge \neg B\, p_n \supset p_0 \\ \alpha(ii) & \neg B\, p \supset \neg p \end{array} \tag{3.2}$$

and let

$$\alpha(\Pi) = \{\alpha(r) : r \in \Pi\}.$$

Proposition 3.2.1. *For any program $\Pi \in G$, and any set A of literals in the language of Π, A is an answer set of Π iff $Th(A)$ is a stable expansion of $\alpha(\Pi)$. Moreover, every stable expansion of $\alpha(\Pi)$ can be represented in the above form.*

A similar proposition proven in 1987 for stratified logic programs showed that, at least in a simple case, default negation can be interpreted as an epistemic operator. This connection played an important role in the later development of stable model semantics. There are other interesting mappings of programs of Answer Set Prolog into autoepistemic logic and its variants, but none seem to provide a really good explanation of the meanings of the *or* and $\leftarrow$ connectives of Answer Set Prolog in terms of autoepistemic logic.

3.2.3 Reiter's Default Theories

A Reiter's *default* is an expression of the form

$$\frac{p : M\ j_1, \ldots, M\ j_n}{f} \tag{3.3}$$

where p, f, and js are quantifier-free first-order formulas; f is called the *consequent* of the default, p is its *prerequisite*, and js are its *justifications*. A default may have no prerequisite or no justification. An expression $M\ j$ is interpreted as "it is consistent to believe j." A pair $\langle D, W\rangle$ where D is a set of defaults and W is a set of first-order sentences is called Reiter's **default theory**.

Definition 3.2.2. *(Extension of a Default Theory)*
Let $\langle D, W\rangle$ be a default theory and E be a set of first-order sentences. Consider $E_0 = W$ and, for $i \geq 0$, let D_i be the set of defaults of form (3.3) from D such that $p \in E_i$ and $\neg j_1 \notin E, \dots, \neg j_n \notin E$. Finally, let $E_{i+1} = Th(E_i) \cup \{conseq(\delta) : \delta \in D_i\}$ where $Th(E_i)$ is the set of all classical consequences of E_i and $conseq(\delta)$ denotes δ's consequent. The set E is called an **extension** *of $\langle D, W\rangle$ if*

$$E = \bigcup_0^\infty E_i.$$

Extensions of a default theory D play a role similar to that of stable expansions of autoepistemic theories. The simple mapping α from programs of ASP without disjunction to default theories identifies a rule r

$$l_0 \leftarrow l_1, \dots, l_m,\ not\ l_{m+1}, \dots,\ not\ l_n$$

with the default $\alpha(r)$

$$\frac{l_1 \wedge \dots \wedge l_m : M\ \overline{l}_{m+1}, \dots, M\ \overline{l}_n}{l_0} \tag{3.4}$$

(Recall that $\overline{l}$ stands for the literal complementary to l.)

Proposition 3.2.2. *For any nondisjunctive program Π of ASP*

(i) *if S is an answer set of Π, then $Th(S)$ is an extension of $\alpha(\Pi)$;*
(ii) *for every extension E of $\alpha(\Pi)$ there is exactly one answer set S of Π such that $E = Th(S)$.*

Thus, the class of nondisjunctive ASP programs can be identified with the class of default theories with empty W and defaults of the form (3.4). Perhaps somewhat surprisingly, the proposition is not easily generalized to a program with disjunction. One of the problems in finding a natural translation from arbitrary ASP programs to default theories is related to these theories' inability to use defaults with empty justifications in reasoning by cases. The default theory with

$$D = \left\{ \frac{q :}{p}, \quad \frac{r :}{p} \right\}$$

and

$$W = \{q \vee r\}$$

does not have an extension containing p and therefore does not entail p. The corresponding logic program

$$p \leftarrow q$$
$$p \leftarrow r$$
$$q \ or \ r.$$

has two answer sets, $\{p, q\}$ and $\{p, r\}$, and hence entails p.

3.3 ASP and Negation in Logic Programming

In the late 1970s and early 1980s different approaches to nonmonotonicity were investigated in the area of logic programming. The goal was to give a declarative semantics for the **negation as failure**[2] of the logic-based programming language Prolog. The new connective was initially defined in procedural terms referring to a particular inference mechanism of Prolog called SLDNF resolution (for more details, see Chapter 12). The statement $not \ a$ is viewed as true if SLDNF resolution finitely fails to prove a. The attempts to find suitable declarative semantics of this nonmonotonic connective, together with the work on the general nonmonotonic reasoning formalisms discussed earlier, played a major role in the discovery of ASP and several other logic-programming-based KR formalisms. In this section we briefly outline some important developments in this area. In what follows we limit our attention to logic programs consisting of rules of the form

$$a_0 \leftarrow a_1, \ldots, a_m, \ not \ a_{m+1}, \ldots, \ not \ a_n \tag{3.5}$$

where the as are atoms. For historical reasons we refer to such programs as **normal logic programs** or *nlp*s.

3.3.1 Clark's Completion

The research on finding a declarative semantics for negation as failure in *nlp*s started with the pioneering work of Keith Clark. He suggested that, given an *nlp*, we could view the bodies of rules with a predicate p in their

[2] In other parts of this book, we refer to this as default negation. In this historical context, we have left the name as is.

heads as "sufficiency" conditions for inferring atoms formed by p from the program. Clark stated that the bodies of these rules could also be taken as "necessary" conditions, with the result that negative information about p could be assumed if none of these conditions are met. More precisely, let us consider the following two-step transformation of an *nlp* Π into a collection of first-order formulas:

Step 1: For every *nlp* rule (3.5) in Π, let $a_0 = p(t_1, \dots, t_k)$ and $Y_1, \dots, Y_s$ be the list of variables appearing in r. By $\alpha_1(r)$ we denote an FOL formula:

$$\begin{array}{l} \exists\, Y_1 \dots Y_s : X_1 = t_1 \wedge \cdots \wedge X_k = t_k \wedge \\ a_1 \wedge \cdots \wedge a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_n \;\supset\; p(X_1, \dots, X_k) \end{array} \tag{3.6}$$

where, $X_1 \dots X_k$ are variables not appearing in r.

$$\alpha_1(\Pi) = \{\alpha_1(r) : r \in \Pi\}$$

Step 2: For each predicate p rename its variables to make all the implications in $\alpha_1(\Pi)$ be of the form

$$\begin{array}{l} E_1 \supset p(X_1, \dots, X_k) \\ \vdots \\ E_j \supset p(X_1, \dots, X_k). \end{array}$$

Next, replace these formulas by

$$\forall\, X_1 \dots X_k : p(X_1, \dots, X_k) \equiv E_1 \vee \cdots \vee E_j$$

if $j \geq 1$ and by

$$\forall\, X_1 \dots X_k : \neg p(X_1, \dots, X_k)$$

if $j = 0$.

Definition 3.3.1. *(Clark's Completion)*
The resulting first-order theory combined with natural axioms for equality called free equality axioms[3] *is called* **Clark's completion** *of* Π *and is*

[3] In addition to the usual equality axioms, free equality axioms include

- $f(X_1, \dots, X_n) \neq g(X_1, \dots, X_n)$ for each distinct pair of function symbols f and g of arity n.
- $t(X) \neq X$ for each term $t(X)$ (other than X) in which X occurs.
- $f(X_1, \dots, X_n) = f(Y_1, \dots, Y_n) \supset X_1 = Y_1 \wedge \cdots \wedge X_n = Y_n$ for each n-ary function symbol f.

These axioms guarantee that ground terms are equal iff they are identical.

denoted by $Comp(\Pi)$. *A literal* l *is* entailed *by* Π *if* l *is the first-order consequence of* $Comp(\Pi)$.

To better understand the construction, let us consider the following example: Let Π be the program

$$\begin{aligned}&p(a).\\&p(b).\\&q(Y) \leftarrow p(Y).\\&r \leftarrow \ not\ s.\end{aligned}$$

Then $\alpha_1(\Pi)$ has the form, say,

$$\begin{aligned}&X_1 = a \supset p(X_1)\\&X_2 = b \supset p(X_2)\\&\exists\, Y(X = Y \wedge p(X)) \supset q(X)\\&\neg s \supset r.\end{aligned}$$

It is easy to check that the third axiom is equivalent to

$$p(X) \supset q(X).$$

To properly prepare for Step 2 in which we collect our formulas under the universal quantifier, we replace X_1 and X_2 in the first two axioms by, say, X. The result, $Comp(\Pi)$, is

$$\begin{aligned}&\forall X : p(X) \equiv X = a \vee X = b\\&\forall X : q(X) \equiv \exists\, Y(X = Y \wedge p(X))\\&r \equiv \neg s\\&\neg s.\end{aligned}$$

The second formula can be simplified to

$$q(X) \equiv p(X).$$

The following theorem establishes the relationship between models of Clark's completion of Π and the notion of a **supported model**. (A set S of atoms is supported by an *nlp* Π if, for every $a \in S$ there is a rule (3.5) such that $a = a_0, a_1, \dots, a_m \in S$ and $a_{m+1}, \dots, a_n \notin S$.)

Theorem 1. *A set* S *of atoms is a model of Clark's completion of* Π *iff* S *is supported and satisfies the rules of* Π.

Models of Clark's completion may obviously differ from answer sets of Π. Program $p \leftarrow p$ has two Clark's models, $\{\,\}$ and $\{p\}$, but only one answer set $\{\ \}$. This should not be surprising – the completion semantics intends to capture the notion of finite failure of a particular inference mechanism, SLDNF resolution, whereas answer set semantics formalizes the more general notion of default negation. It is also important to note that Theorem 1 immediately implies that every answer set of an *nlp* program Π is also a model of the completion of Π, and hence every literal entailed by Π with respect to Clark's semantics is also entailed by Π with respect to the answer set semantics.

The existence of Clark's declarative semantics facilitated the development of the theory of logic programs. It made possible the first proofs of correctness of the inference mechanism of Prolog based on SLDNF resolution, proofs of program equivalence, and some other properties of programs. It is still widely and successfully used for logic programming applications. Unfortunately in many situations Clark's semantics appears too weak. Consider, for instance, the following example:

Example 3.3.1. Suppose that we are given a graph, say,

$$edge(a, b).\ \ edge(c, d).\ \ edge(d, c).$$

and want to describe vertices of the graph reachable from a given vertex a. The natural solution seems to be to introduce these rules:

$$\begin{aligned} &reachable(a).\\ &reachable(X) \leftarrow edge(Y, X),\\ &\qquad\qquad\qquad\qquad reachable(Y). \end{aligned}$$

We clearly expect vertices c and d not to be reachable. However, Clark's completion of the predicate 'reachable' gives only

$$reachable(X) \equiv (X = a \vee \exists\, Y : reachable(Y) \wedge edge(Y, X))$$

from which such a conclusion cannot be derived.

This difficulty was recognized as serious and prompted the development of other logic programming semantics, including that of ASP. Even though now there are comparatively few knowledge representation languages that use Clark's completion as the basis for their semantics, the notion has not lost its importance for KR. As an illustration let us consider its use for computing answer sets of logic programs. We need the following terminology.

Definition 3.3.2. *A* nlp Π *is called* **tight** *if there is a mapping* $||\ ||$ *of ground atoms of* Π *into the set of natural numbers such that for every* *rule (3.5) of* Π

$$||a_0|| > ||a_1||, \ldots, ||a_m|| \tag{3.7}$$

Theorem 2. *If* Π *is tight, then* S *is a model of* $Comp(\Pi)$ *iff* S *is an answer set of* Π.

Theorem 2 is due to François Fages. There are some recent results extending the notions of Clark's completion and of tightness, as well as discovering more general conditions for equivalence of the two semantics. Note that whenever the two semantics of Π are equivalent, Π's answer sets can be computed by satisfiability solvers (specialized inference engines for propositional theories). The connections between answer sets of a program and models of its Clark's completion have been recently used to substantially improve the efficiency of ASP solvers.

3.3.2 Well-Founded Semantics

In the late 1980s there were several attempts to deal with the problems of Clark's semantics of *nlp*. Two of them – stable model and well-founded semantics – are probably most relevant to the use of logic programs for knowledge representation. The stable model semantics has an epistemic character and can be viewed as a variant of ASP semantics in the context of *nlp*. The well-founded semantics of Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf is based on a different idea. It defines the notion of a unique three-valued model of an *nlp* program Π, called Π's *well-founded* model.

For any *nlp* Π, the function Γ_Π from sets of atoms to sets of atoms is defined by equation

$$\Gamma_\Pi(X) = ans(\Pi^X) \tag{3.8}$$

where $ans(\Pi^X)$ is the answer set of the reduct Π^X from the definition of the answer set. It is clear that stable models of Π can be characterized as the fixpoints[4] of Γ_Π. It is not difficult to show that if $X \subset Y$ then $\Gamma_\Pi(Y) \subset \Gamma_\Pi(X)$. This implies that the function Γ^2_Π is monotone and hence has, by the Knaster-Tarski Theorem, a least and a greatest fixpoint.

[4] An element x from the domain of a function f is called a *fixpoint* of f if $f(x) = x$. If f is defined on a collection of sets, then a fixpoint x is called a *least* fixpoint of f if no proper subset of x is a fixpoint of f; similarly, x is called a greatest fixpoint if no proper superset of x is a fixpoint of f.

(Γ^2_Π is shorthand for $\Gamma_\Pi(\Gamma_\Pi(X))$.) Atoms belonging to the least fixpoint of Γ^2_Π are called *well founded* relative to Π. Atoms belonging to the complement of the greatest fixpoint of Γ^2_Π are called *unfounded* relative to Π.

Definition 3.3.3. *A three-valued function that assigns* 1 *(*true*) to atoms well founded relative to* Π*,* 0 *(*false*) to atoms unfounded relative to* Π*, and* $1/2$ *(*undefined*) to all the remaining atoms is called the* **well-founded model** *of* Π*. An atom is a* **well-founded consequence** *of* Π *if it is true in* Π*'s well-founded model.*

From this definition one can easily see that every *nlp* has a unique well-founded model and that every well-founded consequence of Π is also Π's consequence with respect to the stable model semantics. To better understand the difference between the semantics, let us look at several examples.

Example 3.3.2. Consider the following program Π

$$\begin{aligned}
&a \leftarrow \ not\ b.\\
&b \leftarrow \ not\ a.\\
&c \leftarrow a.\\
&c \leftarrow b.
\end{aligned}$$

To construct its well-founded model, we need to find the least fixpoint and the greatest fixpoint of Γ^2_Π. The empty set seems a good candidate for a least fixpoint, so we begin by letting $X = \emptyset$. The reduct $\Pi^\emptyset$ is

$$\begin{aligned}
&a.\\
&b.\\
&c \leftarrow a.\\
&c \leftarrow b.
\end{aligned}$$

and because $\{a, b, c\}$ is the reduct's answer set, $\Gamma_\Pi(\emptyset) = \{a, b, c\}$. Applying Γ_Π to $\Gamma_\Pi(\emptyset)$ gives us the empty set, and thus, $\emptyset$ is a fixpoint of Γ^2_Π. Since the set is empty, there are no well-founded literals in the well-founded model of Π.

The set $\{a, b, c\}$ is the greatest candidate for a fixpoint we can test. The reduct of $\Pi^{\{a,b,c\}}$ is

$$\begin{aligned}
&c \leftarrow a.\\
&c \leftarrow b.
\end{aligned}$$

and $\Gamma_\Pi(\{a, b, c\}) = \emptyset$. Applying Γ_Π again, we get $\Gamma^2_\Pi\{a, b, c\} = \{a, b, c\}$. Thus, $\{a, b, c\}$ is the greatest fixpoint of Π. Its complement is $\emptyset$, so there are no unfounded literals in the well-founded model of Π either. This means that, in the well-founded model, all of the program's literals are undefined.

This is not so for the stable model semantics under which there are two answer sets: $\{a, c\}$ and $\{b, c\}$. You can see that c is true in both models, and thus, under stable model semantics Π entails c, whereas under well-founded semantics it does not. This example demonstrates that well-founded semantics does not support reasoning by cases, whereas this ability is a built-in feature of stable model semantics.

Example 3.3.3. Consider a program Π consisting of the following rule:

$$p \leftarrow \ not\ p.$$

From the standpoint of stable model semantics, it is inconsistent. It has no stable model (and hence the set of stable consequences of Π contains all the *nlp* literals of the language of Π). In contrast, from the standpoint of the well-founded semantics, Π is consistent. It has empty sets of well-founded and unfounded atoms. Therefore, it has the well-founded model in which every *nlp* literal of Π is undefined (i.e., is assigned the value $1/2$). Hence the answer given by the program to query p is *undefined*.

Example 3.3.4. Consider the following program Π :

$$\begin{aligned} p &\leftarrow \ not\ a. \\ p &\leftarrow \ not\ b. \\ a &\leftarrow \ not\ b. \\ b &\leftarrow \ not\ a. \end{aligned}$$

Π has two stable models: $\{p, a\}$ and $\{p, b\}$. As in the previous example, the well-founded model of Π has empty sets of well-founded and unfounded atoms. Hence the well-founded model of Π assigns *undefined* to every *nlp* literal of Π. This means that p is a consequence of Π in the stable model semantics, whereas the answer to p in the well-founded semantics is *undefined*.

Finally, let us look at the following example from Dix (1991).

Example 3.3.5. Consider Π consisting of rules:

$$\begin{aligned} a &\leftarrow \ not\ b. \\ b &\leftarrow c,\ not\ a. \\ c &\leftarrow a. \end{aligned}$$

This program has one answer set $\{a, c\}$, and thus has a and c as its consequences. The well-founded model of Π has no well-founded atoms. Its unfounded atoms are $\{a, b, c\}$, and hence, according to the well-founded semantics, atoms a, b, and c are undefined.

There are large classes of programs for which both well-founded and stable model semantics coincide. For instance, this happens for programs without recursive definitions or, more generally, without recursive definitions through negation (i.e., stratified programs). A careful reader may notice that the programs in the earlier examples do have such definitions. Examples 3.3.2 and 3.3.4 both define a in terms of $not\ b$ and b in terms of $not\ a$. Example 3.3.3 defines p in terms of $not\ p$ and Example 3.3.5 defines a in terms of $not\ b$, and b is defined in terms of c and $not\ a$.

It is important to notice that the SLDNF resolution of Prolog is sound with respect to the well-founded semantics, but it is not complete. Several attempts were made to define variants of SLDNF resolution that compute answers to goals according to the well-founded entailment. One interesting approach, **SLS resolution**, was introduced by Teodor Przymusinski. SLS resolution is based on a type of oracle and, therefore, cannot be viewed as an algorithm. There are, however, several algorithms and systems that can be viewed as SLS-based approximations of the well-founded semantics. One of the most powerful of such systems, **XSB**, expands SLDNF with tabling and loop checking. (It can be found at `www.cs.sunysb.edu/~sbprolog/xsb-page.html`.) Its use allows the avoidance of many of the loop-related problems of Prolog. For instance, XSB's answer to query $reachable(c)$ for the program from Example 3.3.1 will be *no*, whereas Prolog interpreters will loop on this query.

It is not difficult to expand well-founded semantics of *nlp* to programs allowing classical negation. Of course in this case, not every program will be consistent even according to well-founded semantics. ($\{a., \neg a.\}$ is a simple example of a program that is inconsistent under both semantics). However, it proved to be more difficult to find an elegant extension of well-founded semantics to disjunctive programs. There are interesting knowledge representation languages based on the well-founded semantics of logic programs. The future may show if any of those languages will be competitive with ASP as a general KR language or will be preferable to ASP in some special cases. Well-founded semantics has already proven to be very useful for the development of efficient ASP solvers and question-answering systems that are sound with respect to ASP, such as XSB.

Summary

The chapter started with a brief introduction to classical first-order logic with its notions of objects, functions, and relations; the methodology of describing the world in these terms; clear separation of syntax and semantics; and ideas of interpretation, model, and entailment. This was followed by a review of early nonmonotonic formalisms that strongly influenced the development of ASP. We discussed *circumscription*, which uses the language of first-order logic but only considers models minimal with respect to some ordering, and *autoepistemic logic* and *default logic*, which are propositional theories whose semantics are aimed at capturing the notion of rational belief. Next, we gave a brief review of the negation as failure operator of Prolog and Clark's completion – the earliest attempt to give a declarative semantics of *nlp*. We concluded with a section on well-founded semantics that, for programs without disjunction, can be viewed as an alternative to the stable model semantics used in this book. The intent was to allow the reader to better understand the background in which ASP has been developed. Each of the formalisms outlined in this chapter is of independent interest and can be seriously studied using other sources.

References and Further Reading

Gottlob Frege (1879/2002) presented a logical formalism similar to first-order logic. In addition to boolean connectives (Boole 1854), the formalism contained quantified variables, which became a major tool in mathematics and mathematical logic. A good description of FOL and its relation to Knowledge Representation can be found in Lifschitz, Morgenstern, and Plaisted (2008). As a comprehensive introduction to logic for computer scientists, we recommend Nerode and Shore (1997). The original papers on circumscription, autoepistemic logic, and default logic appeared in the same issue of the *Journal of Artificial Intelligence*: McCarthy (1980), McDermott and Doyle (1980), and Reiter (1980). The original formalisms were quickly extended and generalized by multiple authors; see, for instance, Lifschitz (1985) and Moore (1983). The monograph by Marek and Truszczynski (1993) gives a mathematically rigorous but very readable introduction to the field. Procedural definition of the negation as failure operator of Prolog can be found in Lloyd (1987). Clark's completion was first presented in Clark (1978), well-founded semantics in Gelder, Ross, and Schlipf (1991); and stable model semantics in Gelfond and Lifschitz (1988). The last work was based on the discovery of the relationship between negation as failure of Prolog and the belief operator of autoepistemic logic (Gelfond 1987). The relationship between logic programs (without disjunction) and Reiter's default logic was independently established in Gelfond and Lifschitz (1991) and Bidoit and Froidevaux (1987, 1991). A good early (and still very useful) survey of different declarative formalizations of negation as failure can be found in Apt and Bol (1994). SLS resolution, published as a technical report in 1987, appeared as a journal publication in Przymusinski (1995). (See also related work in Ross (1989)). To learn more about the XSB system, one can consult Chen, Swift, and Warren (1995). Jack Minker (1982) expanded the language of logic programs without default negation by disjunction. Theorem 1 first appeared in Marek and Subrahmanian (1989). The notion of tightness (under a different name) together with Theorem 2 was introduced in Fages (1994) and generalized by a number of authors; see, for instance, Erdem and Lifschitz (2003). The relationship between generalized stable model semantics and circumscription is presented in Ferraris, Lee, and Lifschitz (2011).

Exercises

1. Given universe $\{a, t, d, b, c\}$, FOL signature:

$$\mathcal{O} = \{ant,\ tarantula,\ dragonfly,\ butterfly,\ centipede\}$$
$$\mathcal{F} = \{\ \}$$
$$\mathcal{P} = \{insect,\ spider,\ bigger\}$$
$$\mathcal{V} = \{X, Y\}$$

and interpretation

$$\begin{aligned}
&I(ant) = \{a\},\\
&I(tarantula) = \{t\},\\
&I(dragonfly) = \{d\},\\
&I(butterfly) = \{b\},\\
&I(centipede) = \{c\},\\
&I(insect) = \{a, d, b\}\\
&I(spider) = \{t\}\\
&I(bigger) = \{t, a\}
\end{aligned}$$

check if the following formulas are true under this interpretation:

(a) $bigger(butterfly, ant)$
(b) $\forall X\ (insect(X) \supset \neg spider(X))$

(c) $\neg\exists X\ (insect(X) \land spider(X))$
(d) $\forall X\ (insect(X) \lor spider(X))$

2. Consider an FOL theory consisting of all four sentences of Exercise 1. Is it satisfiable? Justify.
3. Given the signature from Exercise 1, show that the theory consisting of sentence

$$\forall X\ insect(X)$$

entails $insect(tarantula)$.

4. Given a signature consisting of object constants a and b, predicate constants p and q, and a partial order $<_p$ as defined in the chapter, use circumscription to find $<_p$-minimal models of theory:

$$a \neq b$$
$$\forall X(X = a \lor X = b)$$
$$p(a) \lor q(a)$$

Assume that the universe consists of two elements, e_1 and e_2, and $M(a) = e_1$ and $M(b) = e_2$.

5. Use the relationship between Answer Set Prolog and autoepistemic logic outlined in Proposition 3.2.1 to find a stable expansion of

$$\neg Bp \supset q$$
$$\neg Bq \supset \neg q$$

6. Use the relationship between Answer Set Prolog and default theory outlined in Proposition 3.2.2 to find an expansion of the default theory given by

$$\left\{ \frac{p(a) : Mp(b)}{p(c)},\quad \frac{M\neg p(a)}{p(a)} \right\}$$

7. Find Clark's completion of predicate *edge* given in Example 3.3.1.
8. Find Clark's completion of the following program:

$$tool(hammer).$$
$$person(fred).$$
$$animal(horse).$$
$$animate(X) \leftarrow person(X).$$
$$animate(X) \leftarrow animal(X).$$
$$inanimate(X) \leftarrow \ not\ animate(X).$$

9. Using the definitions in Section 3.3.2 and given the following program Π:

$$\begin{aligned}&p.\\&q \leftarrow p.\\&r \leftarrow \ not\ q.\end{aligned}$$

   (a) Find the least fixpoint of Γ^2_Π.
   (b) Find the greatest fixpoint of Γ^2_Π.
   (c) Find the well-founded model of Π.