379
N81d
No. 3517

USING EXTENDED LOGIC PROGRAMS TO FORMALIZE COMMONSENSE REASONING

DISSERTATION

Presented to the Graduate Council of the
University of North Texas in Partial
Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

By

Wen-Bing Horng, B.S., M.S.

Denton, Texas

May, 1992

USING EXTENDED LOGIC PROGRAMS TO FORMALIZE COMMONSENSE REASONING

DISSERTATION

Presented to the Graduate Council of the
University of North Texas in Partial
Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

By

Wen-Bing Horng, B.S., M.S.
Denton, Texas

May, 1992

Horng, Wen-Bing, Using Extended Logic Programs to Formalize Commonsense Reasoning. Doctor of Philosophy (Computer Science), May, 1992, 172 pp., 14 illustrations, bibliography, 141 titles.

In this dissertation, we investigate how commonsense reasoning can be formalized by using extended logic programs. In this investigation, we first use extended logic programs to formalize inheritance hierarchies with exceptions by adopting McCarthy's simple abnormality formalism to express uncertain knowledge. In our representation, not only credulous reasoning can be performed but also the ambiguity-blocking inheritance and the ambiguity-propagating inheritance in skeptical reasoning are simulated.

In response to the anomalous extension problem, we explore and discover that the intuition underlying commonsense reasoning is a kind of forward reasoning. The unidirectional nature of this reasoning is applied by many reformulations of the Yale shooting problem to exclude the undesired conclusion. We then identify defeasible conclusions in our representation based on the syntax of extended logic programs. A similar idea is also applied to other formalizations of commonsense reasoning to achieve such a purpose.

One of the biggest problems with the existing formalizations of nonmonotonic reasoning is that they cannot deal with conflicting information. We propose the generalized answer set semantics to resolve conflicts by capturing the idea of dependency-directed backtracking in truth maintenance systems. Also, similar approaches are applied to other formalizations to resolve conflicting information.

Although McCarthy's abnormality formalism is commonly used to represent uncertain knowledge, we find that it has the semantical inconsistency problem in interpreting conclusions and propose a simple method to fix it. We then explore an interesting property called "unless" semantics in commonsense reasoning and enhance the generalized answer set semantics to incorporate this important semantics. It turns out that the transformed logic programs become more coherent and their conclusions become more reasonable.

Since the definition of an answer set of an extended logic program is a generalization of that of stable models of a general logic program, both of them are non-constructive. We present a new approach to compute stable models and a method to compute answer sets based on the computation of stable models, which make our formalization more practical.

ACKNOWLEDGMENTS

I would first like to thank my major advisor, Chao-Chih Yang, for his guidance, encouragement, and support. He introduced me to the fields of logic programming and commonsense reasoning and motivated me to the problems considered in this dissertation. I have learned a great deal from his classes and personal discussions of how to do research independently.

I would also like to thank to the other members of my doctoral committee, Roy Thomas Jacob, Sajal K. Das, James L. Poirot, and particularly Frank Vlach for helpful comments on my earlier drafts.

This work was supported by the Department of Computer Sciences, University of North Texas. It is my pleasure to acknowledge the departmental chairman, Paul S. Fisher, for his generous financial assistantships.

Finally, I wish to thank my family and friends for their supports and expectations. I am grateful to my parents, Chen-Der and Ah-Chau, and would like to dedicate this dissertation to them. They sacrificed their whole lives for bringing up me, teaching me, and encouraged and supported me in pursuing advanced study. I also indebted to my brother and sisters for taking care of my over seventy-year-old parents during my six years' study in the United States. Special thanks to my wife, Shu-Chin, for her patiently looking after me. In addition, I owe a great deal to my lovely daughter, Terry. Due to busy writing my dissertation, I had little time to play with her. Now, I have finished my dissertation, I can have more time to stay with my family.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

Commonsense reasoning prevails in our everyday life. It is one of the important reasoning abilities a human being possesses besides deductive reasoning [24], inductive reasoning [74], probabilistic reasoning [89, 90], fuzzy reasoning [139, 140, 141], and so on. Usually, our knowledge about a specific domain is incomplete, and sometimes we are forced to draw conclusions from it. In this case we may need to make some default assumptions which correspond to our intuition in order to derive reasonable conclusions. Nevertheless, once new information which conflicts with any default assumptions made is learned, we need to withdraw the conclusions previously derived from these defeated assumptions.

A canonical example about flying birds is often used as an illustration in the literature of commonsense reasoning. Since we know "typically, birds fly," if we now only have the information that Tweety is a bird, we may directly jump to the conclusion that Tweety can fly by making the default assumption that Tweety is a normal bird. Later on, if we are told that Tweety is an ostrich, we will retract our previous belief and draw another new conclusion that Tweety cannot fly to reflect the new information and the general rule "normally, ostriches do not fly."

From the above example, we know that commonsense reasoning is a kind of *plausible* reasoning. This kind of reasoning has the property of *nonmonotonicity* and is also called nonmonotonic reasoning in artificial intelligence (AI). Informally, a reasoning system is called nonmonotonic if the set of theorems derived from a theory does not increase when the theory is augmented. This means that some previously derived theorems will be retracted if new knowledge is added. Therefore, the conclusions drawn from nonmonotonic reasoning are tentative; they are defeasible when more complete knowledge is obtained.

1.1. Commonsense Reasoning

The importance of commonsense reasoning in AI had been recognized long before formal theories were proposed. Some early researchers advocated using mathematical logic as formal systems to formalize such a kind of reasoning. Minsky [83], however, invoked the nonmonotonic nature of commonsense reasoning against classical logic and challenged the proponents of logic-based reasoning in AI to formalize the inheritance of prototypical properties and their exceptions, such as the foregoing flying birds example. Since then, the formalization of nonmonotonic reasoning, based on formal logic, has become a whole new field. There are several major formalizations of nonmonotonic reasoning, each based on different concepts: Reiter's *closed world assumption* (CWA) [105] presumes total knowledge of the domain being represented, McDermott and Doyle's *nonmonotonic logic* [73, 72] and Reiter's *default logic* [106] both are grounded in the notion of logic consistency, McCarthy's *circumscription* [70, 71] is based on the notion of minimal models, and Moore's *autoepistemic logic* [84] is built on epistemic logic. In addition, a lot of research work has been done to improve the above formalizations by eliminating their defects or by strengthening their expressive powers. Although so many formal theories are proposed, none of them is superior to the others; each one of them has its own merits and its own demerits.

While AI people were cheered by the progress of the formalization of nonmonotonic reasoning, they were depressed to hear the result from the celebrated paper by

Hanks and McDermott [44, 45]. They presented a simple problem, the famous Yale shooting problem (YSP), in temporal reasoning and examined three of the major formalizations of nonmonotonic reasoning: default logic, circumscription, and nonmonotonic logic. Unfortunately, they found out that all these formalizations generate an undesired, counter-intuitive conclusion, which causes the so-called *anomalous extension* problem. Because of this, two questions will naturally arise: (1) Is there anything going wrong with these nonmonotonic reasoning formalizations? (2) Does the anomalous extension problem just belong to temporal reasoning? For the first question, the answer should be *no*. With careful formulation of commonsense knowledge, the anomalous extensions will not appear. For example, Morris [86], Gelfond [32], and You and Li [138] independently reformalized the YSP based, respectively, on default logic, autoepistemic logic, and circumscription to subtly exclude the undesired extension. However, these reformalizations are going to 'crash' if conflicting information (i.e., information contrary to what we expect) is learned later; they do not even generate any conclusions. From this point of view, we may say that the current nonmonotonic reasoning formalizations are too weak to deal with conflicting information. Thus, research in enhancing this capability is still going on [42, 87].

As to the second question, the answer is absolutely *no*. Without appropriate formulation of commonsense knowledge, unwanted conclusions may also appear in non-temporal reasoning. It seems that some researchers misdiagnosed the problem by regarding the anomalous extension problem as only appearing in temporal reasoning and proposed various methods to cure this situation [117]. However, Morris [86] (and other authors in [71, 108]) observed that this anomalous extension problem occurs not only in temporal reasoning but also in non-temporal reasoning. He also presented an example from taxonomic reasoning which is structurally similar to the YSP and generates an anomalous extension. The most important contribution of his work is that Morris discovered that if the above examples are formalized in terms of a truth maintenance system (TMS) [21], then surprisingly, the undesired extension disappears. This is due to the nonmonotonic characteristic of the TMS and its unidirectional nature of inference [86]. Although the TMS is a powerful reasoning mechanism, it lacks formal semantics. Owing to Morris' new finding, looking for a suitable semantics for the TMS becomes more urgent.

Recently, several proposals have been presented to provide the TMS with proper semantics [31, 92, 103, 104], each based on default logic, autoepistemic logic, or the stable model semantics [33] in logic programming. All these proposals, however, fail to capture the idea of *dependency-directed backtracking*, an important process in the TMS to resolve conflicts. Giordano and Martelli [42] tried to capture this idea by generalizing the stable model semantics, but the anomalous extension of the YSP reappears when the generalized stable model semantics is used. Nevertheless, this unsuccessful attempt gives us a clue that commonsense reasoning may be formalized by logic programming with a suitable semantics, which is less explored in the AI community. The reason for this may be traced back to the history of logic programming.

1.2. Logic Programming

Logic programming was introduced in the early 1970s by Kowalski [55] and Colmerauer *et al.* [18] based on the notion that logic can be used as a programming language. It is a declarative programming following the principle of separation of *logic* and *control* proposed by Kowalski [56, 57]. Ideally, a programmer should be concerned only with the declarative meaning of his or her program, while the procedural aspects of the program's execution are handled automatically by the system. Unfortunately, this ideal has not yet been achieved with current logic programming systems. One of the reasons is the lack of clarity as to what should be the proper declarative semantics of logic programs and, in particular, what should be the meaning of negation in logic programming [97].

Earlier research [128] focused mostly on definite logic programs (i.e., programs do not contain negation in their bodies) because of their simplicity and execution efficiency. For dealing with negation, Clark [16] subsequently proposed an effective procedure called the *negation as failure* rule to implement the CWA in logic programming systems. This rule works well in definite programs, but it encounters several severe problems, such as the *termination* and *floundering* problems, when applied to general logic programs (i.e., programs that contain negation in their bodies) [114, 115, 116]. Recently, Apt *et al.* [3] and Van Gelder [129] independently introduced a broad class of logic programs called *stratified* to alleviate the negation problem and defined a unique 'canonical' model as the declarative semantics for these stratified programs. Beyond stratification, it seems that no canonical models can be defined, for there may exist more than one minimal model and it is hard to choose one from them [33]. To overcome this difficulty, borrowing the idea of stable expansions from autoepistemic logic, Gelfond and Lifschitz [33], among others [8, 95, 130], proposed the *stable model* semantics for non-stratified logic programs.

The problem with traditional logic programming is that total knowledge is presumed because the negation as failure rule automatically applies to every negative literal. Hence, it does not allow one to deal directly with incomplete information. In order to cope with this problem, Gelfond and Lifschitz [34] further extended their stable model semantics to the *answer set* semantics and called such a kind of programs *extended logic programs.* The nice features of extended programs are: (1) they include classical negation, $\neg$, in addition to the conventional negation-as-failure, *not*, (2) they allow the CWA to be applied to some specified predicates, and (3) they permit multiple models, and each one is considered as a possible answer. With these powerful capabilities, extended logic programs may be used as a knowledge representation tool to formalize commonsense reasoning.

Kowalski and Sadri [58] may be the pioneers in the attempt to express exceptions in logic programs. They modified the answer set semantics so that it can deal with the default reasoning of Poole's approach [94] by allowing the explicit representation of exceptions in addition to general rules. In this mechanism, general rules are represented as clauses with positive literals in the heads, while exceptions are represented with negative literals in the heads. The reasoning process is performed by assigning exceptions a higher priority than rules, so that if a contradiction arises between a rule and an exception, the exception overrides the rule. Although such a scheme is simple, it suffers from two major limitations: It cannot directly express general rules (respectively, exceptions) with negative (respectively, positive) literals in the heads. Moreover, it does not allow exceptions to exceptions, which often occur in inheritance hierarchies with exceptions.

1.3. Organization of the Dissertation

The major purpose of this dissertation is to investigate how commonsense reasoning can be formalized by extended logic programs. In this investigation, we adopt McCarthy's *simple abnormality formalism* [71] to express exceptions, for it can provide reasonable explanations when conflicting information is recognized. It turns out that our formalization of commonsense knowledge by using extended logic programs does not have the limitations of Kowalski and Sadri's method. In addition, several important problems, such as anomalous extensions, defeasible conclusions, conflicting information, and "unless" semantics, in commonsense reasoning are explored, and solutions to these problems are also proposed.

The remainder of this dissertation is organized as follows. Chapter 2 surveys several major formalizations of nonmonotonic reasoning which include varied forms of closed world reasoning, nonmonotonic logic, autoepistemic logic, default logic, and circumscription. In addition, nonmonotonic inference mechanisms such as the *negation as failure* rule and *program completion* used in logic programming to generate negative information and the syntax of extended logic programs with *answer set* semantics are also reviewed.

In Chapter 3, we investigate how inheritance hierarchies with exceptions can be formalized by using extended logic programs. In this investigation, our formalization can perform not only the conventional *credulous reasoning* but also it can simulate the *ambiguity-blocking inheritance* and *ambiguity-propagating inheritance* in the newly proposed *skeptical reasoning.*

In Chapter 4, three examples in the literature demonstrating the anomalous extension problem are first illustrated. The intuition behind commonsense reasoning is then investigated. Finally, we show how the methods (including our formalization using extended logic programs) proposed to exclude anomalous extensions are all based on this intuition.

In Chapter 5, defeasible conclusions in commonsense reasoning are identified because not every conclusion is retractable. We first identify defeasible conclusions in our representation just based on the simple syntax of extended logic programs. Then, we apply similar approaches to other formalizations of nonmonotonic reasoning to achieve this purpose.

In Chapter 6, we tackle the problem of conflicting information in commonsense reasoning, which most nonmonotonic reasoning formalizations cannot deal with. Based on the idea of *dependency-directed backtracking* in TMSs, we propose the *generalized answer set* semantics to resolve conflicts. Also, similar approaches are applied to other formalizations of nonmonotonic reasoning to handle conflicts.

In Chapter 7, we first point out the semantic inconsistency problem in McCarthy's original simple abnormality formalism and propose a simple way to fix it. We then explore "unless" semantics in commonsense reasoning and propose the *extended answer set* semantics to incorporate this new semantics in the generalized answer set semantics. It turns out that a clause in an extended logic program can be transformed into one of four other forms based on the given information so that the conclusions will become more reasonable and coherent. Also, we endow the other formalizations of nonmonotonic reasoning with the full power of the above "unless" semantics.

In Chapter 8, we first present a new approach to compute stable models of general logic programs. We then propose a method to compute answer sets of extended logic programs based on the computation of stable models. The thesis ends up with conclusions, discussions, and possible future research.

CHAPTER 2

FORMALIZATIONS OF NONMONOTONIC REASONING

As noted in the previous chapter, commonsense reasoning is a kind of plausible reasoning. It is interesting to see that almost all examples demonstrated in AI which call upon such a reasoning have the following common pattern [107]:

Normally, A holds.

Several possible paraphrases of this pattern are "typically, A is the case," "assume A by default," and "in the absence of information to the contrary, assume A." Note that the last paraphrase is based on the notion of logical consistency which motivates several earlier formal theories of nonmonotonic reasoning such as nonmonotonic logic [73, 72] and default logic [106].

In the standard flying birds example, "birds fly" cannot be interpreted as "all birds fly" because there exist exceptions such as ostriches, penguins, dead birds, and so forth. It should be represented by an instance of the above pattern of plausible reasoning as "normally, birds fly," and may be paraphrased as "typically, birds fly," "if X is a bird, then assume by default that X flies," or "if X is a bird and nothing is known that X cannot fly, then assume X flies."

Can the above common pattern of plausible reasoning be formalized by classical logic? As recognized by most AI researchers, the answer is negative. One obvious argument is that plausible inference is not transitive due to the presence of exceptions. For example, cephalopods are molluscs and molluscs normally are shell-bearers, but cephalopods normally are *not* shell-bearers [30, 28].

There are still two popular arguments against the use of classical logic for formalizing commonsense reasoning [107]. Firstly, if we formalize the above flying birds example by explicitly listing all possible exceptions by the axiom

$$bird(X) \wedge \neg ostrich(X) \wedge \neg penguin(X) \wedge \neg dead(X) \wedge \cdots \rightarrow fly(X),$$

we still could not derive $fly(tweety)$ if we only know that Tweety is a bird. This is because we are not given that Tweety is not an exception, such as an ostrich or penguin. Since the antecedent of the implication cannot be derived, there is no way we can conclude the consequent. The second argument against classical logic is due to the monotonicity of classical logic, while commonsense reasoning is nonmonotonic.

Throughout the dissertation, the following naming convention is followed unless otherwise specified. Strings beginning with lowercase letters are used to denote predicate symbols, function symbols, constants, and propositions, while strings beginning with uppercase letters are used to denote predicates, literals, variables, formulas, and so on. In some places, lowercase Greek letters are also used to denote formulas. The standard logic connectives are "$\wedge$" (conjunction), "$\vee$" (disjunction), "$\neg$" (negation), "$\rightarrow$" (implication), and "$\leftrightarrow$" (equivalence), while the quantifiers are "$\exists$" (existential) and "$\forall$" (universal). Sometimes, "$\wedge$" and "," are used interchangeably for convenience. Note that in the convention of logic programming, "$\leftarrow$" is used to denote "if" and has the unidirectional property. In this dissertation, "$\rightarrow$" is used to denote implication of an axiom in first-order logic, while "$\leftarrow$" is used to denote "if" of a clause in logic programming. Formulas, such as axioms or clauses, without explicit quantifications are assumed to be universally quantified. In the following, we briefly survey several major formalizations of nonmonotonic reasoning and nonmonotonic inference rules in logic programming dealing with the negation problem.

2.1. Closed World Reasoning

The basic idea of closed world assumption is to assume a ground atom false if it is not a logical consequence of a theory. This idea was first used to derive negative information in database systems and later was modified to deal with commonsense reasoning.

2.1.1. Closed World Assumption

Reiter [105] proposed the *closed world assumption* (CWA) which is based on the presumption of total knowledge about the domain being represented. This assumption is usually applied to the theory of databases [133]. Suppose we have a student enrollment database. If we cannot find a record, say John enrolls course CS110, then it is reasonable to assume that John does not take the course. It is natural to extend this notion to the case of deductive databases [67, 76, 77, 78]: if a positive fact is not derivable from a given database, then it is assumed false.

Let T be a first-order theory, which contains a set of first-order formulas. Reiter defines the CWA of T by

$$\text{CWA}(T) = T \cup \{\neg A | A \text{ is a ground atom and } T \nvdash A\},$$

where a set alternatively represents a conjunction. In other words, the implicit negative information of T sanctioned by the CWA includes those negative ground literals whose positive counterparts are not derivable from T. Under the CWA, queries are evaluated with respect to (wrt) CWA(T), rather than T itself.

The major advantage of the CWA is that it can save tremendous memory space because the number of negative facts, in general, far exceeds the number of positive ones. However, the problem with the CWA is that the CWA of a theory might be inconsistent. For example, given $T_1 = \{p(a) \vee p(b)\}$, because $T_1 \nvdash p(a)$ and $T_1 \nvdash p(b)$, CWA(T_1) $= \{p(a) \vee p(b),\ \neg p(a),\ \neg p(b)\}$, which is inconsistent. But if theories are restricted to be Horn (i.e., each formula in the theory is a Horn clause, which is a disjunction of literals with at most one positive literal), Reiter has shown that the CWA of a Horn theory T preserves the consistency of T. In this case, the CWA for Horn theories can be regarded as a syntactic counterpart of the semantic definition of the *least Herbrand models* defined by van Emden and Kowalski [128]: Given a Horn theory T, a ground negative literal $\neg A$ in T can be assumed if and only if A is not in the least Herbrand model of T. (Note that the review of closed world reasoning in logic programming, the *negation as failure rule*, is in Subsection 2.6.1.)

2.1.2. Generalized Closed World Assumption

In order to cope with the inconsistency problem under the CWA, Minker [75] proposed the *generalized CWA* (GCWA) for non-Horn theories. Given a non-Horn theory T, the syntactic definition of the GCWA is defined by

$$\begin{aligned}\mathrm{GCWA}(T) = T \cup \{\neg A \mid\, & A \text{ is a ground atom and there is no ground} \\ & \text{positive clause (or null clause) } K \text{ such that} \\ & T \vdash A \vee K \text{ but } T \nvdash K\},\end{aligned}$$

where a ground positive clause is a disjunction of ground atoms. Minker also provided a semantic definition for the GCWA as follows: For a non-Horn theory T, a ground negative literal $\neg A$ in T can be assumed if and only if A is not in any minimal model of T, where a model M of T is *minimal* if there is no proper subset of M which is also a model of T. Consider the above non-Horn theory $T_1 = \{p(a) \vee p(b)\}$. Since the minimal models of T_1 are $\{p(a)\}$ and $\{p(b)\}$, no ground negative literal can be assumed. Thus, the GCWA of T_1 is T_1 itself, which is consistent. Furthermore, Minker has shown that the syntactic and semantic definitions of the GCWA are equivalent. In addition, for the case of Horn theories, the GCWA reduces to the CWA.

2.1.3. Extended Generalized Closed World Assumption

Since both the CWA and the GCWA only assume ground negative literals, Yahya and Henschen [132] extended the GCWA, called the *extended GCWA* (EGCWA), to accept ground negative clauses (i.e., disjunctions of ground negative literals). Note that this approach interprets the disjunction as exclusive or rather than inclusive or. Given a non-Horn theory T, the syntactic definition of the EGCWA is defined by

$$\begin{aligned}\text{EGCWA}(T) = T \cup \{C|C \text{ is a ground negative clause } \neg P_1 \vee \cdots \vee \neg P_n \text{ and}\\ \text{there is no ground positive clause (or null clause) } K\\ \text{such that, for } i = 1, \ldots, n,\ T \vdash P_i \vee K \text{ but } T \nvdash K\}.\end{aligned}$$

Its semantic definition is defined as follows: Given a non-Horn theory T, a ground negative clause $C = \neg P_1 \vee \cdots \vee \neg P_n$ in T can be assumed if and only if C is true in every minimal model of T. Again, consider the above non-Horn theory $T_1 = \{p(a) \vee p(b)\}$, which has two minimal models $\{p(a)\}$ and $\{p(b)\}$. Because the disjunction is interpreted as exclusive or in the EGCWA, $p(a)$ and $p(b)$ cannot both be true simultaneously. Since the ground negative clause $\neg p(a) \vee \neg p(b)$ is true in both minimal models of T_1, it can be assumed by the EGCWA. Yahya and Henschen have also shown that the semantic and syntactic definitions of the EGCWA are equivalent. Furthermore, if the minimal ground negative clauses assumed by the EGCWA are restricted to be unit (i.e., ground negative literals only), the EGCWA reduces to the GCWA.

2.1.4. Careful Closure Procedure

Due to the fact that all the CWA, the GCWA, and the EGCWA apply the closed world reasoning to each predicate in a given theory, none of them can derive any new positive information. Thus, they are rarely used in commonsense reasoning. Borrowing the idea of circumscribing some specified predicate symbols from circumscription [70, 71], Gelfond and Przymusinska [35] proposed the *careful closure procedure* (CCWA), which modifies Minker's GCWA to a subset of predicate symbols of a theory to solve some simple commonsense reasoning problems. Given a first-order theory T, let P and Z be disjointed sets of predicate symbols from T. A ground atom A is called *free for negation* in T if there exists no ground positive clause K containing predicate symbols from P such that $T \vdash A \vee K$ but $T \nvdash K$. The syntactic definition of the CCWA is then defined by

$$\text{CCWA}(T; P; Z) = T \cup \{\neg A | A \text{ is free for negation in } T\}.$$

The corresponding semantic definition of the CCWA is as follows: A ground negative literal $\neg A$ whose predicate symbol is in P can be assumed if A is not in every (P, Z)-minimal model of T (for the definition of (P, Z)-minimal models, see Section 2.5). Now, let us illustrate how the CCWA solves simple commonsense reasoning. Consider the theory $T_2 = \{bird(tweety),\ bird(X) \wedge \neg ab(X) \rightarrow fly(X)\}$ with $P = \{ab\}$ and $Z = \{fly\}$. It is easy to see that $ab(tweety)$ is free for negation in T_2 because there does not exist a ground positive clause K in the definition of CCWA. Thus, $\text{CCWA}(T_2; P; Z) = T_2 \cup \{\neg ab(tweety)\}$. This can derive the conclusion that Tweety can fly, which is what we want to draw from commonsense reasoning. Note that none of the CWA, the GCWA, and the EGCWA can derive such a conclusion. It has been shown by the authors that the syntactic and semantic definitions are equivalent. Furthermore, if P contains all predicate symbols from T, then the CCWA coincides with the GCWA.

2.1.5. Extended Closed World Assumption

Gelfond *et al.* [36, 37] subsequently proposed the *extended CWA* (ECWA) by combining the ideas from the CCWA and the EGCWA. They first refined the notion of free for negation as follows: Given a first-order theory T, let P and Z be sets of predicate symbols defined in the CCWA and Q be the remaining predicate symbols from T. For notational convenience, given a set R of predicate symbols, R^+ (respectively, R^-) is used to denote the set of all ground positive (respectively, negative) literals with predicate symbols from R. Also, a *sentence* denotes a closed formula. Now, an arbitrary sentence S not involving predicate symbols from Z is said *free for negation* in T if there exists no disjunction $K = K_1 \vee \cdots \vee K_n$, where each K_i is in $P^+ \cup Q^+ \cup Q^-$, such that $T \vdash S \vee K$, but $T \nvdash K$. In this definition, the disjunction $S \vee K$ is required to be minimal in T. Then, the syntactic definition of the ECWA is defined by

$$\text{ECWA}(T) = T \cup \{\neg S | S \text{ is free for negation in } T\}.$$

The corresponding semantic meaning of the ECWA is that a sentence S not involving predicate symbols from Z can be assumed false if it is false in every (P, Z)-minimal model of T. It is easy to see that if $Z = Q = \emptyset$, then the ECWA is identical to the EGCWA. Gelfond *et al.* [37] also proved that the ECWA is, in fact, equivalent to circumscription through model-theoretical definition. They further extended the ECWA to the *iterated CWA* (ICWA) for the stratified theories and showed that the ICWA is equivalent to prioritized circumscription. Besides the above various kinds of CWA, Rajasekar *et al.* [99] also proposed a weaker definition of the GCWA, called the *weak GCWA* (WGCWA), for deriving negative information in non-Horn theories.

2.2. Nonmonotonic Logic

McDermott and Doyle's *nonmonotonic logic* (NML) [73] is a consistency-based approach to formalize commonsense knowledge. They modified a standard first-order logic by introducing a modal operator M, whose informal interpretation is "is consistent." The flying birds example can be formalized in their logic by the axiom

$$bird(X) \wedge M\,fly(X) \rightarrow fly(X).$$

This formula can be read informally as "if X is a bird and it is consistent to assert that X can fly, then X flies." Owing to the need of consistency checking, this approach can then have a single general nonmonotonic inference rule whose intuitive content is "$M\phi$ is derivable if $\neg\phi$ is not derivable."

Let A be a nonmonotonic theory. McDermott and Doyle defined a fixed point T of A if

$$T = th(A \cup \{M\omega | \neg\omega \notin T\}),$$

where th denotes closure under first-order logical consequence. They also defined the theorems of T to be the intersection of all its fixed points. The major problem with the NML is that the notion of consistency is too weak. The M operator fails to adequately capture the intuitive concept of consistency. For example, the nonmonotonic theory $\{Mp, \neg p\}$ is consistent.

In response to this defect, McDermott [72] later tried to develop several stronger versions of the logic based on the standard modal logics T, S4, and S5[1]. Unfortunately, these attempts turn out either to be too weak to adequately characterize the M operator in the cases of T and S4 or to collapse the nonmonotonic S5 to ordinary S5 and thus become monotonic. Moore [84] pointed out that the problem with above logics is that they all contain the schema $L\phi \rightarrow \phi$, which means that if an agent believes ϕ, then ϕ is true—but this is not generally true, even for ideally rational agents. Note that the modal operator L is used in Moore's autoepistemic logic.

[1]All modal logics discussed here have two inference rules: ***modus ponens*** $(\phi \wedge (\phi \rightarrow \psi) \rightarrow \psi)$ and ***necessitation*** $(\phi \rightarrow L\phi)$, where $L = \neg M \neg$. They include some axiom schemata from the following list: K: $L(\phi \rightarrow \psi) \rightarrow (L\phi \rightarrow L\psi)$, T: $L\phi \rightarrow \phi$, 4: $L\phi \rightarrow L\,L\phi$, and 5: $\neg L\phi \rightarrow L\neg L\phi$. Note that T is the model logic based on the axiom schemata K and T, S4 is T together with 4, and S5 is S4 together with 5.

2.3. Autoepistemic Logic

In response to the semantic deficiencies of McDermott and Doyle's NML, Moore [84] proposed a reconstruction of their logic based on belief rather than consistency, which he calls *autoepistemic logic* (AEL). Recall that the former logic uses a modal operator M to mean consistency. AEL invokes a dual operator L to mean "is believed," which roughly corresponds to $\neg M \neg$. Moore's AEL is a propositional logic only with the usual formulas formed from a propositional language augmented with the modal operator L.

AEL is intended to model the beliefs of an agent reflecting his own beliefs. The central role in Moore's formalization is played by the notion of a stable autoepistemic expansion of a set of formulas A which intuitively represents a possible set of beliefs of an ideal rational agent. The agent is ideally rational in the sense that he believes in all and only those facts which he can conclude from A and from his other beliefs. If this expansion is unique, then it can be viewed as the set of theorems which follow from A in AEL.

Given some set A of premises, a set T of formulas is a *stable expansion* of A if it satisfies the fixed point condition

$$T = th(A \cup \{L\omega | \omega \in T\} \cup \{\neg L\omega | \omega \notin T\}),$$

where th denotes closure under the entailment relation of propositional logic. Note that under the dual correspondence of L with $\neg M \neg$, Moore's stable expansions differ from McDermott and Doyle's fixed points only by the inclusion of $\{L\omega | \omega \in T\}$ in his fixed point construction. This set provides for an agent's perfect "positive introspection"; if ω is in the agent's belief set, then he believes ω so that $L\omega$ is also in his belief set. The second set in the definition provides for perfect "negative introspection"; if ω is not in an agent's belief set, then the agent does not believe ω so that $\neg L\omega$ is in his belief set.

Levesque [59, 60] subsequently generalized Moore's logic to the first order case. The flying birds example can then be formalized in this logic by the axiom

$$bird(X) \wedge \neg L \neg fly(X) \rightarrow fly(X),$$

which is possibly paraphrased as "if X is a bird, and if you do not believe (know) that X cannot fly, then X flies." As observed by Konolige [54], stable expansions have some undesirable properties. He noted that the theory $\{L\,p \rightarrow p\}$ has two stable expansions. One contains $\neg L\,p$ only, which is intuitively appropriate because there is no ground for entering p into the belief set. The other one contains both $L\,p$ and p, which is intuitively unacceptable. The unwanted one corresponds to an agent arbitrarily entering p, hence also $L\,p$, into his belief set. In order to eliminate this undesired property of Moore's AEL, Konolige [54] proposed the *strongly grounded expansion*, a refinement of the original stable expansion, to achieve it. Another problem with Moore's AEL is that some simple theories do not have stable expansions, which was pointed out by Morris [87]. He then introduced the notion of *stable closure* to fix this difficulty.

Recently, Shvarts [118] argued that McDermott's non-monotonic modal logics [72] may be viewed as autoepistemic logics, and Moore's logic is surely an important one of them. In addition, Shvarts indicated that many problems arising in Moore's logic may be solved with McDermott's logic by an appropriate choice of the underlying modal system. For instance, if S4 is taken to be the underlying system, then the ungrounded extensions found by Konolige disappear, and the additional extensions introduced by Morris take their place.

2.4. Default Logic

The other promising approach to the formalization of nonmonotonic reasoning is Reiter's *default logic* [106], which is also consistency-based. Default logic is very similar to NML except that default reasoning is expressed by rules of inference, called *defaults*, rather than by assertions in the logical language itself. A default is an expression of the form:

$$\frac{\alpha(\boldsymbol{x}) : M\beta_1(\boldsymbol{x}), \ldots, M\beta_m(\boldsymbol{x})}{\omega(\boldsymbol{x})},$$

where $\alpha(\boldsymbol{x})$, $\beta_i(\boldsymbol{x})$, and $\omega(\boldsymbol{x})$ are all formulas whose free variables are among those in $\boldsymbol{x} = x_1, \ldots, x_n$, and "," means "conjunction." $\alpha(\boldsymbol{x})$ is called the *prerequisite* of the default, $\beta_i(\boldsymbol{x})$'s are called its *justifications*, and $\omega(\boldsymbol{x})$ is called its *consequent*. The intuitive meaning of the default is "if $\alpha(\boldsymbol{x})$ holds and each one of $\beta_i(\boldsymbol{x})$ can be consistently assumed, then $\omega(\boldsymbol{x})$ can be derived." If the prerequisite is empty, it may be understood as any tautology. There are two classes of defaults with only a single justification, $\beta(\boldsymbol{x})$. Those with $\beta(\boldsymbol{x}) = \omega(\boldsymbol{x})$ are said to be *normal*, while those with $\beta(\boldsymbol{x}) = \omega(\boldsymbol{x}) \wedge \gamma(\boldsymbol{x})$, for some $\gamma(\boldsymbol{x})$, are called *semi-normal*. Virtually all the defaults occurring in the literature fall into one of these two classes. The flying birds example can be represented by the default

$$\frac{bird(X) : M fly(X)}{fly(X)},$$

which may be read as "if X is a bird, and it can be consistently assumed to fly, then that X flies can be derived."

A default theory is an ordered pair (D, W), where D is a set of defaults and W is a set of sentences. Recall that a sentence is a first-order closed formula. The extensions of a default theory (D, W) are defined by a fixed point construction. For any set S of sentences, define $\Gamma(S)$ to be the smallest set satisfying the following three properties:

(1) $W \subseteq \Gamma(S)$.

(2) $\Gamma(S)$ is closed under first-order logical consequence.

(3) If $(\alpha : M\beta_1, \ldots, M\beta_m)/\omega \in D$, $\alpha \in \Gamma(S)$, and $\neg\beta_i \notin S$ (for $i = 1, \ldots, m$), then $\omega \in \Gamma(S)$.

Then, E is defined to be an *extension* of the default theory (D, W) if and only if $\Gamma(E) = E$; i.e., if and only if E is a fixed point of the operator Γ. A default theory (D, W) may have multiple extensions, and each extension is viewed as a possible set of beliefs for an agent.

Because normal defaults are a very common default pattern, they are broadly applied to formalize commonsense knowledge. Reiter [107] argued that one of the advantages of default logic is that there is a "proof theory" in the case that all default rules are normal. The sense in which normal defaults have a "proof theory" is the following: Given a set W of sentences, a set D of normal defaults, and a sentence β, then β is in some extension of W wrt the defaults D if and only if the "proof theory" sanctions this. The problem with this argument is that a contradictory pair of sentences can be derived from this "proof theory." This is because each sentence belongs to a different extension. For example, let $W_1 = \{p(a) \vee p(b)\}$ and $D_1 = \{: M\neg p(X)/\neg p(X)\}$. The theory (D_1, W_1) has two extensions; one contains $p(a)$ and $\neg p(b)$, and the other one contains $\neg p(a)$ and $p(b)$. By the claimed "proof theory" above, the contradictory pair $p(a)$ and $\neg p(a)$ are derived although each extension is consistent.

One major problem with the default theory is that defaults sometimes interact with one another, but normal defaults cannot adequately constraint these interactions [108]. A typical example, taken from [108], to demonstrate this problem is as follows:

- Typically, adults are employed.

- Typically, high-school dropouts are adults.
- Typically, high-school dropouts are not employed.

This may be expressed by the following normal defaults:

$$\frac{adult(X) : M\, employed(X)}{employed(X)},$$
$$\frac{dropout(X) : M\, adult(X)}{adult(X)},$$
$$\frac{dropout(X) : M \neg employed(X)}{\neg employed(X)}.$$

Given a dropout Tony, the theory will generate two extensions, which differ only in his status of employment because of the interaction of the first and the third defaults. Intuition suggests us to assume that Tony is unemployed, for typical dropouts are not typical adults. This atypicality should block the transitivity from *dropout* through *adult* to *employed*. The first default, however, does not contain explicit information of these exceptional situations which should block its application. One way to overcome this problem is to put these exceptional information in the justification. Thus, the first default above becomes

$$\frac{adult(X) : M\, employed(X) \land \neg dropout(X)}{employed(X)},$$

which is not applicable to known dropouts.

From the above example, it is easy to see how semi-normal defaults can be used to resolve the ambiguities resulting from the interaction between defaults. This approach, however, has three major drawbacks: Firstly, the complexity of theories with semi-normal defaults is substantially worse than that of theories with normal defaults. Secondly, it is possible to over the intersections between defaults so that the resulting theory has no extension. Finally, in a large, complicated system, it is not easy to find all the intersections at the time when new knowledge is given to the system.

2.5. Circumscription

One of the most powerful formalizations of nonmonotonic reasoning is called *circumscription* introduced by McCarthy [70, 71]. This approach is based upon the notion of truth in all minimal models. Assume L is a first-order language. Suppose $P = \{p_1, \ldots, p_m\}$ and $Z = \{z_1, \ldots, z_n\}$ are tuples of distinct predicate symbols of L. The predicate symbols from P are to be circumscribed, and those from Z are allowed to vary when the circumscription process is performed and thus are called *variables*. Let $A(P, Z)$ be a theory which contains predicate symbols from P and Z. The process of circumscription transforms $A(P, Z)$ into a stronger second-order sentence such that it satisfies certain properties. The circumscription of P in $A(P, Z)$ with variable Z is the following sentence CIRC$(A(P, Z); P; Z)$:

$$A(P, Z) \wedge \forall P', Z'[(A(P', Z') \wedge P' \rightarrow P) \rightarrow P' \equiv P], \tag{2.1}$$

where $P = \{p'_1, \ldots, p'_m\}$ and $z = \{z'_1, \ldots, z'_n\}$ are tuples of predicate variables similar to P and Z, and $P' \rightarrow P$ stands for $\wedge_{i=1}^{m} p'_i(X) \rightarrow p_i(X)$. Formula (2.1) states that P has a minimal possible extension under the assumption that $A(P, Z)$ holds and Z is allowed to vary in the process of minimization. The above sentence CIRC$(A(P, Z); P; Z)$ can also be written in the following way

$$A(P, Z) \wedge \neg\exists P', Z'[A(P', Z') \wedge P' < P], \tag{2.2}$$

where $P' < P$ is the abbreviation for $(P' \leq P) \wedge \neg(P \leq P')$ in which $P' \leq P$ denotes $P' \rightarrow P$. The second conjunct in formula (2.2) is called the *circumscription axiom* of $A(P, Z)$. It says that the extensions in $A(P, Z)$ of predicate symbols of P cannot be made smaller even when the extensions of predicate symbols of Z are allowed to vary; or more succinctly, P is minimal in A with Z varying.

Lifschitz [63] further provided the model-theoretic meaning of circumscription to clarify this notion as follows: Let P and Z be defined as above. For any two structures M_1 and M_2, define $M_1 \leq^{P,Z} M_2$ if

(1) M_1 and M_2 have the same domain,

(2) M_1 and M_2 interpret all function symbols and predicate symbols other than those of P and Z identically, and

(3) for each predicate symbol p_i of P, p_i's extension in M_1 is a subset of its extension in M_2.

Thus, $M_1 \leq^{P,Z} M_2$ if M_1 and M_2 differ only in how they interpret the constants in P and Z, and the extension of each p_i in M_1 is a subset of its extension in M_2. Note that the relation $\leq^{P,Z}$ places no restrictions on how M_1 and M_2 interpret the predicates of Z. A structure M is *minimal* wrt $\leq^{P,Z}$ in a class S of structures if $M \in S$ and there is no structure $M' \in S$ such that $M' <^{P,Z} M$, in which $M' <^{P,Z} M$ is defined to be $M' \leq^{P,Z} M$ but not $M \leq^{P,Z} M'$. A model of a theory T is called (P, Z)-*minimal* if it is minimal wrt $\leq^{P,Z}$ in the class of all models of T. Lifschitz [63] and Etherington [27] independently proved that the sentences being true in all (P, Z)-minimal models of $A(P, Z)$ are precisely the sentences entailed by $\text{CIRC}(A(P, Z); P; Z)$.

As illustrated by McCarthy [71] with an example (see Subsection 4.1.1), circumscription seems not adequate to formalize commonsense knowledge because of the existence of undesired minimal models. He then proposed a more powerful version of circumscription, called *prioritized circumscription*, to choose a preferred minimal model by assigning an appropriate priority to each predicate symbol of the theory. Let a tuple P of predicate symbols be partitioned into disjointed parts $P^1, \ldots, P^k$ such that the predicate symbols in P^i have higher priorities than those in P^j for $i < j$. The prioritized circumscription of A with the priorities $P^1 > \cdots > P^k$ and variable Z is denoted by

$$\text{CIRC}(A; P^1 > \cdots > P^k; Z) = \textstyle\bigwedge_{i=1}^{k} \text{CIRC}(A; P^i; P^{i+1}, \ldots, P^k, Z).$$

Although prioritized circumscription is a powerful formalization of commonsense knowledge, one problem with it is: How do we assign an appropriate priority to each predicate symbol before it is performed so that the result will conform to our intuition? The other question with circumscription is that because choosing P and Z differently will result in different conclusion, how do we appropriately choose the correct P and Z so that the conclusion will agree with our intuition? One major drawback of the circumscription is that it is difficult to implement because its definition involves a second-order quantifier. Lifschitz [63] investigated some special cases in which the circumscription reduces to a first-order formula. Subsequently, Przymusinski [96], Ginsberg [40, 41], and Baker and Ginsberg [5] proposed algorithms to compute the more general case of circumscription, each based on the ECWA or the assumption-based TMS (ATMS) [20].

There are various extensions of circumscription. Minker and Perlis [80, 81, 82] developed *protected circumscription* to allow prescription of what objects are or are not to be included in the circumscription process. Lifschitz [64], partly in response to the challenge from Hanks and McDermott [44], developed *pointwise circumscription* to deal with the frame problem. Perlis [91] reformulated the circumscription, called the *autocircumscription*, based on the consistency which is related to self-knowledge and especially to negative introspection. Lifschitz [65] further modified the autocircumscription to include both negative and positive introspection and called it *introspective circumscription*, which is in many ways similar to Moore's AEL. Also, by introducing the concept of supportedness in logic programming into circumscription, You and Li [138] proposed *supported circumscription* to eliminate the anomalous extension of the YSP.

2.6. Logic Programs

Logic programming was introduced in the early 1970s by Kowalski [55] and Colmerauer *et al.* [18], and the first Prolog interpreter was implemented by Roussel in 1972 [112]. It is a direct outgrowth of earlier work in automatic theorem proving and AI. Also, it is heavily founded on the earlier fundamental discovery of the *resolution principle* by Robinson [110] in 1965, for resolution is an inference rule which is particularly well-suited to automation on computers.

Logic programming is based on the idea of declarative programming stemming from Kowalski's [55, 56] principle of separation of logic and control. Ideally, a programmer should be concerned only with the declarative meaning of his or her program, while the procedural aspects of program's execution are handled automatically by systems. Unfortunately, this ideal has not yet been achieved with current logic programming systems. One of the reasons is the lack of clarity as to what should be the proper declarative semantics of logic programs and, in particular, what should be the meaning of negation in logic programming [97].

Initially, logic programming almost deals with definite logic programs (i.e., each clause in the program does not contain negation in its body). van Emden and Kowalski [128] first defined the *least Herbrand model* as the declarative semantics of definite logic programs as follows: Every definite logic program P has exactly one minimal Herbrand model, called the least Herbrand model of P, which is the set of all ground atoms being logical consequences of P. The procedural semantics of definite logic programs was first described by Kowalski [55] and was called SLD-resolution in [4], where SLD-resolution stands for Linear resolution with Selection function for Definite clauses. It was shown that SLD-resolution is sound and complete in [4, 17, 47].

2.6.1. Negation as Failure Rule

Clark [16] subsequently proposed an effective procedure, called the *negation as failure* rule (NF-rule), to deal with negation in normal logic programs[2] (i.e., each clause in the program allows negation in its body). The NF-rule is a practical implementation of closed world assumption in logic programming systems to derive negative information and thus is a nonmonotonic inference rule. It states that if a ground atom A is in the SLD finite failure set of a normal logic program, then $\neg A$ can be assumed. Informally, a ground negative literal $\neg A$ is deduced by the NF-rule if every possible proof of A fails finitely. Since the SLD finite failure set is a subset of the complement of the least Herbrand model, it can be seen that the NF-rule is less powerful than the CWA.

Since Clark's NF-rule is based on SLD-resolution to derive negative information, the SLD-resolution augmented by the NF-rule is called SLDNF-resolution [67]. In SLDNF-resolution, there is no restriction on which positive literal in a goal can be selected for resolution; however, only the ground version of a negative literal can be selected. This condition is called the *safeness* condition on the selection of literals. Basically, the procedure of SLDNF-resolution is as follows: When a positive literal is selected, SLD-resolution is essentially used to derive a new goal. However, when a ground negative literal $\neg A$ is selected, an attempt is made to construct a finitely failed SLDNF-tree with the ground goal clause $\leftarrow A$ as its root. If such a finitely failed tree is constructed, then the subgoal $\neg A$ succeeds. Otherwise, if an SLDNF-refutation is found for $\leftarrow A$, then the subgoal $\neg A$ fails. Note that bindings are only made by successful calls of positive literals. In addition, the NF-rule is purely a test, for negative calls never create bindings; they only succeed or fail.

[2]Normal logic programs are also called general logic programs in other literature.

One problem with SLDNF-resolution procedure is the floundering problem which is defined as follows: Let P be a normal logic program, and G be a normal goal. A computation of $P \cup \{G\}$ *flounders* if at some point in the computation, a goal is reached which contains only non-ground negative literals. In this case, no literal can be selected to satisfy the safeness condition. For example, if G is $\leftarrow \neg p(X)$ and P is any normal program, then the computation of $P \cup \{G\}$ flounders immediately. In response to this difficulty, restrictions on the form of clauses in normal programs are proposed by several authors [19, 67] to avoid floundering. For instance, a clause in a normal program is *range-restricted* if every variable of the clause occurs in some positive literal of its body. This restriction ensures that all negative literals can be fully instantiated before they are selected. The other problem with SLDNF-resolution procedure is the *termination* problem. If a normal program contains recursive rules, it may exist infinite SLDNF-trees, and thus cause the non-termination of tree construction. Thus, finding a mechanism for detecting infinite branches is an important issue in logic programming [93].

2.6.2. Program Completion

Clark [16] also introduced the concept of *program completion* to define a declarative semantics for negation in normal logic programs. In a normal program, the bodies of clauses with predicate symbol p in the head can be viewed as "sufficient" conditions for deriving p from the program. Clark suggested that the bodies of clauses can also be taken as "necessary" conditions with the result that negative information about p can be assumed if all these conditions are not met. He also provided a constructive definition for *completed programs* and used it to prove the soundness and completeness of the NF-rule for definite logic programs.

Suppose that

$$p(T_1,\ldots,T_n) \leftarrow L_1 \wedge \cdots \wedge L_m \tag{2.3}$$

is a clause in a normal program P about predicate symbol (relation) p. We will require a new predicate symbol $=$ not appearing in P whose intended interpretation is the identity relation. Let $X_1,\ldots,X_n$ be variables not appearing in the clause. Then, clause (2.3) is equivalent to

$$p(X_1,\ldots,X_n) \leftarrow (X_1 = T_1) \wedge \cdots \wedge (X_n = T_n) \wedge L_1 \wedge \cdots \wedge L_m.$$

Finally, if $Y_1,\ldots,Y_d$ are the variables of clause (2.3), then the above form is equivalent to

$$\begin{aligned} p(X_1,\ldots,X_n) \leftarrow \exists Y_1 \cdots \exists Y_d((X_1 = T_1) \wedge \cdots \wedge (X_n = T_n) \wedge \\ L_1 \wedge \cdots \wedge L_m). \end{aligned} \tag{2.4}$$

We call this the *general form* of clause (2.3). Suppose there are exactly k clauses, $k > 0$, in P about predicate symbol p. Let

$$\begin{aligned} &p(X_1,\ldots,X_n) \leftarrow E_1 \\ &\cdots \\ &p(X_1,\ldots,X_n) \leftarrow E_k \end{aligned} \tag{2.5}$$

be the k general forms of these clauses. Each of the E_i will be an existentially quantified conjunction of literals as the left-hand side of (2.4). The *completed definition* of p, implicitly given by P, is

$$\forall X_1 \cdots \forall X_n(p(X_1,\ldots,X_n) \leftrightarrow E_1 \vee \cdots \vee E_k).$$

The if-half of this definition is just the k general forms of clauses (2.5) grouped as a single implication. The only-if half is the completion law for p. If there is no clause for p, the completed definition of p, implicitly given by P, is

$$\forall X_1 \cdots \forall X_n(p(X_1,\ldots,X_n) \leftrightarrow false),$$

where $false$ is a contradiction.

Let P be a normal program. The *completion* of P, denoted by comp(P), is the collection of completed definitions of predicate symbols in P together with an appropriate equality theory [16, 67]. Having defined program completion, Clark [16] suggested that negative information not given by a completed program is taken to be false.

One major problem with program completion is that it may cause inconsistency in normal programs. For example, let P contain only one clause $p(a) \leftarrow \neg p(a)$, which should be equivalent to $p(a)$. However, $\text{comp}(P) \equiv \forall X(p(X) \leftrightarrow (X = a) \wedge \neg p(a))$ is not consistent. Furthermore, Clark's program completion is sensitive to the syntax of programs. For instance, both the theories $\{p(a) \leftarrow \neg q(a)\}$ and $\{q(a) \leftarrow \neg p(a)\}$ should be equivalent, but they have different models. The former one has model $\{p(a)\}$, while the latter one has model $\{q(a)\}$. In order to overcome the defects of program completion, several refinements are proposed; these includes weak completion theory [68], generalized predicate completion [122, 52], parallel predicate completion [38], prioritized predicate completion [123], generalized program completion [23], and strong completion of logic programs [22].

2.6.3. Stratified Logic Programs

Because of the above difficulties of negation in logic programming, a lot of research work has been done in attempting to find a more precise semantics for larger classes of logic programs. Apt *et al.* [3] and Van Gelder [129] independently introduced an important class of logic programs (see also earlier work by Chandra and Harel [13]), called *stratified*, whose definition is given below. Let M be a level mapping of a logic program from its set of predicate symbols to the non-negative integers. Let $Sym(L)$ return the predicate symbol of literal L. A logic program P is called *stratified* if there is a level mapping M such that, for every clause

$H \leftarrow L_1 \wedge \cdots \wedge L_n$ in P, $M(Sym(H)) \geq M(Sym(L_i))$ for each positive literal L_i, and $M(Sym(H)) > M(Sym(L_j))$ for each negative literal L_j. The most important feature of stratified programs is that they do not allow recursion through negation and thus form a collection of natural strata (or levels). Evaluation of a literal on an upper stratum will depend on the values of the same and lower strata. Therefore, a unique 'natural' minimal Herbrand model (the canonical model) of a stratified logic program is easily defined by an iterated manner, and such a model (called *iterated fixed point* model) is then argued to represent the declarative semantics of the program.

Beyond stratification, it seems that no canonical model can be defined because there may exist more than one model and it is difficult to determine one among them [33]. To overcome this difficulty, several different semantics for non-stratified logic programs have been proposed. The *stable model* semantics is proposed by Gelfond and Lifschitz [33], which is based on the stable expansions of Moore's AEL. The *default model* semantics is proposed by Bidoit and Froidevaux [8], which is based on the extensions of Reiter's default logic and coincides with stable models. The *weakly perfect model* semantics is proposed by Przymusinska and Przymusinski [95], which is based on McCarthy's circumscription or on Reiter's CWA. The *well-founded model* semantics is proposed by Van Gelder *et al.* [130] and is based on three-valued logic.

2.6.4. Extended Logic Programs

The problem with the traditional logic programming is that total knowledge is presumed because the NF-rule is automatically applied to every negative literal. Therefore, it does not allow one to deal directly with incomplete information. In order to cope with this problem, Gelfond and Lifschitz [34] further extended their stable model semantics to the *answer set* semantics and called this kind of programs *extended logic programs*, which contain the classical negation $\neg$ in addition to the negation-as-failure *not*. In such extended logic programs, negative information is represented explicitly by the classical negation $\neg$. The *answer set semantics* of extended logic programs is defined so that the answer for a ground query A is *yes*, *no*, or *unknown*, depending on whether the answer set contains A, $\neg A$, or neither, respectively. The answer *no* corresponds to the presence of explicit negative information in the program, while the answer *unknown* corresponds to the absence of both explicit positive and explicit negative information. Gelfond and Lifschitz also showed that there is a closed relationship between extended logic programs and Reiter's default logic [106]: the answer sets for an extended logic program coincide with the extensions of the corresponding default theory. With this augmented syntactics and the new semantics, extended logic programs provide more expressive power as a knowledge representation tool to formalize nonmonotonic reasoning.

Since the dissertation will base on extended logic programs to formalize commonsense knowledge, in the following, the syntactics and answer set semantics of these programs are briefly reviewed. An *extended logic program* is a set of formulas of the form

$$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \tag{2.6}$$

where $n \geq m \geq 0$, and each L_i is a literal. Formulas of the above form are also called *clauses* in the context of extended logic programs. A *literal* is a formula of the form A or $\neg A$, where A is an atom. Note that the negation sign in the negative literal $\neg A$ represents the classical negation, but not the negation-as-failure denoted by *not* in (2.6), so that expressions of the form *not* A are not literals according to this definition.

In the semantics of extended logic programs, a clause with variables is treated as standing for the set of all its ground instances. Therefore, it is sufficient to define the notion of answer set for extended logic programs without variables. The definition includes two parts. First, we consider the special case where the extended logic programs without variables do not contain the negation-as-failure *not*. Then, we consider the general case.

Let Π be an extended logic program without variables that does not contain *not*, and let *Lit* be the set of all ground literals in the language of Π. The *answer set* of Π, denoted by $\alpha(\Pi)$, is the smallest subset S of *Lit* such that

(i) for any clause $L_0 \leftarrow L_1, \ldots, L_m$ in Π, if each $L_i \in S$ (for $i = 1, \ldots, m$), then $L_0 \in S$,

(ii) if S contains a pair of complementary literals, then $S = Lit$.

In case (ii), we say that Π is *contradictory*. Consider the program $\Pi_1 = \{Q \leftarrow \neg P,\ \neg P \leftarrow\}$. It has the only answer set $\alpha(\Pi_1) = \{\neg P,\ Q\}$.

Now, let Π be any extended logic program not containing variables. Let *Lit* be defined as before. For any set $S \subseteq Lit$, let Π^S be the extended logic program obtained from Π by deleting

(i) each clause containing an expression *not* L in its body with $L \in S$, and

(ii) all expressions of the form *not* L in the bodies of the remaining clauses.

Clearly, Π^S does not contain *not*, so that its answer set is already defined. If this answer set coincides with S, then we say that S is an *answer set* of Π. That is, S is an answer set of Π if and only if $S = \alpha(\Pi^S)$. For example, consider the program $\Pi_2 = \{\neg Q \leftarrow not\ P\}$. It is easy to see that Π_2 has only one answer set: $\{\neg Q\}$ which is also the answer set of $\Pi_2^{\{\neg Q\}} = \{\neg Q \leftarrow\}$. By the answer set semantics, the answers that the program Π_2 should give to the queries P and Q are, respectively, *unknown* and *no*.

An extended logic program may have more than one answer set. For example, the program $\Pi_3 = \{P \leftarrow not\ Q,\ Q \leftarrow not\ P\}$ has two answer sets: $\{P\}$ which is also the answer set of $\Pi_3^{\{P\}} = \{P \leftarrow\}$, and $\{Q\}$ which is also the answer set of $\Pi_3^{\{Q\}} = \{Q \leftarrow\}$. An extended logic program may have no answer sets. For example, the program $\Pi_4 = \{P \leftarrow \textit{not } P\}$ does not have any answer set. Moreover, an extended logic program may also be contradictory. For example, the program $\Pi_5 = \{P \leftarrow, \neg P \leftarrow\}$ is contradictory.

Note that the answer set semantics treats clauses of the form $Q \leftarrow P$ as *inference rules* "from P derive Q," rather than conditionals. So it inhibits the use of the "contrapositive." That is, the inference "from $\neg Q$ derive $\neg P$" is not allowed.

CHAPTER 3

INHERITANCE HIERARCHIES WITH EXCEPTIONS

Semantic networks [98, 102] are an important tool for knowledge representation. They express knowledge in terms of concepts, their properties, and the hierarchical relationship between concepts. There is an important class of information structures called *inheritance hierarchies* which is a central part of almost all semantic networks and knowledge representation languages [9, 10, 29, 43]. In an inheritance hierarchy, each concept is represented by a node, and the hierarchical relationship between concepts is depicted by connecting appropriate concept nodes via IS-A links[1]. Nodes at the lowest level of an IS-A hierarchy denote individuals, whereas nodes at higher levels denote classes of individuals. Properties are also represented by nodes, and the fact that a property applies to a concept is represented by connecting the property node and the concept node via an appropriately labeled link. In general, a property is attached to the highest concept in an IS-A hierarchy to which the property applies. For inheritance hierarchies without exceptions, a property attached to a node also applies to all its descendants.

Although semantic networks are a powerful notation for representing knowledge, they lack formal semantics as pointed out by Woods [131]. In response to this deficiency, Cercone and Schubert [12, 113] first regarded semantic networks as notational variants of first-order logic. Hayes [46] and Charniak [15] subsequently formalized semantic networks based on first-order logic. This formalization can be viewed as providing the semantics for semantic networks [28]. The translation of semantic

[1]For brevity, IS-A links are used to denote both IS-A and INSTANCE-OF links in semantic networks.

networks into first-order logic invokes mapping individuals to constants, classes to unary predicates, and properties to either unary or binary predicates. For example, an IS-A link between nodes[2] TWEETY and CANARY in an inheritance hierarchy is expressed as *canary*(*tweety*), while the IS-A link between CANARY and BIRD is represented as

$$canary(X) \rightarrow bird(X). \tag{3.1}$$

If a property is mapped to a unary predicate, then property specifications, such as "canaries are yellow," may be expressed as

$$canary(X) \rightarrow yellow(X).$$

In this translation, inheritance is obtained by one or more applications of the inference rules of *universal instantiation* and *modus ponens.* For instance, given *canary*(*tweety*) and axiom (3.1), one may infer *bird*(*tweety*) by a single application of the above two inference rules.

This elegant translation described above works well only in the case of no exceptions existing in the inheritance hierarchies. However, if exceptions are permitted, the translation suffers from an important problem, for first-order logic is monotonic but inheritance hierarchies with exceptions are nonmonotonic. Therefore, research on considering the effects of allowing exceptions in inheritance hierarchies becomes important [29, 30, 28, 124, 125, 26, 27].

The NETL system proposed by Fahlman [29] was one of earlier approaches to deal with exceptions in inheritance hierarchies. However, this method may lead to unintuitive or even invalid conclusions. Later on, Etherington and Reiter [28, 26, 27] proposed that inheritance hierarchies with exceptions could be formalized by default logic. But this approach does not clearly describe how to add exception links to

[2]Nodes in an inheritance hierarchy are represented by strings of small capitals.

their new network representation. Touretzky [124, 125] also presented the notion of *inferential distance ordering* to compute inheritance based on the NETL network representation but without using exception links. Nevertheless, this approach does not completely capture the idea of the given knowledge because only default links are allowed in its network representation. Both the above two approaches (except NETL) may generate multiple conclusions. Because this type of inference generates every possible conclusion, it is called *credulous reasoning* [126]. Inheritance hierarchies with only one conclusion is called *ambiguous*, and *unambiguous* otherwise. Horty *et al.* [50, 51] later presented a new type of inference called *skeptical reasoning* for ambiguous inheritance hierarchies with exceptions. The latter inference type is further divided into two different cases: *ambiguity blocking inheritance* and *ambiguity propagating inheritance*. Both of these two cases also generate intuitive conclusions for different requirement.

Recently, Kowalski and Sadri [58] modified the answer set semantics of extended logic programs to deal with default reasoning of Poole's approach [94]. In their method, general rules are represented as clauses with positive literals in the heads, while exceptions are clauses with negative literals in the heads and having higher priority than general rules. Although the mechanism is simple, it has an important limitation: the syntax is too restricted. That is, general rules with negative literals in the heads and exceptions with positive literals in the heads cannot be expressed directly. Moreover, exceptions to exceptions cannot be expressed either.

In this chapter, we investigate how logic programs can be used to represent inheritance hierarchies with exceptions so that the limitations of the above mechanism can be eliminated. In our approach, we adopt the simple abnormality formalism, which was first proposed by McCarthy [71] for circumscriptive theories, to formalize these inheritance structures. Also, exceptions are assumed to be expressed directly in our approach. It turns out that if the inheritance hierarchies are unambiguous, the abnormality formalism can be directly applied. However, if the inheritance hierarchies are ambiguous, directly applying this formalism will encounter difficulties. In this case, we need to modify it depending on the required inference type. Hereafter, inheritance hierarchies (or networks) are used to denote these hierarchies with exceptions unless otherwise indicated.

3.1. Related Work

NETL proposed by Fahlman [29] is an earlier approach dealing with exceptions in inheritance networks. It adopts shortest path heuristic to compute inheritance. In the NETL network representation, all links are default links and only exception links for IS-NOT-A links are allowed. Unfortunately, this efficient shortest path algorithms may lead to unintuitive or even invalid conclusions as observed in [28, 108, 124].

Subsequently, Etherington and Reiter [26, 27, 28] proposed that default theories could be used as the semantics of inheritance hierarchies in the same spirit as first-order theories for semantic networks. They enhanced the NETL network representation, called E&R network representation, of inheritance hierarchies with five different link types:

(1) strict IS-A links (——▶),

(2) default IS-A links (——>),

(3) strict IS-NOT-A links (++++▶),

(4) default IS-NOT-A links (++++>), and

(5) exception links (- - - - >).

In the enhanced network representation, IS-A and IS-NOT-A links are divided into strict and default links, and exception links which prohibit from invoking some

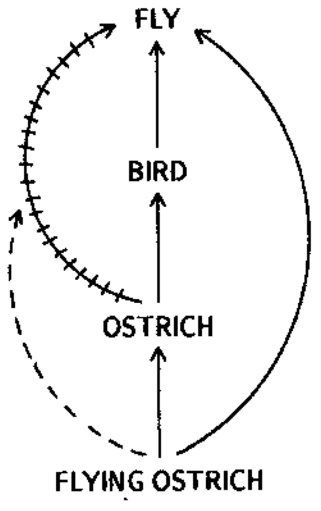

Figure 3.1: NETL network representation

default links are fully expressed, which is contrary to the NETL representation where exception links are only allowed for IS-NOT-A links. Moreover, they also presented a translation of these hierarchies into default logic. The translation of strict links into first-order formulas is the same as that for semantic networks as described above, while a default link corresponds to a default rule with the exception information in its justification if exception links for the default link exist. Consider the following knowledge:

- Normally, birds fly.
- Ostriches are birds but normally do not fly.
- Flying ostriches are ostriches and can fly.

The NETL network representation of the above knowledge is shown in Figure 3.1, whereas the corresponding E&R network representation is shown in Figure 3.2. Note that in the NETL representation, all links are default links and only exception links for IS-NOT-A links are allowed. The corresponding default theory (D_1, W_1) of Figure 3.2 is as follows:

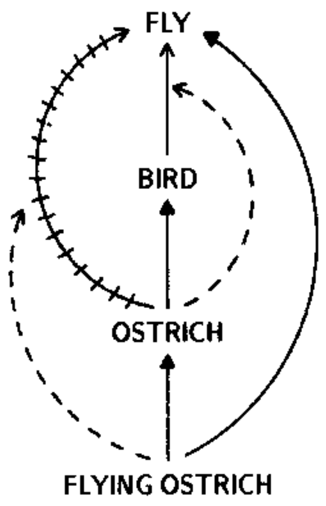

Figure 3.2: E&R's network representation

$$D_1 = \left\{ \frac{bird(X) : M\,fly(X) \wedge \neg ostrich(X)}{fly(X)}, \right.$$
$$\left. \frac{ostrich(X) : M\neg fly(X) \wedge \neg flying\_ostrich(X)}{\neg fly(X)} \right\},$$
$$W_1 = \{flying\_ostrich(X) \rightarrow fly(X),$$
$$ostrich(X) \rightarrow bird(X),$$
$$flying\_ostrich(X) \rightarrow ostrich(X)\}.$$

Clearly, this approach uses semi-normal defaults to represent inheritance networks in default logic. Given a flying ostrich Henry, the above default theory will conclude that Henry is an ostrich, but can fly. Since semi-normal defaults are used, this approach inherits the disadvantages of semi-normal defaults as described in Section 2.4. In addition, this approach does not clearly describe how to add the exception links to the E&R network representation.

Touretzky [124, 125] later presented the notion of *inferential distance ordering* to deal with the exceptions in inheritance hierarchies. He adapted the NETL network representation, but did not use exception links. His approach can be briefly described

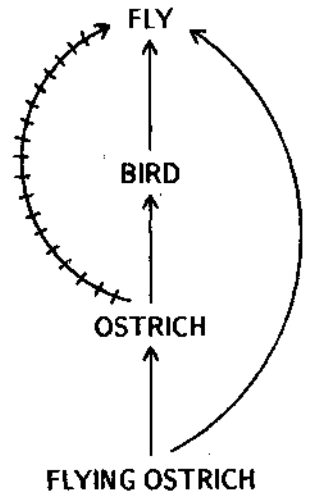

Figure 3.3: Touretzky's network representation

as follows: Given an inheritance hierarchy, if node A inherits property P from node B and property ¬P from node C, then "if A has an inheritance path via B to C and not vice versa, then conclude P; if A has an inheritance path via C to B and not vice versa, then conclude ¬P; otherwise report an ambiguity." [125] For the above example, the corresponding inheritance hierarchy of Touretzky's network representation is depicted in Figure 3.3. Given the flying ostrich Henry, because Henry inherits property FLY from FLYING_OSTRICH and property ¬FLY from OSTRICH, and there is an inheritance path from FLYING_OSTRICH to OSTRICH but not vice versa, we conclude that Henry can fly. The problem with this approach is that the network representation seems not completely capture the given knowledge, for the links in his network representation are all default links [11].

The latter two methods described above both generate multiple extensions from an inheritance hierarchy if the corresponding defaults defeat each other. We will call the inheritance hierarchies with more than one extension *ambiguous*, and *unambiguous* otherwise. Recently, a new type of inference for ambiguous inheritance networks called *skeptical reasoning* was proposed by Horty *et al.* [50, 51] to distinguish it from the previous inference type called *credulous reasoning.* The intuition behind the credulous strategy is that it will believe any of the conflicting arguments and thus may produce an exponential number of extensions from a single inheritance hierarchy. In contrast to the credulous reasoners, skeptical reasoners will not believe any conflicting argument. In other words, they only conclude something which is unambiguous and thus always generate a unique extension. However, there are still two different cases for the skeptical reasoning. Horty *et al.* [50, 51] first attempted to achieve this reasoning by blocking any inference from an ambiguous node, which is referred to as *ambiguity blocking inheritance.* This kind of skeptical reasoning is not equivalent to taking the intersection of all credulous extensions. Touretzky *et al.* [126] mentioned another version of skepticism which will propagate ambiguities from ambiguous nodes. Such version is often referred to as *ambiguity propagating inheritance* and is equivalent to the intersection of all credulous extensions. Stein [120] subsequently proposed an algorithm to compute extensions of this ambiguity propagating inheritance.

Recently, Kowalski and Sadri [58] modified the answer set semantics of extended logic programs to deal with default reasoning of Poole's approach [94] by allowing the explicit representation of exceptions in addition to general rules. In this mechanism, *general rules* are represented as clauses with positive literals in the heads, while *exceptions* are clauses with negative literals in the heads. The reasoning with general rules and exceptions is performed by assigning exceptions to have a higher priority than rules so that if a contradiction arises between a rule and an exception, the exception overrides the rule. To illustrate this, consider the following extended logic program which contains a general rule

$fly(X) \leftarrow bird(X)$

and an exception

$\neg fly(X) \leftarrow ostrich(X).$

This program states that normally birds fly and ostriches do not fly. If Tweety is both an ostrich and a bird, then we conclude that Tweety does not fly because the exception overrides the general rule.

Although the representation of general rules and exceptions in the above mechanism is simple, it suffers from an important limitation: the syntax is too restricted. In other words, general rules with negative literals in the heads and exceptions with positive literals in the heads cannot be expressed directly. For example, the clause

$\neg fly(X) \leftarrow ground\_animal(X)$

cannot be directly used to represent a general rule since the literal in the head is negative. Although $\neg fly(X)$ may be expressed by some other predicate like $nofly(X)$, it loses the expressive merits of the extended logic programs. Moreover, exceptions to exceptions cannot be expressed, and hence, this complicated case cannot be handled correctly. For instance, the clause

$fly(X) \leftarrow flying\_ostrich(X),$

which states that flying ostriches can fly, is an exception to the exception "ostriches do not fly" of the above logic program. If Sam is both an ostrich and a flying ostrich, then by this mechanism, we will make the wrong conclusion that Sam cannot fly because the new exception above is regarded as a general rule.

3.2. Formalizing Exceptions in Inheritance Hierarchies

In general, human knowledge about a specific domain contains at least two parts: *facts* and *general rules*. A fact is certain knowledge, which is universally true and hence contains no exceptions. A fact usually has the form "P is always Q" (or "P is always not Q") which is represented by the strict IS-A link P ——▶ Q (or by the strict IS-NOT-A link P ++++▶ Q) in an inheritance hierarchy[3]. If P is an individual of class Q, then the above strict IS-A and IS-NOT-A links can be expressed, respectively, in extended logic programs by the clauses $q(p) \leftarrow$ and $\neg q(p) \leftarrow$. If P is a subclass of Q, then these two strict IS-A and IS-NOT-A links can be expressed, respectively, by the clauses

$$q(X) \leftarrow p(X)$$

and

$$\neg q(X) \leftarrow p(X).$$

On the other hand, a general rule is an uncertain belief, which contains exceptions. A general rule usually has the form "normally, P is Q" (or "normally, P is not Q") which is represented by the default IS-A link P ——> Q (or by the default IS-NOT-A link P ++++> Q) associated with a unique abnormality label ab_i in the hierarchy. The above two default IS-A and IS-NOT-A links can be expressed, using the abnormality formalism, in extended logic programs by the clauses

$$q(X) \leftarrow p(X) \wedge \mathit{not}\ ab_i(X) \tag{3.2}$$

and

$$\neg q(X) \leftarrow p(X) \wedge \mathit{not}\ ab_i(X), \tag{3.3}$$

respectively. The predicate $ab_i(X)$ means that an object X of class P is abnormal in aspect i, and the negation-as-failure *not* stands for "not derivable (or provable)." Usually, in clause (3.2), an object of class P being abnormal in aspect i means that it has something wrong (abnormal) so that it cannot be in class Q (or cannot have property Q). That is, aspect i is closed related to class (or property) Q. Note that the classical negation $\neg$ in front of abnormality predicates of the original abnormality

[3]Here, we adapt Etherington's five link types [26, 28] to represent inheritance hierarchies for our purpose.

formalism

$$p(X) \wedge \neg ab_i(X) \rightarrow q(X) \tag{3.4}$$

and

$$p(X) \wedge \neg ab_i(X) \rightarrow \neg q(X) \tag{3.5}$$

is changed into the negation-as-failure *not* in our representation. Note also that the corresponding axioms (3.4) and (3.5) in the original abnormality formalism are first-order formulas, whereas the above clauses (3.2) and (3.3) are clauses in extended logic programs which are treated as inference rules as noted in Subsection 2.6.4. As discussed by Kowalski and Sadri in [58], the negation-as-failure *not*, rather than the classical negation $\neg$, is usually interpreted as the negation implicit in the word "unless." This is the reason why the classical negation $\neg$ in front of abnormality predicates of the original abnormality formalism is replaced by the negation-as-failure *not* in the representation of using extended logic programs. We think this modification is more intuitive to capture the meaning of the general rules. Clause (3.2) states that if we cannot prove an object of class P abnormal in aspect i, then we conclude that it is in class Q; or more succinctly, $p(X)$ implies $q(X)$ unless $ab_i(X)$. Similar interpretation can also be applied to clause (3.3). For the convenience of subsequent discussions, if there is no need to distinguish between default and strict links, the IS-A link $\text{P} \rightarrow \text{Q}$ is used to denote either a strict or a default IS-A link from P to Q, and the IS-NOT-A link $\text{P} \not\rightarrow \text{Q}$ is used to denote either a strict or a default IS-NOT-A link from P to Q.

Up to this point, each of the facts and the general rules in one's knowledge about a specific domain is easily expressed by a clause in extended logic programs. We call such a set of clauses a *basic* program. Since the basic program may contain abnormality predicates, $ab_i(X)$, now the question is: How do we know that a certain class is abnormal in some aspects according to the above representation? Ordinarily, if we know that P is a subclass of Q, then P could inherit the properties of Q. However, this is not the case if exceptions are allowed in the networks. The intuition underlying inheritance hierarchies with exceptions is that subclasses are more specific so that they can incorporate information about exceptional cases [124, 125, 126]. That is, subclasses should be allowed to override superclasses. This gives us a clue to find a certain class abnormal in some aspects so that the resulting representation satisfies this intuition. In order to do so, some notion about the knowledge to be formalized needs to be clarified.

Definition 3.1: Knowledge about a specific domain is called *semantically inconsistent* if, intuitively, a pair of contradictory conclusions may be drawn from it, and is called *semantically consistent* otherwise. □

To illustrate this notion, consider the following knowledge about birds:

- Winged birds can fly.
- Birds are normally winged birds.
- Ostriches are birds but normally do not fly.

Since "ostriches normally do not fly" is a more specific information, by the above intuition that subclasses should override superclasses, we may conclude that ostriches do not fly. On the other hand, since ostriches are birds and birds are normally winged birds, and since we cannot infer that ostriches have no wings, by intuition we may also conclude that ostriches are winged birds, which can fly. Here, a pair of contradictory conclusions are drawn: ostriches can fly and also do not fly. Therefore, the above knowledge is semantically inconsistent. If the intuition that subclasses should override superclasses is the dominate inference rule (such as in [125]) to conclude that ostriches do not fly, then we need to retract the other conclusion that ostriches can fly. The only way to retract this conclusion is that ostriches must be abnormal birds wrt having wings because "winged birds can fly" is a fact which is universally true. However, we cannot say that ostriches are abnormal birds wrt having wings, which conflicts with

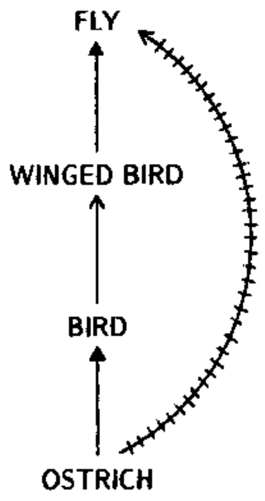

Figure 3.4: Network of the semantically inconsistent knowledge

our intuition because there is no way we can infer that ostriches have no wings from the given knowledge.

For the above semantically inconsistent knowledge, Touretzky's inferential distance ordering approach [124, 125] will conclude that Tweety does not fly if we know that Tweety is an ostrich. This is because all links in his network representation are default links; it does not completely capture the given information. As to the approach of Etherington and Reiter [28], it is not clear how to add the exception links to their network representation for the above example. If no exception links are added, the corresponding network representation is depicted in Figure 3.4 and its default theory (D_2, W_2) is as follows:

$$D_2 = \left\{ \frac{bird(X) : M winged\_bird(X)}{winged\_bird(X)}, \right.$$
$$\left. \frac{ostrich(X) : M \neg fly(X)}{\neg fly(X)} \right\},$$
$$W_2 = \{winged\_bird(X) \rightarrow fly(X),$$
$$ostrich(X) \rightarrow bird(X)\}.$$

Given an ostrich Tweety, the default theory will conclude that Tweety is a winged bird and can fly. Note that the conclusion that Tweety is a winged bird matches our intuition, but the conclusion that Tweety can fly does not.

However, if the fact "winged birds can fly" in the above semantically inconsistent knowledge is changed into the general rule "winged birds normally can fly" or the phrase "winged birds" is replaced by "flying birds," then the revised knowledge becomes semantically consistent. In the former case, we can intuitively say that ostriches are abnormal winged birds wrt flying, for "ostriches normally do not fly" is a more specific information. In the latter case, we can say that ostriches are abnormal birds wrt flying because of the same reason. Here, we say that the class FLYING_BIRD is most related to the property FLY. Motivated by this example, in the following, only semantically consistent knowledge is considered. Before defining a given class being abnormal in some aspects, some prerequisite definitions are given first.

Definition 3.2: Let Γ be an inheritance hierarchy. A path from A to B in Γ is called *positive* if each link in the path is an IS-A link. A path from A to B in Γ is called *negative* if the last link is $C \not\rightarrow B$ and the path from A to C is positive. □

Definition 3.3: Given a path from A to B, a default link is called the *nearest default link* in the path if the default link has the minimum number of links away from B. □

Definition 3.4: Given a path from A to B, if

(i) the last link in the path is a default link, then B is called *most related* to itself,

(ii) the last link in the path is a strict link, then D is called *most related* to B if the nearest default link in the path is $C \rightarrow D$. □

Definition 3.5: A path from A to B in an inheritance hierarchy is called *shortest* if it contains the minimum number of links. □

Definition 3.6: Let Γ be an inheritance hierarchy. Class R is called abnormal in aspect i if the following conditions are both satisfied:

(i) There exists a link R $\not\rightarrow$ Q in Γ.

(ii) There exists a shortest positive path from R to Q such that P $\rightarrow$ S is the nearest default link with label ab_i, and R is a subclass of P. □

Note that S and Q in the above definition are not necessarily distinct. By this definition, since R is a subclass of P and since class R contains exceptional information (i.e., it may derive $\neg q(X)$), R must be abnormal in aspect i so that the default link P $\rightarrow$ S is blocked. In other words, class R does not inherit property Q from its superclasses, which satisfies the intuition that subclasses should override superclasses. Similarly, we have the following symmetric definition.

Definition 3.7: Let Γ be an inheritance hierarchy. Class R is called abnormal in aspect i if the following conditions are both satisfied:

(i) There exists a link R $\rightarrow$ Q in Γ.

(ii) There exists a shortest negative path from R to Q such that P $\rightarrow$ S (or P $\not\rightarrow$ Q if Q = S) is the nearest default link with label ab_i, and R is a subclass of P. □

In the above two definitions, class R being abnormal in aspect i can be expressed in extended logic programs by the clause

$$ab_i(X) \leftarrow r(X),$$

which is represented by the exception link R ----> ab_i in the inheritance hierarchy, where ab_i is the label of the default link from P to S. Such a clause is called an *inheritance cancellation rule* by McCarthy [38, 71]. All the inheritance cancellation rules need to be added to the above basic program so that the resulting representation can satisfy the intuition underlying inheritance networks with exceptions. Note that the shortest paths in the above two definitions are not necessarily required. This condition only provides computational efficiency for finding most related properties

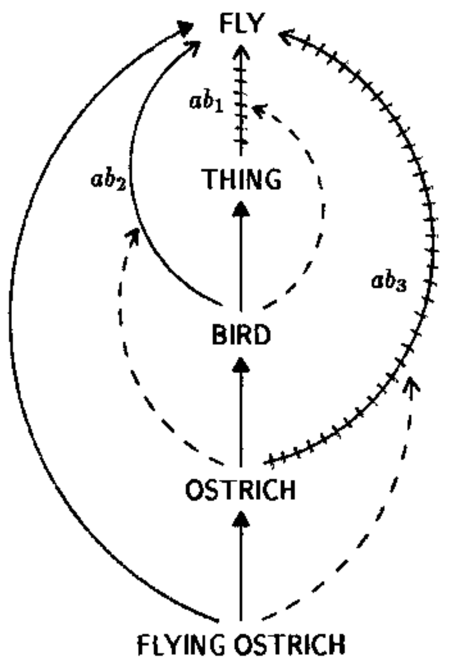

Figure 3.5: Network of flying birds

because the most related nodes must be in these paths so that subclasses can override superclasses due to appropriate cancellation rules.

3.3. Exceptions in Unambiguous Inheritance Hierarchies

We first illustrate the simple case where the inheritance networks with exceptions are unambiguous. Recall that in this case, each network has only one credulous extension. Consider the following commonsense knowledge about birds:

- Things normally do not fly.
- Birds are things but normally fly.
- Ostriches are birds but normally do not fly.
- Flying ostriches are ostriches and can fly.

The corresponding inheritance network is shown in Figure 3.5. Obviously, "birds are things," "ostriches are birds," and "flying ostriches are ostriches and can fly" are facts, which are easily expressed by the following clauses:

1. $thing(X) \leftarrow bird(X)$,
2. $bird(X) \leftarrow ostrich(X)$,
3. $ostrich(X) \leftarrow flying\_ostrich(X)$,
4. $fly(X) \leftarrow flying\_ostrich(X)$.

The general rule "things normally do not fly" is expressed, according to our representation using abnormality formalism, by the clause

5. $\neg fly(X) \leftarrow thing(X) \wedge not\ ab_1(X)$,

which states that things cannot fly unless they are abnormal in aspect 1. Since birds are a subclass of things and since we know that birds normally fly which is a more specific information, obviously, birds are among those objects that are abnormal in aspect 1. More formally, because there exists a link BIRD $\rightarrow$ FLY and because in the shortest negative path from BIRD to FLY the nearest default link is THING $\not\rightarrow$ FLY with abnormality label ab_i, by definition, BIRD is abnormal in aspect i. Therefore, we have the following inheritance cancellation rule

6. $ab_1(X) \leftarrow bird(X)$.

Because the above inheritance cancellation rule declares that birds are abnormal in aspect 1, it blocks the inheritance of the non-flying property of things. Similarly, we have the following clauses expressing the remaining general rules and cancellation rules:

7. $fly(X) \leftarrow bird(X) \wedge not\ ab_2(X)$,
8. $ab_2(X) \leftarrow ostrich(X)$,
9. $\neg fly(X) \leftarrow ostrich(X) \wedge not\ ab_3(X)$,
10. $ab_3(X) \leftarrow flying\_ostrich(X)$.

We call this extended logic program Π_1.

As noted by Genesereth and Nilsson in [38], the important feature of this way of dealing with exceptions is that additional clauses about abnormalities can be added at any time. New knowledge about flying or non-flying objects can be expressed by adding clauses to the logic program instead of by having to change them. For example, if we later learn that penguins are birds but normally do not fly, we just need to add the following new clauses to Π_1:

11. $birds(X) \leftarrow penguin(X)$,

12. $\neg fly(X) \leftarrow penguin(X) \wedge not\ ab_4(X)$,

13. $ab_2(X) \leftarrow penguin(X)$.

Now, if we know that Tweety is an ostrich, we add the clause

14. $ostrich(tweety) \leftarrow$

to Π_1 and obtain the answer set

$\{ostrich(tweety), bird(tweety), thing(tweety), \neg fly(tweety),$
$ab_1(tweety), ab_2(tweety)\}$,

which concludes that Tweety is also a bird and a thing, but it does not fly. On the other hand, if we learn that Sam is a flying ostrich, in the similar way, we can conclude that Sam is also an ostrich, a bird, and a thing, and it can fly. From the above example, we can see that the abnormality formalism can be directly applied in unambiguous inheritance networks.

3.4. Exceptions in Ambiguous Inheritance Hierarchies

Now, we illustrate the general case where the inheritance networks with exceptions are ambiguous. Recall that in this case, each network has more than one credulous extension. Consider the following famous Nixon diamond problem asked by Reiter [71]:

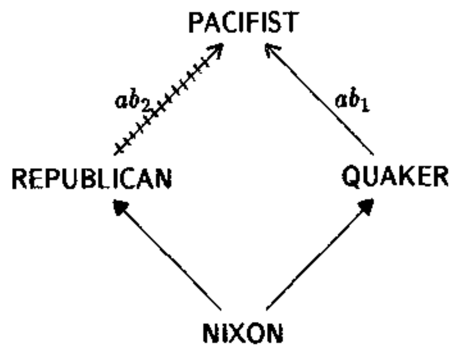

Figure 3.6: The Nixon diamond

- Quakers are normally pacifists.
- Republicans are normally non-pacifists.
- How about Nixon, who is both a Quaker and a Republican?

The corresponding inheritance network is shown in Figure 3.6. These facts and general rules are easily expressed, according to our representation using abnormality formalism, by the following extended logic program Π_2:

1. $pacifist(X) \leftarrow quaker(X) \wedge not\ ab_1(X)$,
2. $\neg pacifist(X) \leftarrow republican(X) \wedge not\ ab_2(X)$,
3. $quaker(nixon) \leftarrow$,
4. $republican(nixon) \leftarrow$.

Because neither Quakers nor Republicans are a subclass of each other, we do not have inheritance cancellation rules in this example.

Since Nixon is both a Quaker and a Republican, how do we know whether or not he is a pacifist? Careless readers might think that this knowledge is semantically inconsistent because there is no way we can infer $ab_1(nixon)$ and $ab_2(nixon)$. Thus, the pair of contradictory literals $pacifist(nixon)$ and $\neg pacifist(nixon)$ are concluded. However, this is not the case. The above knowledge is perfectly semantically consistent because if Nixon is a pacifist, he must be abnormal in aspect 1, or similarly, if Nixon is a non-pacifist, he must be abnormal in aspect 2. Since it has two valid extensions, the corresponding inheritance network is ambiguous. Some earlier inheritance reasoners, such as FRL [109] and NETL [29], cannot identify this ambiguity. So far, two major different intuitive and correct inference types have been proposed to deal with these ambiguous inheritance networks [126]: credulous reasoning and skeptical reasoning. In the following, we elaborate how our representation with a minor modification can be used to simulate these two inference types.

3.4.1. Credulous Reasoning

A *credulous reasoner* [28, 125] tries to draw as many conclusions as possible. It will allow one to conclude that Nixon is either a pacifist or a non-pacifist, but not both. Clearly, the above program Π_2 cannot draw any one of these conclusions. In fact, it has no answer sets at all. For Π_2 to perform credulous reasoning, we can simulate it in the following way: Once we infer that Nixon is a pacifist (respectively, non-pacifist), he must not be a non-pacifist (respectively, pacifist). In other words, if an individual is known to be a pacifist (respectively, non-pacifist), he must be abnormal in the corresponding aspects appeared in the bodies of the general rules which conclude non-pacifists (respectively, pacifists) so that these general rules will be blocked. In doing so, we need to add the clauses

5. $ab_1(X) \leftarrow \neg pacifist(X)$,

6. $ab_2(X) \leftarrow pacifist(X)$,

to Π_2 and obtain the following two answer sets:

$$\{quaker(nixon),\ republican(nixon),\ pacifist(nixon),\ ab_2(nixon)\}$$

and

$$\{quaker(nixon),\ republican(nixon),\ \neg pacifist(nixon),\ ab_1(nixon)\}.$$

Obviously, these two answer sets are the conclusions a credulous reasoner generates. To formalize the credulous reasoning in our representation, we need the following definitions.

Definition 3.8: A path is called a *strict path* if every link in the path is a strict link and is called a *default path* otherwise. □

Definition 3.9: Let Γ be an inheritance hierarchy. A pair of *conflicting paths* from A to B in Γ is defined to be a pair of positive and negative default paths from A to B in Γ. □

Definition 3.10: The *nearest abnormality label* in a path is the abnormality label of the nearest default link in the path. □

Definition 3.11: Let ab_i be the label of a default link A $\rightarrow$ B. Then, A is called the *starting* node of label ab_i. □

Definition 3.12: Let Γ be an inheritance hierarchy. A node B is called *ambiguous* wrt node A if there exists a pair of conflicting paths from A to B in Γ and the length of each path is at least two. □

Now, we can simulate the credulous reasoning by using extended logic programs with defining backward exception links as follows:

Definition 3.13: Let Γ be an inheritance hierarchy. Suppose node B is ambiguous wrt node A. Let ab_i and ab_j be the nearest abnormality labels of the negative and positive paths from A to B, respectively. Also, let C and D be the starting nodes of labels ab_i and ab_j, respectively. Then, the *positive backward exception link* A - - - -> ab_i and the *negative backward exception link* A + + + +> ab_j are added to Γ, which correspond, respectively, to the clauses

$$ab_i(X) \leftarrow a(X) \wedge c(X)$$

and

$$ab_j(X) \leftarrow \neg a(X) \wedge d(X). \ \square$$

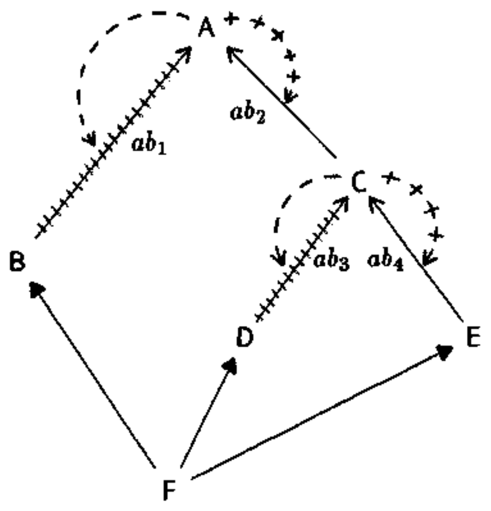

Figure 3.7: Nested Nixon diamond in credulous reasoning

The above formulation of adding backward exception links is exactly in the same spirit of the interaction of defaults in default logic. Consider Figure 3.7 which is a nested Nixon diamond. Node A is ambiguous wrt to node F because there exist a positive default path from F to A via E and C and a negative default path from F to A via B. Similarly, node C is ambiguous wrt node F because there exist a positive default path from F to C via E and a negative default path from F to C via D. Therefore, the positive backward exception links A -----> ab_1 and C -----> ab_3 and the negative backward exception links A ++++> ab_2 and C ++++> ab_4 need to be added to the network. The corresponding extended logic program Π_3 of Figure 3.7 is as follows:

1. $b(f)$,
2. $d(f)$,
3. $e(f)$,
4. $\neg a(X) \leftarrow b(X) \wedge not\ ab_1(X)$,
5. $a(X) \leftarrow c(X) \wedge not\ ab_2(X)$,
6. $\neg c(X) \leftarrow d(X) \wedge not\ ab_3(X)$,

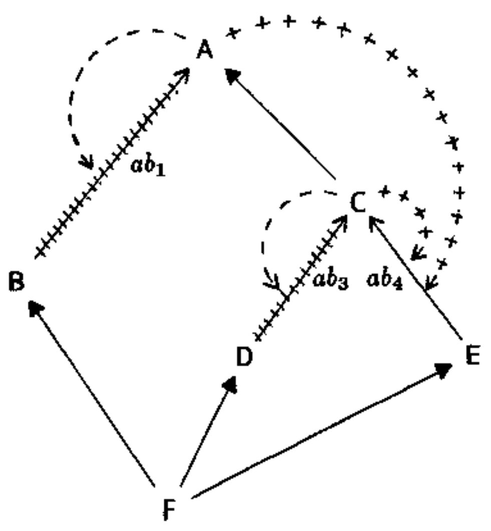

Figure 3.8: Nested Nixon diamond in credulous reasoning

7. $c(X) \leftarrow e(X) \wedge not\ ab_4(X)$,

8. $ab_1(X) \leftarrow a(X) \wedge b(X)$,

9. $ab_2(X) \leftarrow \neg a(X) \wedge c(X)$,

10. $ab_3(X) \leftarrow c(X) \wedge d(X)$,

11. $ab_4(X) \leftarrow \neg c(X) \wedge e(X)$.

The above extended logic program Π_3 has three answer sets:

$S_1 = \{b(f),\ d(f),\ e(f),\ \neg a(f),\ \neg c(f),\ ab_4(f)\}$,

$S_2 = \{b(f),\ d(f),\ e(f),\ c(f),\ \neg a(f),\ ab_2(f),\ ab_3(f)\}$,

$S_3 = \{b(f),\ d(f),\ e(f),\ a(f),\ c(f),\ ab_1(f),\ ab_3(f)\}$,

which agree with the extensions produced by the corresponding default theory.

If the default link C $\rightarrow$ A is changed into a strict link in the above network, the modified inheritance hierarchy is shown in Figure 3.8. In this case, the original negative backward exception link A + + + + > ab_2 becomes the new backward exception link A + + + + > ab_4 because ab_4 is the nearest abnormality label of the positive path from F to A. In this network, there are only two answer sets S_1 and S_3.

3.4.2. Skeptical Reasoning

A *skeptical reasoner* [50] refuses to draw conclusions in ambiguous situations. For the above Nixon diamond example, it will offer no opinion as to whether Nixon is or is not a pacifist. The above program Π_2 also cannot obtain this conclusion. For the case of ambiguity-propagating inheritance, since its extension is equivalent to taking the intersection of all credulous extensions, we can simply simulate this version of skeptical reasoning by taking the intersection of all answer sets obtained by using credulous reasoning. For the case of ambiguity-blocking inheritance, however, its extension is not equivalent to taking the intersection of all credulous extensions. We can simulate this version of reasoning in the following way for the Nixon diamond example. Since Quakers are normally pacifists, they must block other general rules which conclude non-pacifists. In other words, Quakers must be abnormal in the corresponding aspects appeared in the bodies of the general rules which conclude non-pacifists. Similarly, since Republicans are normally non-pacifists, they must block other general rules which conclude pacifists. That is, Republicans must be abnormal in the corresponding aspects appeared in the bodies of the general rules which conclude pacifists. In doing so, the clauses

7. $ab_1(X) \leftarrow republican(X)$,

8. $ab_2(X) \leftarrow quaker(X)$,

need to be added to Π_2. Clearly, in this case, there is only one answer set

$$\{quaker(nixon),\ republican(nixon),\ ab_1(nixon),\ ab_2(nixon)\}$$

which offers no opinion about Nixon's pacifism. To formalize the ambiguity-blocking inheritance in our representation, we need the following definitions:

Definition 3.14: Given a path from A to B whose first link is a strict link, a strict subpath from A to C of the given path is called the *longest* if the subpath contains the maximum number of links. □

Definition 3.15: Given a path from A to B which contains ambiguous nodes, the one with the minimum number of links away from B is called the *nearest ambiguous node* of the path. □

Definition 3.16: Given a path from A to B, a node C is called the *nearest certain node* of the path if one of the following conditions is satisfied.

(i) Node C is the nearest ambiguous node of the path when there exist ambiguous nodes in the path.

(ii) The path from A to C is the longest strict subpath of the given path when there is no ambiguous node in the path. □

Now, we simulate the ambiguity-blocking inheritance by adding forward exception links as follows:

Definition 3.17: Let Γ be an inheritance hierarchy. Suppose node B is ambiguous wrt node A. Let ab_i and C be the nearest abnormality label and the nearest certain node in the positive path from B to A, respectively. Also, let ab_j and D be the nearest abnormality label and the nearest certain node in the negative path from B to A, respectively. Then, the following two *forward exception links* are added to Γ: the exception links C -----> ab_j and D -----> ab_i, which are represented, respectively, by the clauses

$$ab_j(X) \leftarrow c(X) \wedge e(X)$$

and

$$ab_i(X) \leftarrow d(X) \wedge f(X),$$

where E and F are the starting nodes of labels ab_j and ab_i, respectively. □

Consider Figure 3.9 which is basically the same as Figure 3.7, but for skeptical reasoning. In this figure, since there exists a pair of conflicting paths from F to A, the forward exception links C -----> ab_1 and B -----> ab_2 need to be added to the inheritance hierarchy. Similarly, for the pair of conflicting paths from F to C,

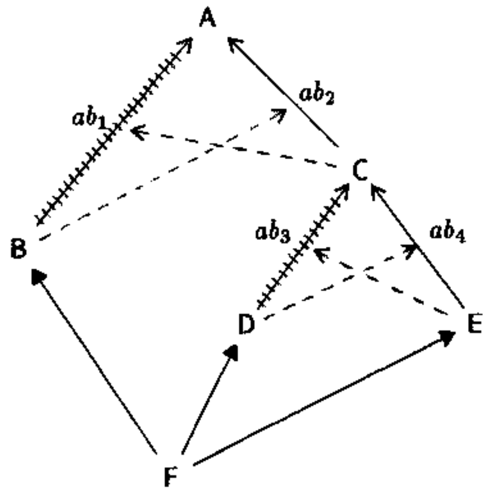

Figure 3.9: Nested Nixon diamond in skeptical reasoning

the forward exception links E -----> ab_3 and D -----> ab_4 need to be added to the network. The corresponding extended logic program Π_4 of Figure 3.9 is as follows:

1. $b(f)$,
2. $d(f)$,
3. $e(f)$,
4. $\neg a(X) \leftarrow b(X) \wedge not\ ab_1(X)$,
5. $a(X) \leftarrow c(X) \wedge not\ ab_2(X)$,
6. $\neg c(X) \leftarrow d(X) \wedge not\ ab_3(X)$,
7. $c(X) \leftarrow e(X) \wedge not\ ab_4(X)$,
8. $ab_1(X) \leftarrow c(X) \wedge b(X)$,
9. $ab_2(X) \leftarrow b(X) \wedge c(X)$,
10. $ab_3(X) \leftarrow e(X) \wedge d(X)$,
11. $ab_4(X) \leftarrow d(X) \wedge e(X)$.

Since node C is ambiguous, it is blocked. Thus, the negative link B $\not\rightarrow$ A is not blocked, and we can conclude $\neg a(f)$. Clearly, there is only one answer set of the

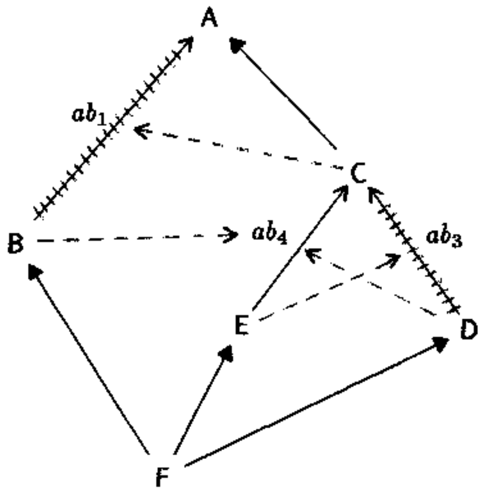

Figure 3.10: Nested Nixon diamond in skeptical reasoning

above program Π_4:

$$\{b(f),\ d(f),\ e(f),\ \neg a(f)\},$$

which is not equivalent to the intersection of the credulous extensions.

Consider the above figure with the default link C $\rightarrow$ A replaced by a strict link. In this case, the modified inheritance hierarchy is shown in Figure 3.10. Now, node C is the nearest ambiguous node, and ab_4 is the nearest abnormality label in the positive path from F to A. Here, the original forward exception link B $\dashrightarrow$ ab_2 is replaced by the new exception link B $\dashrightarrow$ ab_4. As we can see, the negative link B $\not\rightarrow$ A is still not blocked by the new exception link B $\dashrightarrow$ ab_4, and thus, we have the same answer set as before.

3.5. Redundant Statements in Inheritance Hierarchies

As noted by Touretzky in [124], true but redundant statements may cause problems in some earlier inheritance reasoners, such as FRL [109] and NETL [29]. Consider the following commonsense knowledge in [124]:

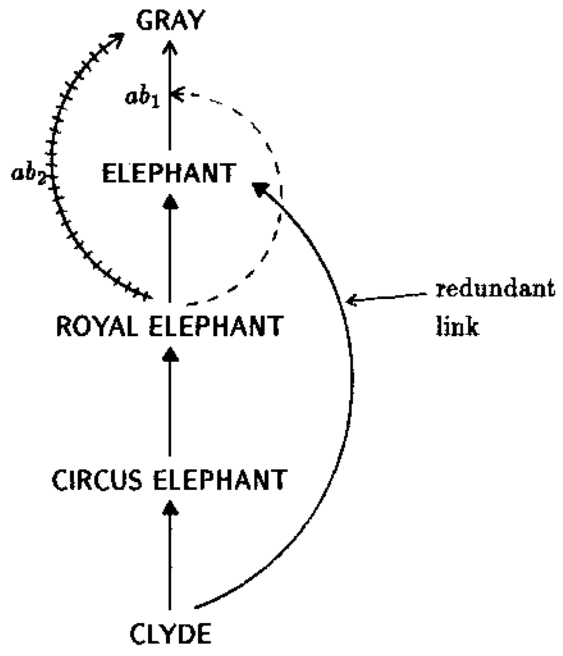

Figure 3.11: Redundant links in the network

- Elephants are typically gray.
- Royal elephants are elephants but are typically not gray.
- Circus elephants are royal elephants.
- Clyde is a circus elephant.

The corresponding inheritance network is shown in Figure 3.11. By the intuition underlying all inheritance systems that subclasses should override superclasses, Clyde is not gray. Clyde being an elephant is a redundant statement because it can be derived from the given information. If we add this fact explicitly to the above knowledge, it causes problems in some inheritance reasoners. In FRL, Clyde will inherit properties through both circus elephants and elephants. Thus, FRL will conclude that Clyde is and also is not gray without identifying the contradiction. In NETL, the redundant statement that Clyde is an elephant establishes the shortest inference path to gray, which is shorter than the other two paths (one to gray and the other one to non-gray) which go through circus elephant. NETL will, therefore, draw the intuitively wrong conclusion that Clyde is gray.

It is easy to see that our approach is unaffected by such true but redundant statements. By our representation, we have the following extended logic program Π_5:

1. $gray(X) \leftarrow elephant(X) \wedge not\ ab_1(X)$,
2. $elephant(X) \leftarrow royal\_elephant(X)$,
3. $\neg gray(X) \leftarrow royal\_elephant(X) \wedge not\ ab_2(X)$,
4. $ab_1(X) \leftarrow royal\_elephant(X)$,
5. $royal\_elephant(X) \leftarrow circus\_elephant(X)$,
6. $circus\_elephant(clyde) \leftarrow$.

If we add the redundant statement

7. $elephant(clyde) \leftarrow$

to Π_5, we still conclude that Clyde is not gray. This is because Clyde is both a circus elephant and a royal elephant which implies that Clyde is abnormal in aspect 1 and block the general rule to conclude gray.

CHAPTER 4

THE ANOMALOUS EXTENSION PROBLEM

The simple abnormality formalism was originally proposed by McCarthy [71] primarily to express commonsense knowledge in circumscriptive theories. Unfortunately, this formalism seems not adequate to fully achieve such a purpose as first illustrated by McCarthy himself in [71], for there may exist undesired, unintutive minimal models, which are called *anomalous extensions*[1]. Later on, Hanks and McDermott in their celebrated paper [44, 45], the best paper at AAAI-86 [1], carefully examined McCarthy's proposed solution to the frame problem using such a formalism, and then, they discovered that there exists an anomalous extension in the famous Yale shooting problem (YSP) from temporal default reasoning when it is formalized in three major existing non-monotonic reasoning mechanisms (i.e., Reiter's default logic [106], McCarthy's circumscription [71], and McDermott-Doyle's non-monotonic logic [73]). Subsequently, Morris also in his celebrated paper [85, 86], the most outstanding paper at AAAI-87 [2], demonstrated that the anomalous extension problem not only presents in temporal reasoning, but also occurs in non-temporal reasoning by presenting an example from taxonomic reasoning.

Several solutions have been proposed to cope with this anomalous extension problem. However, some of them [117] only deal with temporal reasoning. In this chapter, we are only interested in the solutions to more general cases which also include non-temporal reasoning. Morris [86] first proposed that when the YSP is

[1]As noted in [45], because of the parallel properties between extensions in default logic and minimal models in circumscription, both terms will be used interchangeably.

reformulated in terms of a truth maintenance system (TMS) [21], the anomalous extension noted by Hanks and McDermott disappears. He also showed that by mimicking the TMS formulation, non-normal defaults in default logic can be used to exclude the anomalous extensions. Similar to the non-normal representation, Gelfond [32] also presented a reformulation of the YSP based on autoepistemic logic to avoid the unintuitive expansion. Recently, You and Li [138] also present the *supported circumscription* based on the notion of supportedness in logic programming to achieve the same purpose of excluding anomalous extensions.

In this chapter, we first illustrate the three aforementioned examples with anomalous extensions and then show that if they are formulated in extended logic programs as presented in Chapter 3, the undesired conclusions are excluded. We then investigate the intuition behind commonsense reasoning; especially, when there are more than one conclusion, how do we choose a preferred one which corresponds with our intuition? Then, we review each of the solution mentioned before to see how it can exclude unwanted conclusions. Finally, we give a brief discussion on the other approach to the famous YSP.

4.1. Examples with Anomalous Extensions

4.1.1. McCarthy Anomaly

McCarthy [71] first recognized that the abnormality formalism is not adequate to formalize commonsense reasoning in circumscriptive theories. He illustrated this point with the following first-order theory [71]:

1. $\neg ab_1(X) \rightarrow \neg fly(X)$,
2. $bird(X) \rightarrow ab_1(X)$,
3. $bird(X) \wedge \neg ab_2(X) \rightarrow fly(X)$,

4. $canary(X) \wedge \neg ab_3(X) \rightarrow bird(X)$,
5. $canary(tweety)$.

Since we know that Tweety is a canary, can we conclude that Tweety can fly? As noted by McCarthy, simply circumscribing the above axioms over ab_1, ab_2, and ab_3, varying fly, leaves this question undecided, because Tweety can either be abnormal in aspect 1 or in aspect 3. That is, there are two minimal models: one concludes that Tweety can fly, and the other does not. However, commonsense tells us that we should conclude that Tweety can fly. In other words, the conclusion that Tweety does not fly is counter-intuitive, which is an anomalous extension. In order to achieve this intuition, we may prefer to have Tweety abnormal in aspect 1 to having Tweety abnormal in aspect 3. This observation caused McCarthy to introduce a more powerful version of circumscription, called *prioritized circumscription*, to choose a preferred minimal model by assigning an appropriate priority to each predicate symbol of the theories.

Surprisingly, when the above theory is formalized by our representation using extended logic programs, it has only one answer set which corresponds with our intuition; the anomalous minimal model disappears. In our representation, the classical negation $\neg$ in the front of each abnormal predicate, $ab_i(X)$, is simply replaced by the negation-as-failure *not* and the following extended logic program Π_1 is obtained:

1. $\neg fly(X) \leftarrow not\ ab_1(X)$,
2. $ab_1(X) \leftarrow bird(X)$,
3. $fly(X) \leftarrow bird(X) \wedge not\ ab_2(X)$,
4. $bird(X) \leftarrow canary(X) \wedge not\ ab_3(X)$,
5. $canary(tweety) \leftarrow$.

By the answer set semantics, program Π_1 has only one answer set

$$\{canary(tweety),\ bird(tweety),\ fly(tweety),\ ab_1(tweety)\},$$

which correctly answers the question that Tweety can fly. The main reason why our representation has only one answer set is because the extended logic program above becomes stratified after renaming each negative literal to be a positive literal with the same arity (see Section 8.1 for more details), predicate symbols in lower stratum have higher priorities as discussed in [37]. Thus, the priority assigned to ab_3 is higher than that to ab_1, which corresponds to the prioritized circumscription.

4.1.2. Hanks-McDermott Anomaly

Hanks and McDermott [44] presented the YSP in temporal reasoning to demonstrate the anomalous extension problem as follows:

- A person is alive at situation s_0.
- A gun becomes loaded any time a *load* event happens.
- The person becomes dead any time he is shot with a loaded gun, and

  being shot with a loaded gun is abnormal with respect to staying alive.

When the example is formulated by the situation calculus and the abnormality formalism, the following situation-calculus abnormality theory is obtained [45]:

1. $t(alive, s_0)$,
2. $t(loaded, result(load, S))$,
3. $t(loaded, S) \rightarrow t(dead, result(shoot, S)) \land ab(alive, shoot, S)$,
4. $t(F, S) \land \neg ab(F, E, S) \rightarrow t(F, result(E, S))$,

where $t(F, S)$ represents the assertion that fact F is true in state S, $result(E, S)$ represents the situation resulting from applying event E to the situation S, and $ab(F, E, S)$ represents the assertion that fact F is abnormal wrt event E occurring in state S. Note that the last axiom is a *frame axiom*, which states that fact F will remain true in a new situation after event E happens unless F is abnormal in the new state. Consider the following situations:

s_0 — the initial state,

$s_1 = result(load, s_0)$ — the state resulting from loading a gun in state s_0,

$s_2 = result(wait, s_1)$ — the state resulting from waiting for a while in state s_1,

$s_3 = result(shoot, s_2)$ — the state resulting from shooting the gun in state s_2.

The question is whether the person is alive or dead at situation s_3. Similarly, simply circumscribing ab, varying t, leaves this question undecided because there are two minimal models: One concludes that the person is dead when he was shot, which agrees with our intuition. The other one concludes that the gun mysteriously becomes unloaded during the waiting period, so the person is still alive after shooting, which is counter-intuitive.

When the shooting example is formulated by our representation, the following extended logic program Π_2 is obtained:

1. $t(alive, s_0) \leftarrow,$
2. $t(loaded, s_1) \leftarrow,$
3. $t(dead, s_3) \leftarrow t(loaded, s_2),$
4. $ab(alive, shoot, s_2) \leftarrow t(loaded, s_2),$
5. $t(alive, s_1) \leftarrow t(alive, s_0) \wedge not\ ab(alive, load, s_0),$
6. $t(alive, s_2) \leftarrow t(alive, s_1) \wedge not\ ab(alive, wait, s_1),$
7. $t(alive, s_3) \leftarrow t(alive, s_2) \wedge not\ ab(alive, shoot, s_2),$
8. $t(loaded, s_2) \leftarrow t(loaded, s_1) \wedge not\ ab(loaded, wait, s_1),$
9. $t(loaded, s_3) \leftarrow t(loaded, s_2) \wedge not\ ab(loaded, shoot, s_2).$

By the answer set semantics, program Π_2 has only one answer set:

$$\{t(alive, s_0),\ t(alive, s_1),\ t(alive, s_2),\ t(loaded, s_1),\ t(loaded, s_2),$$
$$t(loaded, s_3),\ t(dead, s_3),\ ab(alive, shoot, s_2)\},$$

which correctly conclude that the person is dead at state s_3. Note that the above logic program is also stratified.

4.1.3. Morris Anomaly

Morris also presented an example in taxonomic reasoning which is structurally very similar to the YSP and also reproduces the same difficulty as follows:

- Animals normally cannot fly.
- Winged animals are exceptions to this; they can fly.
- All birds are animals.
- Birds normally have wings.
- Tweety is a bird.

To simplify the formulation of the example, he made the assignments

$animal$ = "Tweety is an animal,"

$bird$ = "Tweety is a bird,"

$wing$ = "Tweety has wings,"

fly = "Tweety can fly,"

abA = "Tweety is an abnormal animal with respect to not flying,"

abB = "Tweety is an abnormal bird with respect to having wings."

By following the abnormality formalism, the example above is formalized by the propositional theory:

1. $animal \wedge \neg abA \rightarrow \neg fly$,
2. $wing \rightarrow abA$,
3. $wing \rightarrow fly$,
4. $bird \rightarrow animal$,
5. $bird \wedge \neg abB \rightarrow wing$,
6. $bird$.

Intuitively, we may conclude that Tweety can fly because we only know that it is a bird. However, as before, simply circumscribing abA and abB, varying fly, leaves this intuition not confirmed because there are two minimal models: One concludes that

Tweety can fly, which matches our intuition. The other one concludes that Tweety cannot fly because it has no wings, which is counter-intuitive.

Similarly, when the example is formulated by our representation, the following extended logic program Π_3 is obtained:

1. $\neg fly \leftarrow animal \wedge not\ abA$,
2. $abA \leftarrow wing$,
3. $fly \leftarrow wing$,
4. $animal \leftarrow bird$,
5. $wing \leftarrow bird \wedge not\ abB$,
6. $bird \leftarrow$.

Clearly, program Π_3 has only one answer set

$\{bird, wing, fly, animal, abA\}$,

which agrees with our intuition that Tweety can fly. Note that after renaming each negative literal, the above program also becomes stratified.

4.2. Investigating Intuition Behind Commonsense Reasoning

Recall that commonsense reasoning is usually trying to derive plausible conclusions from incomplete information by making some reasonable default assumptions. In the standard flying bird example, "birds normally fly" can be represented by the first-order axiom

$bird(X) \wedge \neg ab_1(X) \rightarrow fly(X)$

with $P = \{ab_1\}$ and $Z = \{fly\}$ in circumscriptive formalization. Now, suppose we only know that Tweety is a bird. Since we have no idea about whether Tweety is abnormal wrt flying, we will naturally make the reasonable assumption that Tweety is a normal bird. Hence, we can conclude that Tweety can fly by default. The basic idea in commonsense reasoning is that when a ground predicate A is not derivable, then A is assumed false. In general, this idea works well in simple cases such as the above flying bird example. However, if there exists a case when newly made assumptions are derived from previously made assumptions, then some undesired, unintuitive conclusions may be generated. Such situation may cause the anomalous extension problem, which is illustrated by the following example.

Consider the commonsense knowledge:

- If professor John has students, he normally meets them every Friday.
- In the meeting, he normally likes to have a cup of coffee.

The above knowledge can be formalized by the propositional theory T_1

1. $has\_students \wedge \neg ab1 \rightarrow meeting$,
2. $meeting \wedge \neg ab2 \rightarrow coffee.$

Suppose we only know that professor John does have some students working with him. Then, we add this fact

3. $has\_students$

to the above theory T_1. Now, we can conclude that professor John will meet his students next Friday (*meeting*) and will have a cup of coffee at the meeting (*coffee*), for we have no idea about anything abnormal wrt attending meeting or having coffee of professor John. Indeed, this is the conclusion drawn from the circumscription of T_1 with $P = \{ab1, ab2\}$ and $Z = \{meeting, coffee\}$. Note that both *meeting* and *coffee* in the conclusion are the consequents of axioms, and the assumptions made, $\neg ab1$ and $\neg ab2$, are both in the antecedents of axioms.

Suppose we later learn more information about professor John:

- If he is in meeting, then he is abnormal wrt being in his office.
- If he is not abnormal wrt being in his office, then he will stay in his office.

This can be expressed by the axioms:

4. $meeting \rightarrow ab3$,

5. $\neg ab3 \rightarrow office$.

Since we have already concluded that professor John will meet his students on next Friday, we may further conclude that he will be abnormal wrt being in his office during the meeting time ($ab3$). This conclusion agrees well with our intuition. Let T_2 be the theory containing the above five axioms. However, when we circumscribe T_2 with $P = \{ab1, ab2, ab3\}$ and $Z = \{meeting, coffee, office\}$, two minimal models are obtained: One is $M_1 = \{has\_students, meeting, coffee, ab3\}$, which is the intuitive one. The other one is $M_2 = \{has\_students, ab1, office\}$, which is an undesired conclusion stating that professor John will be in his office and will be abnormal wrt attending his students' meeting. In the following, we carefully examine how these two minimal models are generated and how one of them matches our intuition while the other one does not.

Firstly, we look at how the unintuitive model M_2 is obtained. Since we cannot derive that professor John is abnormal wrt being in his office ($ab3$) from T_2, the assumption that he is not abnormal wrt being in his office at the meeting time ($\neg ab3$) can be assumed. With this assumption, we can derive that he is in his office ($office$) by axiom 5 and he is not in the meeting ($\neg meeting$) by the contraposition of axiom 4. Furthermore, by the contraposition of axiom 1, we can derive that he is abnormal wrt being in the meeting ($ab1$). Note that this reasoning pattern is a kind of backward reasoning because it starts from the assumptions in the highest level and performs reasoning backward by using the contrapositives of axioms (i.e., in the reverse direction of implications of axioms). Such kind of backward reasoning is not an intuitive reasoning mechanism human being possesses; it is not natural for people to reason something backwards in incomplete information.

Now, we examine how the intuitive one M_1 is generated. Since we do not know that professor John will be abnormal wrt being in next Friday meeting ($ab1$), we can assume that he will be normally present in the meeting ($\neg ab1$). Thus, he will attend the meeting with his students (*meeting*) on next Friday by axiom 1. Similarly, by axiom 2, we can conclude that professor John will have a cup of coffee (*coffee*) in the meeting. Since he is assumed to be present in the meeting, by axiom 3, he will be abnormal wrt being in his office ($ab3$). Such reasoning pattern is a kind of forward reasoning because it starts from the assumptions in the lowest level and performs reasoning upwards by following the direction of implications. This kind of forward reasoning provides a way of deriving intuitive and plausible conclusions in incomplete information when people perform commonsense reasoning.

From the above discussions, we may note that the direction of implications of axioms in a theory which formulates a given commonsense knowledge plays an important role in commonsense reasoning. Let T be a theory with P and Z be sets of predicate symbols as defined in circumscription. Let A be a ground atom whose predicate symbol is in P. If A does not appear in the consequent of any ground instance of an axiom, then A cannot be derived. Furthermore, if A appears in the consequents of some ground instances of axioms whose antecedents are not derivable, then A also cannot be derived. In both cases, A is assumed false. On the other hand, only the consequents of ground axioms in T can be derivable. Hence, commonsense knowledge itself gives us a clue to make intuitive and reasonable assumptions; i.e., we can perform commonsense reasoning just by following the direction of implication of axioms. This can be formalized by the following way.

Definition 4.1: Let T be a theory formulating a commonsense knowledge. Let p, q, and r be predicate symbols occurring in T. Then, p is called to be *dependent* on q (denoted by $q \gg p$) if

(i) there exists an axiom in T such that p occurs in the consequent and q occurs in the antecedent, or

(ii) p is dependent on r and r is dependent on q. □

Here, p dependent on q means that the truth value of some ground predicate $p(\ldots)$ is dependent on the truth value of some ground predicate $q(\ldots)$. Thus, $q(\ldots)$ will be derived before $p(\ldots)$. So, q has higher priority than p in evaluation. Let a level mapping M of a theory be a mapping from its set of predicate symbols to the non-negative integers.

Definition 4.2: A theory T with $P = \{p_1, \ldots, p_n\}$ is called *stratified* if there exists a level mapping M such that for each pair of predicate symbols p_i and p_j in P, $M(p_i) > M(p_j)$ if

(i) $p_i \gg p_j$ but not $p_j \gg p_i$,

(ii) $p_i \gg p_k$ but not $p_k \gg p_i$ with $M(p_k) = M(p_j)$, or

(iii) $p_k \gg p_j$ but not $p_j \gg p_k$ with $M(p_k) = M(p_i)$;

and $M(p_i) = M(p_j)$, otherwise. □

Consider the above theory T_2. By axioms 1 and 4, we have $ab1 \gg meeting$ and $meeting \gg ab3$, respectively. By Definition 4.1, we have $ab1 \gg ab3$, which is the only dependency we can derive. So, there exists a level mapping M such that $ab2$ and $ab3$ are in the same level, and $ab1$ is in a higher level. Thus, we have $P_1 = \{ab1 > ab3, ab2\}$ in prioritized circumscription. In this case, circumscribing T with $P = P_1$ and Z as before, we only obtain the intuitive minimal model M_1.

In some cases, the theory T is not stratified. For example, consider the following simplified version of the Nixon diamond example:

1. $\neg abQ \rightarrow abR$,
2. $\neg abR \rightarrow abQ$.

We call the theory T_3. In this theory, we have $abQ \gg abR$ and $abR \gg abQ$ by axioms 1 and 2, respectively. By transitivity, we also have $abQ \gg abQ$ and $abR \gg abR$. Clearly, theory T_3 is not stratified. In this case, we assign abQ and abR the same priority. Circumscribing T_3 with $P = \{abQ, abR\}$, two minimal models are obtained: $M_1 = \{abR\}$ and $M_2 = \{abQ\}$, both of which are intuitive conclusions as noted in Chapter 3.

4.3. Related Work

In order to cope with the anomalous extension problem, Morris [85] first demonstrated that the undesired extension of the YSP disappears if it is reformulated in terms of a TMS [21]. The professor John example (i.e., theory T_2) in the previous section can be expressed in the TMS by the following non-monotonic justifications

1. $has\_students \land out(ab1) \rightarrow meeting$,
2. $meeting \land out(ab2) \rightarrow coffee$,
3. $has\_students$,
4. $meeting \rightarrow ab3$,
5. $out(ab3) \rightarrow office$.

Note that each classical negation $\neg$ in front of a predicate symbol in P (to be circumscribed in circumscription) of T_2 is changed into *out* in the TMS formulation. These justifications generate a unique well-founded labeling which matches our intuition. Morris further investigated the reason why the anomalous extensions can be excluded and found out that it is due to the unidirectional nature of TMS inference.

Since defaults in default logic are also unidirectional, Morris [86] also performed a mapping from non-monotonic justifications into non-normal defaults such that the anomalous extension of the YSP also disappears. Consider the professor John example. It can be expressed by non-normal defaults as follows:

$$D_1 = \left\{ \frac{has\_students : M\neg ab1}{meeting}, \frac{meeting : M\neg ab2}{coffee}, \frac{: M\neg ab3}{office} \right\},$$

$$W_1 = \{has\_students,\ meeting \rightarrow ab3\}.$$

Clearly, the default theory (D_1, W_1) only generates the desired extension.

Due to the intimate relation between default logic and autoepistemic logic, Gelfond [32] also presented a reformulation of the YSP to avoid the unwanted expansion in autoepistemic logic. In this approach, the professor John example can be expressed by the following autoepistemic theory:

1. $has\_students \wedge \neg L\, ab1 \rightarrow meeting$,
2. $meeting \wedge \neg L\, ab2 \rightarrow coffee$,
3. $has\_students$,
4. $meeting \rightarrow ab3$,
5. $\neg L\, ab3 \rightarrow office$.

Obviously, there is only one stable expansion of the theory. Morris [87] also pointed out that $\neg L$ in AEL roughly corresponds to *not* in TMSs. This is another reason why Gelfond's reformulation can avoid anomalous extensions.

By noting the importance of the direction of implications in commonsense reasoning, You and Li [138] also proposed the *supported circumscription* to avoid unintuitive minimal models. They first defined *oriented clauses* to preserve the direction of implications in first-order theory. Then, they applied the notion of *supportedness* in logic programming to define their new definition of supported circumscription. A minimal model M of a theory T is *supported* if each atom in M is the consequent of a ground instance of an axiom in T. We look at the professor John example which is represented by the first-order theory T_2 in the previous section. The minimal model M_1 is supported because each atom in M_1 is in the consequents, while M_2 is not supported because $ab1$ is not the consequent of any axiom in T_2.

As pointed out in Section 4.1, if each illustrated example with anomalous extension is formulated by an extended logic program, the undesired conclusion is excluded. This is because each clause in extended logic programs is unidirectional; in addition, each answer set of an extended logic program is also supported. We think that it is more natural to express commonsense knowledge by using extended logic programs because extended logic programs possesses the above two advantages of eliminating anomalous extensions. We look at the professor John example which can be expressed by the following extended logic program Π_4:

1. $meeting \leftarrow has\_students \wedge not\ ab1$,
2. $coffee \leftarrow meeting \wedge not\ ab2$,
3. $has\_students \leftarrow$,
4. $ab3 \leftarrow meeting$,
5. $office \leftarrow not\ ab3$.

It is apparent that the unique answer set of program Π_4 is an intuitive conclusion. As we can see, all the above solutions to coping with the anomalous extension problem apply the unidirectional property of their formulations.

4.4. Discussions

In response to the anomalous extension problem of the famous YSP, many criticisms and other solutions have been proposed to deal with it. For the discussions and debates of these points, refer to the original paper by Hanks and McDermott [45] for details. In the following, we briefly discuss two commonsense theorem provers which also used the YSP as their demonstrative example. The first one is a circumscriptive theorem prover presented by Ginsberg [41] based on his multi-valued logic [39]. This simple prover does not allow prioritized circumscription. In his paper [41], Ginsberg used the original circumscriptive theory of the YSP in [45] to demonstrate that both the queries $t(alive, s_3)$ and $\neg t(alive, s_3)$ are disconfirmed in his prover and then to conclude that these queries do not follow from the circumscription. Note that the above demonstration has nothing to do with the anomalous extension problem of the YSP as discussed in Section 4.2; it is only used to show whether a query is followed from the circumscription so that the theorem prover can successfully prove the query.

The second one is a default logic theorem prover recently proposed by Munoz and Yang [88]. This prover primarily depends on constructing normal deduction graphs (NDGs) to perform default reasoning based on Kleene's [53] three-valued logic by changing the third truth value *unknown* into *maybe*. However, this approach has two limitations: (1) Only a subset of default theories are considered; i.e., the set of *ordered network* default theories [26] are assumed. (2) Only normal defaults with empty prerequisite and the consequent being of the form like Horn clauses are considered. In their paper [88], Munoz and Yang also used the original default theory of the YSP in [45] for demonstration. Because it is impossible to construct an NDG from *true* to $t(alive, s_3)$ and an NDG from *true* to $\neg t(alive, s_3)$, the theorem prover returns the answer *maybe* to the above two queries. Similar to the former prover, the latter prover also does not deal with the anomalous extension problem of the YSP. The following is a comment for the latter approach dealing with commonsense reasoning.

By intuition, when we say something may be true, we usually mean that we know that thing and the possibility that it happens is very high. On the other hand, when we say something is unknown, we usually mean that we have no idea about it. However, the modified three-valued logic adopted by the default logic theorem prover is too simple to differentiate this kind of difference. For example, suppose we have the default theory (D_2, W_2) with $D_1 = \emptyset$ and $W_2 = \{block(a) \vee block(b)\}$. By the latter approach, the prover will returns *maybe* for both queries $block(a)$ and $block(b)$. This answer is reasonable because we know that A or B is a block. However, for the query $block(c)$, the prover will also return *maybe* to mean that C may be a block. But by the above discussion of our intuition, because we have no idea about C, the prover should return a more reasonable answer *unknown*, for C may be a dog, a table, and so on.

CHAPTER 5

DEFEASIBLE CONCLUSIONS

As mentioned in Chapter 1, commonsense reasoning is a kind of plausible reasoning which draws conclusions from incomplete knowledge by making some reasonable default assumptions. Thus, some conclusions are derived from such default assumptions, while others are not. For the standard flying birds example, the commonsense knowledge that birds normally fly and Tweety is a bird can be formalized by the extended logic program:

$fly(X) \leftarrow bird(X) \wedge not\ ab_1(X),$

$bird(tweety) \leftarrow .$

The first clause states that if X is a bird and there is no evidence that X is abnormal in aspect 1 (i.e., wrt flying), then the conclusion that X can fly is drawn. This program has an answer set

$\{bird(tweety),\ fly(tweety)\},$

which corresponds with our intuition that Tweety is a bird and can fly. Note that the conclusion that Tweety can fly is assumed by the default assumption that Tweety is a normal bird wrt flying because we cannot derive the conclusion that Tweety is abnormal wrt flying. However, if new information showing that Tweety is an abnormal bird wrt flying is learned, then $fly(tweety)$ will be defeated. Thus, the conclusion $fly(tweety)$ is merely a temporary belief.

Now, suppose we are told that Tweety is also an ostrich and ostriches are birds and normally do not fly. Such new commonsense knowledge can be expressed by the clauses:

$ostrich(tweety) \leftarrow,$

$bird(X) \leftarrow ostrich(X),$

$\neg fly(X) \leftarrow ostrich(X) \wedge not\ ab_2(X),$

$ab_1(X) \leftarrow ostrich(X),$

where the last clause is an inheritance cancellation rule. The above augmented extended logic program has an answer set

$\{bird(tweety), ostrich(tweety), ab_1(tweety), \neg fly(tweety)\},$

which also agrees with our intuition that Tweety is an ostrich and cannot fly. Note that since Tweety is now an abnormal bird wrt flying, the previous conclusion $fly(tweety)$ is retracted. In addition, the new conclusion that Tweety cannot fly is assumed by the default assumption that Tweety is a normal ostrich wrt non-flying. From the above discussion, it is easy to see that $bird(tweety)$ and $ostrich(tweety)$ are strictly true because they are given facts, whereas $\neg fly(tweety)$ is assumed by default because it is derived from the assumption that Tweety is a normal ostrich wrt non-flying. It is noted that the answer set semantics [34] does not differentiate such a kind of difference.

In commonsense reasoning, it is important to identify which conclusions are assumed by default and which conclusions are strictly true, for only the conclusions assumed by default can be retracted if new information which conflicts with any default assumption made is learned. However, the current major formalizations of nonmonotonic reasoning surveyed in Chapter 2 are not able to distinguish between such two kinds of conclusions. This is due to the use of first-order logic by all of them as well as the use of the definition of minimal models in circumscription [71] and the ECWA [37] and the definition of fixed points in default logic [106], autoepistemic logic [84], and nonmonotonic logic [73] (for details, refer to Section 5.4). Even the answer set semantics defined for extended logic programs [34] which we apply to formalize commonsense knowledge also cannot distinguish between them. This is because in such a semantics, the answer to a ground query is *yes*, *no*, or *unknown* wrt an answer set. In the following, we investigate how our formalization of commonsense knowledge can be used to distinguish such a difference merely by the syntax of the extended logic programs. Then, we briefly explore how similar approaches as used in extended logic programs to identify defeasible conclusions can also be used in the major formalizations of nonmonotonic reasoning to achieve the same purpose.

5.1. Identifying Defeasible Conclusions

Let Π be an extended logic program. Let *Lit* be the set of all ground literals in the language of Π. Let Π^+ be the extended logic program obtained from Π by deleting each clause containing the negation-as-failure *not* in its body. Obviously, Π^+ does not contain *not*, so its answer set, $\alpha(\Pi^+)$, is defined. Recall that $\alpha(\Pi^+)$ is defined to be the smallest subset S of *Lit* such that for any ground clause instance $L_0 \leftarrow L_1 \wedge \cdots \wedge L_m$ in Π^+, if each L_i is in S (for $i = 1, \ldots, m$), then L_0 is also in S. If S contains a pair of complementary literals, then $S = Lit$ and Π^+ is called *contradictory*. It is interesting to see that $\alpha(\Pi^+)$ is monotonic in the sense that the cardinality of $\alpha(\Pi^+)$ will not decrease if Π is augmented.

Theorem 5.1: $\alpha(\Pi^+)$ is monotonic.

Proof: There are two cases to consider.

(i) If Π^+ is contradictory, then $\alpha(\Pi^+)$ will not be changed by adding any clauses into Π because $\alpha(\Pi^+) = Lit$.

(ii) If Π^+ is not contradictory, there are still two subcases to consider. (a) Suppose the added clause contains *not* in its body. Clearly, Π^+ will not include such a clause, and hence, $\alpha(\Pi^+)$ is not changed. (b) Suppose the added clause C is

$L_0 \leftarrow L_1 \wedge \cdots \wedge L_m$, which does not contain *not* in the body. Obviously, C will also be added into Π^+. Let S be the answer set of Π^+ before C is added. By definition, for any ground instance of C by θ (a substitution), if each $L_i\theta$ is in S (for $i = 1, \ldots, m$), then $L_0\theta$ is added to S. Since no literal is deleted from S and $L_0\theta$ is added, the cardinality of $\alpha(\Pi^+)$ increases. □

A literal L in *Lit* is said to be ***strictly true*** if L is in $\alpha(\Pi^+)$, for $\alpha(\Pi^+)$ is monotonic. Thus, any literal in $\alpha(\Pi^+)$ will not be defeated by new information. The ***complement*** of L, denoted by $\overline{L}$, is A if $L = \neg A$ or is $\neg A$ if $L = A$, where A is an atom. Similarly, a literal L in *Lit* is said to be ***strictly false*** if $\overline{L}$ is in $\alpha(\Pi^+)$. In other words, a literal is strictly false if its complement is strictly true. Obviously, if Π^+ is not contradictory, a literal cannot be both strictly true and strictly false.

Let $L \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \textit{not } B_1 \wedge \cdots \wedge \textit{not } B_n$ (for $m, n \geq 0$ and $m + n \geq 1$) be a ground instance of a clause in Π. Then, L is called ***true by default*** if

(i) each A_i is either strictly true or true by default,

(ii) none of B_j is strictly true, and

(iii) at least one A_i is true by default if $n = 0$.

Also, the expression *not* B_j is defined to be ***true by default*** if B_j is not strictly true. Clearly, a literal which is true by default must be derived by at least one expression *not* B_j where B_j is not strictly true. Therefore, such a literal is a possible defeasible conclusion because once B_j is proved to be strictly true by new information, it will be retracted; i.e., it will no longer be true by default derived by *not* B_j. Similarly, a literal is called ***false by default*** if its complement is true by default. A literal in *Lit* is called to have a ***strict*** truth value if it is strictly true or false and is called to have a ***default*** truth value if it is true or false by default. Now, a literal in *Lit* is called ***unsupported*** if it has no strict or default truth value. Note that a literal in *Lit* can have both strict and default truth values.

Example 5.1: Consider $\Pi_1 = \{\neg P \leftarrow, P \leftarrow \neg Q\}$. Clearly, $\Pi_1^+ = \Pi_1$ because there is no occurrence of *not* in Π_1. Since $\alpha(\Pi_1) = \{\neg P\}$, $\neg P$ is strictly true, P is strictly false, and both Q and $\neg Q$ are unsupported. □

Example 5.2: Consider $\Pi_2 = \{\neg P \leftarrow, Q \leftarrow \neg P\}$. Similarly, $\Pi_2^+ = \Pi_2$. Since $\alpha(\Pi_2) = \{\neg P, Q\}$, both $\neg P$ and Q are strictly true, or equivalently, both P and $\neg Q$ are strictly false. Note that Π_1 and Π_2 represent exactly the same theory in first-order logic; however, they express different programs in extended logic programs because of the unidirectional if, $\leftarrow$. □

Example 5.3: Consider $\Pi_3 = \{P \leftarrow \textit{not}\ Q\}$. Since $\Pi_3^+ = \alpha(\Pi_3^+) = \emptyset$ (the empty set), *not* Q is true by default, and P is also, whereas Q and $\neg Q$ are unsupported. □

Example 5.4: Consider $\Pi_4 = \{Q \leftarrow \textit{not}\ P, P \leftarrow \textit{not}\ Q\}$. Similarly, $\Pi_4^+ = \alpha(\Pi_4^+) = \emptyset$. Thus, P, Q, *not* P, and *not* Q are all true by default, whereas $\neg P$ and $\neg Q$ are false by default. □

The following two examples illustrate that some extended logic programs have literals which are both true by default and false by default. It is coincident that these two extended logic programs have no answer sets.

Example 5.5: Consider $\Pi_5 = \{P \leftarrow \textit{not}\ Q, \neg P \leftarrow \textit{not}\ R\}$. Clearly, $\Pi_5^+ = \emptyset$. Thus, both *not* Q and *not* R are true by default, and P and $\neg P$ are also. By definition, P and $\neg P$ are also false by default. □

Example 5.6: Consider $\Pi_6 = \{P \leftarrow \textit{not}\ \neg P, Q \leftarrow P, \neg Q \leftarrow P\}$. We have $\Pi_6^+ = \{Q \leftarrow P, \neg Q \leftarrow P\}$ and $\alpha(\Pi_6^+) = \emptyset$. Hence, *not* $\neg P$ is true by default, and P, Q, and $\neg Q$ are also. By definition, $\neg P$, Q, and $\neg Q$ are false by default. □

Because Π^+ contains no non-standard logic connective (i.e., the negation-as-failure *not*), let Σ_Π be the corresponding first-order theory of Π^+ with the unidirectional if, $\leftarrow$, replaced by the bidirectional material implication, $\rightarrow$. Let $thm(T)$

denote the set of theorems derived from a first-order theory T. Then, the answer set of Π^+ is a subset of the set of theorems derived from its corresponding first-order theory Σ_Π.

Theorem 5.2: $\alpha(\Pi^+) \subseteq thm(\Sigma_\Pi)$.

Proof: Since first-order logic is monotonic, and Π^+ is a subset of Σ_Π in the sense that Σ_Π is allowed to use the contrapositive, whereas Π^+ is not, the theorem is proved. $\square$

Let us define the following notations:

$T_s = \{L | L \text{ is strictly true}\}$,

$F_s = \{L | L \text{ is strictly false}\}$,

$T_d = \{L | L \text{ is true by default}\}$,

$F_d = \{L | L \text{ is false by default}\}$,

$U = \{L | L \text{ is unsupported}\}$.

Clearly, $T_s = \alpha(\Pi^+)$. By definition, it is easy to see that

(i) $Lit = T_s \cup F_s \cup T_d \cup F_d \cup U$,

(ii) $U = Lit - (T_s \cup F_s \cup T_d \cup F_d)$, and

(iii) the expression *not* L is true by default if $L \in Lit - T_s$.

Note that if a literal is unsupported, then its complement is also unsupported. The following theorem shows that if S is an answer set of Π, then $\alpha(\Pi^+)$ must be a subset of S.

Theorem 5.3: If S is an answer set of Π, then $\alpha(\Pi^+) \subseteq S$.

Proof: Since S is answer set of Π, $S = \alpha(\Pi^S)$. By definition, Π^S is obtained from Π by deleting each clause containing *not* L in its body with $L \in S$ and by deleting *not* L in the remaining clauses in Π. Clearly, Π^+ must be contained in Π^S, for it does not contain any *not*. Because $\alpha(\Pi^S)$ is monotonic (by Theorem 5.1) and $\Pi^+ \subseteq \Pi^S$, $\alpha(\Pi^+) \subseteq \alpha(\Pi^S) = S$. $\square$

From the above theorem, it is interesting to see one of the properties of contradictory extended logic programs. If Π is contradictory, then Π^+ is also; and vice versa. Thus, whether or not an extended logic program is contradictory can easily be identified by checking Π^+ only.

Theorem 5.4: Π^+ is contradictory if and only if Π is also.

Proof: (i) if part ($\Leftarrow$): Since Π is contradictory, it has an answer set $S = Lit$. Because Π^S is obtained from Π by deleting each clause containing *not* in its body, we have $\Pi^S = \Pi^+$. By definition, $S = \alpha(\Pi^S) = \alpha(\Pi^+)$, which means Π^+ is contradictory.

(ii) only-if part ($\Rightarrow$): Similarly, since Π^+ is contradictory, it has an answer set $S = Lit$. Suppose Π is not contradictory. There are two cases to consider. (a) Π has answer sets. In this case, each answer set of Π must not be Lit. Let S' be an answer set of Π. By Theorem 5.3, $\alpha(\Pi^+) = Lit$ is a subset of $\alpha(\Pi^{S'})$. Clearly, $\alpha(\Pi^{S'})$ must also be Lit, which contradicts the assumption. (b) Π has no answer set. Obviously, this case does not hold because in case (a), Lit is indeed an answer set of Π. □

Given an extended logic program Π, it has been shown that $\alpha(\Pi^+) = T_s$ is monotonic. In the following, we show that T_d and U are nonmonotonic in the sense that the cardinalities of T_d and U may decrease if Π is augmented. Since T_s is monotonic and is also contained in any answer set of Π, the nonmonotonic behavior of commonsense reasoning formalized by using extended logic programs is indeed owing to the nonmonotonicity of T_d.

Theorem 5.5: T_d and U are nonmonotonic.

Proof: (by contradiction)

(i) Assume T_d is monotonic. Let Π be $\{P \leftarrow not\ Q\}$. By definition, $T_d = \{P\}$. Suppose Q is added to Π. Then, Π becomes $\{P \leftarrow not\ Q, Q \leftarrow\}$, and T_d becomes $\emptyset$. Obviously, the cardinality of T_d decreases, which contradicts the assumption.

(ii) Assume U is monotonic. Let Π be the same as in (i). By definition, U

$= \{Q, \neg Q\}$. Suppose $\neg Q$ is added to Π. Then, Π becomes $\{P \leftarrow \textit{not}\ Q, \neg Q \leftarrow\}$, and U becomes $\emptyset$. Obviously, the cardinality of U decreases, which contradicts the assumption. □

5.2. Answer Set Semantics with Defeasible Conclusions

In the original answer set semantics, literals in an answer set are all assumed to have equal weight; i.e., they are all regarded as true in an extended logic program. As noted earlier, in commonsense reasoning, some conclusions are strictly true, whereas others are assumed by default. In the following, we enhance the answer set semantics so that it can differentiate such two kinds of conclusions.

Definition 5.1: Let Π be an extended logic program, and let S be an answer set of Π. A literal L in S is called *strictly true* and $\overline{L}$ is called *strictly false* if L is in $\alpha(\Pi^+)$. On the other hand, a literal L in S is called *true by default* wrt S and $\overline{L}$ is called *false by default* wrt S if L is not in $\alpha(\Pi^+)$. □

Recall that T_d is the set of possible defeasible conclusions. The following theorem states that any literal which is true by default wrt an answer set must be in T_d. Thus, a literal which is true by default wrt an answer set is called a *defeasible conclusion.*

Theorem 5.6: Let S be an answer set of Π. If L is true by default wrt S, then L is also in T_d.

Proof: Let $L \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \textit{not}\ B_1 \wedge \cdots \wedge \textit{not}\ B_n$ be a ground instance of a clause in Π which derives L. By definition, $\{A_1, \ldots, A_m\}$ must be a subset of S, and each B_j must not be in S (for all $j = 1, \ldots, n$) so that L is in S. Since L is true by default wrt S, it is not in T_s. This means that in the above clause, either $n \geq 1$ or at least one A_i is not in T_s if $n = 0$. Obviously, these conditions satisfy the definition that L is true by default. □

As mentioned earlier, U is defined to the set $Lit - (T_s \cup F_s \cup T_d \cup F_d)$. The following theorem shows that any literal in U must not occur in any answer set of Π.

Theorem 5.7: Let L be a literal in U. Then, L does not appear in any answer set of Π.

Proof: Let D_t and D_f be the sets of literals which are true by default and false by default wrt any answer set of Π, respectively. Clearly, $D_t \subseteq T_d$ and $D_f \subseteq F_d$. Thus, $U = Lit - (T_s \cup F_s \cup T_d \cup F_d) \subseteq Lit - (T_s \cup F_s \cup D_t \cup D_f)$, which is the set of literals that must not appear in any answer set of Π. $\square$

Once the literals in an answer set can be identified to be strictly true or true by default, it is easy to differentiate the answer for a ground query to be *yes*, *no*, *yes by default*, *no by default*, or *unknown*. In the following, we define a query-answering system for the enhanced answer set semantics to obtain such answers for ground queries.

Definition 5.2: Let Q be a ground query, and let S be an answer set of Π. Then, the answer for Q wrt S is

(i) *yes* if Q is strictly true,

(ii) *no* if Q is strictly false,

(iii) *yes by default* if Q is true by default wrt S,

(iv) *no by default* if Q is false by default wrt S, or

(v) *unknown* if neither Q nor $\neg Q$ is in S. $\square$

Since an extended logic program may have multiple answer sets if a ground query is a literal which is strictly true (respectively, strictly false), then the system will reply only one answer *yes* (respectively, *no*) to such a query because it is strictly true (respectively, strictly false) in all answer sets. On the other hand, if a ground query is a literal which is not strictly true or strictly false, then the system will offer multiple answers to such a query according to the above definition, each for an answer set. There is a special case where a ground query is a literal which is unsupported. Because both the unsupported literal and its complement do not appear in any answer set of Π, the system will reply only one answer *unsupported* to such a query to mean that it is *unknown* for every answer set.

Example 5.7: Consider $\Pi_7 = \{P \leftarrow, Q \leftarrow not\ R\}$. We have $T_s = \{P\}$ and $T_d = \{Q\}$. Π_7 has an answer set $\{P, Q\}$. Since P and Q are, respectively, in T_s and T_d, P and Q are, respectively, strictly true and true by default. Thus, the answers for queries P, $\neg Q$, and R are *yes*, *no by default*, and *unknown*, respectively. □

Example 5.8: Consider $\Pi_4 = \{Q \leftarrow not\ P, P \leftarrow not\ Q\}$ in Example 5.4. We have $T_s = \emptyset$ and $T_d = \{P, Q\}$. Π_4 has two answer sets: $\{P\}$ and $\{Q\}$. Since both P and Q are in T_d, they are both true by default. The answers for the query P will be *yes by default* and *unknown*, whereas the answers for the query $\neg Q$ will be *unknown* and *no by default*. Note that since Π_4 has two answer sets, each response from the system corresponds to an answer set. □

5.3. Applying Defeasible Conclusions to Commonsense Reasoning

In this section, we demonstrate how the enhanced answer set semantics can be used to identify defeasible conclusions in the standard flying bird example illustrated in the beginning of this chapter. Consider the first part of commonsense knowledge that birds normally fly and Tweety is a bird. We repeat the corresponding extended logic program in the following and called it Π_8.

1. $fly(X) \leftarrow bird(X) \wedge not\ ab_1(X)$,
2. $bird(tweety) \leftarrow$.

Clearly, $\Pi_8^+ = \{bird(tweety) \leftarrow\}$, and $\alpha(\Pi_8^+) = \{bird(tweety)\}$. Since $ab_1(tweety)$ is not in $\alpha(\Pi_8^+)$, *not* $ab_1(tweety)$ is true by default and $fly(tweety)$ is also. Thus, $T_d =$

$\{fly(tweety)\}$. Program Π_8 has an answer set

$\{bird(tweety), fly(tweety)\}$.

Since $bird(tweety)$ is in $\alpha(\Pi_8^+)$, it is strictly true. On the other hand, since $fly(tweety)$ is in T_d, it is true by default. By the enhanced answer set semantics, the conclusion that Tweety is a bird, $bird(tweety)$, is strictly true, while the conclusion that Tweety can fly, $fly(tweety)$, is true by default. Thus, $fly(tweety)$ is a defeasible conclusion.

Now, we consider the second part of commonsense knowledge that Tweety is also an ostrich and ostriches are birds and normally do not fly. Here, we also repeat the corresponding clauses in the following:

3. $ostrich(tweety) \leftarrow$,
4. $bird(X) \leftarrow ostrich(X)$,
5. $\neg fly(X) \leftarrow ostrich(X) \wedge not\ ab_2(X)$,
6. $ab_1(X) \leftarrow ostrich(X)$.

We call Π_8 augmented with the above clauses to be Π_9. By definition, we have $\alpha(\Pi_9^+) = \{bird(tweety), ostrich(tweety), ab_1(tweety)\}$. Since $ab_2(tweety)$ is not in $\alpha(\Pi_9^+)$, $not\ ab_2(tweety)$ is true by default, and $\neg fly(tweety)$ is also. Thus, $T_d = \{\neg fly(tweety)\}$. Program Π_9 has an answer set

$\{bird(tweety), ostrich(tweety), ab_1(tweety), \neg fly(tweety)\}$.

Since $bird(tweety)$, $ostrich(tweety)$, and $ab_1(tweety)$ are all in $\alpha(\Pi_9^+)$, they are all strictly true. Note that because $ab_1(tweety)$ is strictly true in Π_9, the conclusion $fly(tweety)$ in Π_8 is retracted. Since $\neg fly(tweety)$ is in T_d, it is true by default. By the enhanced answer set semantics, the conclusions that Tweety is an ostrich and a bird and is abnormal wrt flying are strictly true, while the conclusion that Tweety cannot fly is true by default. Hence, $\neg fly(tweety)$ is a defeasible conclusion.

5.4. Defeasible Conclusions in Other Formalizations

Similar to the answer set semantics, the major formalizations of nonmonotonic reasoning also do not identify defeasible conclusions. In this section, we briefly discuss how similar approaches as used in extended logic programs to identify defeasible conclusions can also be used in such formalizations. Because some of these formalizations share the same properties, we divide them into two groups: One is the formalizations based on fixed point construction, and the other is those based on minimal model definition.

5.4.1. Fixed Point Construction Based Formalizations

The definitions of *extension* in default logic, of *stable expansion* in autoepistemic logic, and of *fixed point* in nonmonotonic logic are all based on fixed point construction. For convenience, the term "extension" is used to mean one of the above definitions according to the context. All these three formalizations of nonmonotonic reasoning have the same characteristic that they all use a modal operator, either M or L, in first-order logic. Defeasible conclusions in these formalizations can be identified in the following:

Let T be a theory of one of the above three logics. Let T' be a theory obtained from T by deleting each formula containing modal operators. Obviously, T' is a first-order theory. Recall that we use $thm(A)$ to denote the set of theorems derived from a first-order theory A. By definition, it is easy to prove that if E is an extension of T, then $thm(T')$ must be a subset of E. Similar to the definition defined in extended logic programs, literals in E can be differentiated as follows: A literal L in E is called *strictly true* and $\overline{L}$ is called *strictly false* if L is in $thm(T')$. On the other hand, a literal L in E is called *true by default* wrt E and $\overline{L}$ is called *false by default* wrt E

if L is not in $thm(T')$. Clearly, a conclusion which is true by default wrt any extension is a defeasible conclusion.

Example 5.9: We illustrate the above method of identifying defeasible conclusions in default logic. Consider the first part of the flying bird example. It can be formalized by the default theory $T_1 = (D_1, W_1)$:

$$D_1 = \left\{ \frac{bird(X) : M fly(X)}{fly(X)} \right\}.$$

$$W_1 = \{bird(tweety)\},$$

The default theory T_1 has an extension $E_1 = \{bird(tweety), fly(tweety)\}$. Note that this extension cannot differentiate which conclusions are defeasible. By our method, $T_1' = W_1$, and $thm(T_1') = \{bird(tweety)\}$. Therefore, $bird(tweety)$ is strictly true because it is in $thm(T_1')$, while $fly(tweety)$ is a defeasible conclusion because it is not in $thm(T_1')$. □

5.4.2. Minimal Model Definition Based Formalizations

In terms of model-theoretical definition, it is shown that the ECWA is equivalent to the first-order version of circumscription. Both of these two formalizations of nonmonotonic reasoning use first-order theories to formalize commonsense knowledge together with two sets of predicate symbols: P and Z. The set P contains predicate symbols to be minimized, while the set Z contains predicate symbols which are allowed to vary for achieving the foregoing minimization. Let T be a first-order theory which formalizes a commonsense knowledge. Unlike the previous approach, we cannot extract a theory from T by identifying modal operators. On the contrary, since T is a first-order theory, it is easy to obtain the set, $thm(T)$, of theorems derived from T. It can be shown that for a (P, Z)-minimal model M of T, $thm(T)$ must be a subset of $thm(T \cup M)$. Therefore, defeasible conclusions can be identified in the first-order version of circumscription and the ECWA by the following definition. A literal L in

$thm(T \cup M)$ is called *strictly true* and $\overline{L}$ is called *strictly false* if L is in $thm(T)$. On the other hand, a literal L in $thm(T \cup M)$ is called *true by default* wrt M and $\overline{L}$ is called *false by default* wrt M if L is not in $thm(T)$. Obviously, a conclusion that is true by default wrt any (P, Z)-minimal model is a defeasible conclusion.

Example 5.10: For the above flying bird example, it can be formalized by the circumscriptive theory T_2 with $P = \{ab_1\}$ and $Z = \{fly\}$ as follows:

1. $bird(tweety)$,
2. $bird(X) \wedge \neg ab_1(X) \rightarrow fly(X)$.

The theory T_2 has a (P, Z)-minimal model $M = thm(T_2 \cup M) = \{bird(tweety), fly(tweety)\}$. Note that the model M also cannot identify which conclusions are defeasible. In our approach, $thm(T_2) = \{bird(tweety)\}$. Clearly, $bird(tweety)$ is strictly true because it is in $thm(T_2)$, while $fly(tweety)$ is a defeasible conclusion because it is not in $thm(T_2)$. □

CHAPTER 6

CONFLICT RESOLUTION

Recently, the famous Yale shooting problem (YSP) presented by Hanks and McDermott [45] in temporal reasoning has drawn a lot of attention in the AI research community with respective to discrepancies between intuition and formalization in commonsense reasoning. This is because they formalized such a problem by using three major formalizations of nonmonotonic reasoning (i.e., default logic, circumscription, and nonmonotonic logic) and observed that all of these formalizations generate two conclusions; only one of them corresponds with our intuition, and the unintuitive one is known as the ***anomalous*** extension. Subsequently, Morris [86] also presented a simple example structurally similar to the YSP from taxonomic reasoning, which also produces an anomalous extension. This example demonstrates that the anomalous extension problem not only occurs in temporal reasoning but also appears in non-temporal reasoning, such as taxonomic reasoning.

In response to the anomalous extension problem, several solutions for the YSP have been proposed to exclude the unintended conclusion. For example, Morris [86] first reformulated the YSP in terms of a truth maintenance system (TMS) [21] to exclude the unwanted conclusion. Furthermore, by mirroring the TMS formulation, he used non-normal defaults in default logic to simulate nonmonotonic justifications in TMSs to avoid the anomalous extension. Similar to the above approach, Gelfond [32] also proposed an elegant solution based on autoepistemic logic (AEL) to exclude the undesired expansion. By employing the unidirectional nature of TMSs and the supportedness concept in logic programming, You and Li [138] presented the *supported circumscription* to obtain the same result. Note that in Chapter 3, our formalization of commonsense knowledge by using extended logic programs for the YSP also generates the intended conclusion only. However, of these solutions, only the TMS representation can appropriately revise its beliefs in response to conflicting information through *dependency-directed backtracking.* All other solutions are going to collapse when conflicting information is learned.

It seems that dependency-directed backtracking is a powerful mechanism for conflict resolution. Indeed, in the literature, there are several proposals which borrow the similar idea of this mechanism to resolve conflicts. For example, Morris' *stable closure* [87] is essentially by adding a minimal set of first-order axioms to an autoepistemic theory to simulate dependency-directed backtracking to resolve conflicts. However, this approach has some peculiarities wrt the behavior of a TMS. Moreover, one major problem with this approach is that how to find a minimal set of axioms for an autoepistemic theory is not clear. Giordano and Martelli's *generalized stable model* [42] was proposed to capture the idea of dependency-directed backtracking in logic programming by using contraposition to resolve conflicts. However, the anomalous extension of the YSP reappears in this approach. Although they claimed that it is not a problem with the generalized stable model semantics because this semantics simulates the incremental view of a TMS, what we want is the beliefs obtained from the view of a static TMS. Along the line of Giordano and Martelli, in this chapter, we propose the *generalized answer set semantics* for extended logic programs by using contraposition and by keeping the unidirectional property of clauses. This semantics really mimics the process of dependency-directed backtracking to resolve conflicts from the view of a static TMS. Finally, a similar approach is also applied to default logic, AEL, and circumscription to resolve conflicts.

6.1. Background

For simplicity of discussion, throughout this chapter, we will consider the simplified example adopted from [86] which also contains the essential structure of the YSP, but with less extraneous detail. The simplified example contains the following set of first-order axioms:

$$\{\neg abB \rightarrow abA, \quad \neg abA \rightarrow \neg fly, \quad \neg abB \rightarrow fly\},$$

and with the normal defaults $\{: M\neg abA/\neg abA, : M\neg abB/\neg abB\}$ if it is formalized by default logic or with $P = \{abA, abB\}$ and $Z = \{fly\}$ if it is formalized by circumscription. In which abA, abB, and fly stand for the propositions "Tweety is an abnormal animal," "Tweety is an abnormal bird," and "Tweety can fly," respectively. As we can see, both formalizations generate two conclusions; one concludes that Tweety is an abnormal animal and can fly (which agrees with our intuition), whereas the other one concludes that Tweety is an abnormal bird and cannot fly (which is unintuitive and thus is an anomalous extension).

In order to cope with this anomalous extension problem, Morris [86] has shown that if the YSP is formulated in terms of a TMS, then the unwanted conclusion is excluded. For the simplified example, its TMS representation can be the following set of justifications (refer to Section 6.2 for a review of TMSs):

$$\{out(abB) \rightarrow abA, \quad out(abA) \rightarrow no\_fly, \quad out(abB) \rightarrow fly\}.$$

This generates a unique well-founded labeling IN = $\{abA, fly\}$ and OUT = $\{abB, no\_fly\}$, which matches our intuition. If we subsequently learn that no_fly is true, then a contradiction occurs which causes dependency-directed backtracking. The contraction is resolved by introducing a new justification, $no\_fly \rightarrow abB$, that makes abB coming *in*, which results in abA and fly going *out*. Thus, the beliefs are appropriately revised in response to new conflicting information.

By mapping nonmonotonic justifications to non-normal defaults, Morris [86] also showed that the default logic representation of the YSP by using non-normal defaults can avoid the unintended extension. For the simplified example, it can be represented by the following set of non-normal defaults:

$$\left\{\frac{: M\neg abB}{abA}, \quad \frac{: M\neg abA}{\neg fly}, \quad \frac{: M\neg abB}{fly}\right\},$$

which produces a single extension. However, if $\neg fly$ is added as a new axiom, there is no extension. This means that the non-normal default representation cannot deal with conflicting information.

Gelfond [32] also presented an elegant solution, similar to the non-normal default representation, using AEL to produce a single stable expansion for the YSP. The simplified example can be formalized by this approach as follows:

$$\{\neg L\, abB \rightarrow abA, \quad \neg L\, abA \rightarrow \neg fly, \quad \neg L\, abB \rightarrow fly\}.$$

This axiom set has a unique stable expansion that contains abA and fly. However, if $\neg fly$ is added to the axiom set, there is no stable expansion.

You and Li [138] proposed the supported circumscription by employing the unidirectional nature of TMSs and supportedness concept in logic programming to exclude the unintended conclusion of the YSP. For the simplified example, this approach only obtains a single supported model $\{abA, fly\}$. However, if $\neg fly$ is learned to be true later, the only (P, Z)-minimal model $\{abB\}$ is not supported.

In Chapter 3, we have shown that the undesired conclusion of the YSP is also excluded by our formalization of commonsense knowledge by using extended logic programs. The simplified example can be expressed by our representation as follows:

$$\{abA \leftarrow not\ abB, \quad \neg fly \leftarrow not\ abA, \quad fly \leftarrow not\ abB\}.$$

This program also produces a single answer set. However, if $\neg fly$ is added to the program, there is no answer set.

As we can see from the above solutions to coping with the anomalous extension problem for the YSP, only the TMS representation can revise its beliefs appropriately in response to conflicting information, while all other solutions are not able to deal with this situation. This is because only TMSs have the conflict resolution mechanism dependency-directed backtracking, which is briefly reviewed in the next section.

6.2. Truth Maintenance Systems

In the literature, two classes of truth maintenance systems (TMSs) are widely discussed: Justification-based TMSs [21] support nonmonotonicity and maintain a single set of beliefs at a time which can be revised when new conflicting information is entered. Assumption-based TMSs [20] maintain multiple sets of beliefs simultaneously, but support nonmonotonicity only through awkward extensions to the basic system. In the following, we only review justification-based TMSs on which Morris' reformulation is based.

A *justification* given to a TMS is an implication of the form

$$a_1 \wedge \cdots \wedge a_n \wedge out(b_1) \wedge \cdots \wedge out(b_m) \rightarrow c,$$

where each a_i, each b_j, and c are propositions and $n, m \geq 0$. The intended meaning of such a justification is "belief in $a_1, \ldots, a_n$ and disbelief in $b_1, \ldots, b_m$ justify the belief in c." Each a_i is called an *in-justifier* of the justification, while each b_j is an *out-justifier*. A justification is called *nonmonotonic* if it contains at least one *out*-justifier in its antecedent. A justification with the empty antecedent (i.e., $n = m = 0$) is called a *premise*, which represents a proposition always true.

Given a set J of justifications, the purpose of a TMS is to assign each proposition in J a label *in* (believed) or *out* (disbelieved) so that the labeling is well-founded. Before defining well-founded labelings, some required concepts are introduced first.

A justification is called to be *satisfied* by a labeling if its conclusion is labeled *in*, one of its *in*-justifiers is labeled *out*, or one of its *out*-justifiers is labeled *in*. A labeling is called *valid* if each justification in J is satisfied. A proposition is called *well-justified* if it is the conclusion of some justification whose *out*-justifiers are all labeled *out*, and whose *in*-justifiers are all themselves well-justified. Note that a well-justified proposition is justified by a non-circular argument. A labeling is called *well-founded* if it is valid and each proposition labeled *in* is well-justified. For convenience, we use IN and OUT to denote the sets of propositions labeled *in* and *out* in a well-founded labeling, respectively.

A well-justified proposition supported by at least one *out*-justifier is called an *assumption*. Obviously, an assumption will be believed until one of its *out*-justifiers comes *in*. A labeling is called *inconsistent* if it contains a pair of complementary propositions labeled *in*. In our notation, the complement of a proposition is prefixed by "*no_*." For example, *fly* and *no_fly* are a pair of complementary propositions. When a well-founded labeling obtained from a TMS is inconsistent, a contradiction is found and the *dependency-directed backtracking* mechanism is invoked to resolve the contradiction. However, not all contradictions can be resolved by the mechanism. Only the contradictions derived by some assumptions may be resolved. The major idea for resolving conflicts by dependency-directed backtracking is to get rid of the assumptions that lead to a contradiction. This can be accomplished as follows: The mechanism first traces back from the contradiction to find the supporting assumptions which cause inconsistency, then picks a relevant assumption, and puts it *out* by making one of its *out*-justifiers *in*. To achieve the last step, a new justification is introduced to the original set of justifications so that the labeling is revised. As we can see, dependency-directed backtracking not only properly revises its set of beliefs in response to conflicting information for a given set of justifications but also modifies (augments) the set of justifications itself.

Example 6.1: Consider the TMS representation of the simplified example repeated in the following:

1. $out(abB) \rightarrow abA$,
2. $out(abA) \rightarrow no\_fly$,
3. $out(abB) \rightarrow fly$.

Since abB is not the conclusion of any justification, there is no way for a well-founded labeling to label it *in*. Because abB has to be labeled *out*, abA and fly will be labeled *in* by justifications 1 and 3, respectively. This in turn causes no_fly to be labeled *out* by justification 2. Therefore, the labeling IN = $\{abA,\ fly\}$ and OUT = $\{abB,\ no\_fly\}$ is the only well-founded and consistent labeling. □

Example 6.2: Suppose no_fly is learned to be true later. Then, the following premise is added to the above set of justifications:

4. no_fly.

The only well-founded labeling becomes IN = $\{abA, fly, no\_fly\}$ and OUT = $\{abB\}$, which is inconsistent and thus causes dependency-directed backtracking. Since the contradiction is supported by the premise no_fly (justification 4) and the assumption fly (justification 3), fly is the only assumption which needs to be moved *out* by making the only *out*-justifier abB of it coming *in*. In order to do so, the TMS will add the following new backward justification to the above set of justifications:

5. $no\_fly \rightarrow abB$.

This allows to eliminate the contradiction and to obtain the correct labeling IN = $\{no\_fly,\ abB\}$ and OUT = $\{abA,\ fly\}$. Note that the revised labeling agrees with our intuition that Tweety is an abnormal bird and cannot fly. Note also that the set of justifications is augmented by justification 5. □

6.3. Related Work

The conflict resolution mechanism dependency-directed backtracking employed in TMSs is essentially by putting *in* one of the *out*-justifiers supporting the beliefs in contradictions which are found in an inconsistent labeling. The similar concept is also adapted by other solutions to resolve conflicts. For example, by borrowing the idea of putting *in* some proposition, Morris [87] proposed the *stable closure* for AEL by adding a minimal set of first-order axioms to response the collapse of stable expansions when conflicting information is obtained.

Let A be an autoepistemic theory. A *stable closure* of A is a solution E of fixed point equation

$$E = cl(A \cup G \cup \{L\,x | x \in E\} \cup \{\neg L\,x | x \notin E\}),$$

where cl is first-order logical closure, and G is a minimal set (possibly empty) of first-order axioms. The stable closure E is said to be *generated* by G. The minimality condition on G means that if $G' \subseteq G$ and E is also a solution generated by G', then $G' = G$. Note that each stable expansion of A is also a stable closure of A because $G = \emptyset$ in this case.

Consider the simplified example formalized by AEL in Section 6.1. It has a unique stable expansion, which is also the only stable closure. Note that if $\neg fly$ is added as a new axiom, there is no stable expansion; but there is a stable closure containing $\neg L\,abA$ and $\neg fly$, which is generated by $\{abB\}$.

The revision of beliefs using stable closure for the simplified example appears to agree well with the one obtained from dependency-directed backtracking. Due to this similarity, Morris further drew a rough analogy between $\neg L\,x$ in AEL and $out(x)$ in TMSs. However, the definition of stable closure has some peculiarities wrt the behavior of TMSs as noted by Morris himself: (i) The axiom $L\,p \vee p$ has a nontrivial stable closure $\{p\}$, while it has no stable expansion; however, the corresponding TMS representation, $out(p) \rightarrow p$, has no well-founded labeling. (ii) The axiom set $\{Lp,\ L\neg p\}$ has a unique stable closure, the set of all sentences; however, the corresponding TMS representation, $out(p) \rightarrow false$ and $out(no\_p) \rightarrow false$, has the only inconsistent well-founded labeling $\text{IN} = \{p,\ no\_p\}$, whose contradiction cannot be resolved by dependency-directed backtracking. The most important problem with stable closure is that how to find a minimal set G of first-order axioms for an autoepistemic theory is not clear. Owing to the above mentioned drawbacks, Morris [87] concluded "the notion of stable closure appears to capture at least part of the idea of dependency-directed backtracking."

Recently, Giordano and Martelli [42] presented the *generalized stable model semantics* for TMSs which is able to capture the idea of dependency-directed backtracking to resolve conflicts. Their conflict resolution process is essentially based on the contrapositive use of justifications to resolve inconsistencies. For instance, consider Example 6.2. By justification 4, *no_fly* is labeled *in* because it is a premise. Then, using the contrapositive of justification 3, $no\_fly \rightarrow abB$, proposition abB has to be labeled *in*. Note that the contrapositive of justification 3 is precisely the new justification the TMS introduces on backtracking to obtain a revised labeling.

Let J be a set of justifications which are represented as propositional clauses of the form

$$a_1 \wedge \cdots \wedge a_n \wedge not\ b_1 \wedge \cdots \wedge not\ b_m \rightarrow c,$$

where the consequent c is omitted if the justification is a constraint. Let M be a set of propositions occurring in J. Let J'_M be the set of clauses obtained from J by deleting from the antecedents of each clause in J all the expressions $not\ b_i$ such that $b_i \notin M$. Now, each clause occurring in J'_M

$$a_1 \wedge \cdots \wedge a_n \wedge not\ b_1 \wedge \cdots \wedge not\ b_h \rightarrow c$$

can be regarded as the disjunction of literals (propositions or negated propositions)

$$\neg a_1 \vee \cdots \vee \neg a_n \vee b_1 \vee \cdots \vee b_h \vee c.$$

Let J_M be the set of clauses in J'_M having one and only one literal true in M. Then, M is a *generalized stable model* of J if M is a model of J and

$$M = \{p | p \text{ is a proposition in } J \text{ and } J_M \models p\}.$$

Consider the set of justifications in Example 6.2. It has a unique generalized stable model $M = \{no\_fly, abB\}$, which is the same as IN labeling obtained from a TMS. However, for the set of justifications in Example 6.1, there are two generalized stable models $M_1 = \{abA, fly\}$ and $M_2 = \{abB, no\_fly\}$, while the TMS computes only a single labeling corresponding to M_1. Note that the anomalous extension M_2 reappears in this semantics. The authors claimed that the generalized stable model semantics is in accordance with the incremental view of a TMS because the order of justifications given to the TMS will affect the output labeling. However, the unwanted model M_2 is obtained from the augmented set of justifications by a newly added backward justification in the incremental view of a TMS, not from the original set of justifications in the static view of a TMS. Clearly, what we want is the model M_1 obtained from a static view of the TMS, which is indeed the major point Morris claimed to use TMS representation to resolve conflicts in commonsense reasoning. The major problem with this discrepancy is due to the fact that in the generalized stable model semantics, the unidirectional nature (an important property in commonsense reasoning) of a justification is lost.

6.4. Generalized Answer Set Semantics

As noted in Section 6.1, the answer set semantics in extended logic programs cannot deal with conflicting information. In this section, along the line of Giordano and Martelli [42], we present the *generalized answer set semantics*, which is a generalization of the answer set semantics, so that it can also resolve conflicts. The conflict resolution process in our approach is essentially achieved by using the contrapositives of the related clauses and keeping their unidirectional property so as to exclude the anomalous extension of the YSP which reappears in the generalized stable model semantics. The behavior of the generalized answer set semantics corresponds well with the static view of a TMS, which is what we want in commonsense reasoning to resolve conflicts.

Several proposals [31, 92, 104, 103] have been presented to provide TMSs with declarative and logic semantics based on stable models, default logic, and AEL. However, all these proposals cannot handle conflicting information because of the following reason. In a TMS, a well-founded labeling can be inconsistent and thus dependency-directed backtracking is called to resolve this inconsistency. Note that an inconsistent well-founded labeling only contains a pair of complementary propositions, but not all. On the contrary, if an *answer set* in extended logic programs, an *extension* in default logic, or a *stable expansion* in AEL is inconsistent, then it will contain all the sentences by conventional first-order logic. This is one of the difficulties in the major formalizations of nonmonotonic reasoning as well as the answer set semantics in extended logic programs to handle conflicting information. In order to simulate inconsistent well-founded labelings in a TMS, a *relevance* or *contradiction-tolerant* logic (such as in [66]) can be used to allow contradictions in a model for logic programs or a closure for default logic or AEL. Morris [87] pointed out that an inconsistent model in a relevance logic only reveals that some of one's beliefs may be in error, but such a logic does not have the ability to resolve conflicts. In our approach, we will use a relevance logic to allow contradictions in an answer set and use the contrapositives of the related clauses to simulate dependency-directed backtracking to resolve conflicts if the answer set is inconsistent.

Firstly, we modify the definition of an answer set so that contradictions are allowed in an answer set underlying a relevance logic, and this modified definition is called a *relevance answer set*. The definition of a relevance answer set is exactly the same as the definition of an answer set except that the second condition for *not*-free extended logic programs is deleted. That is, if Π is a *not*-free extended logic program, the *relevance answer set* of Π is the smallest subset R of Lit such that for any clause $L_0 \leftarrow L_1, \ldots, L_m$ in Π, if each L_i is in R (for $i = 1, \ldots, m$), then L_0 is also in R. If a relevance answer set R contains a pair of complementary literals, it is not replaced by Lit as in the definition of an answer set.

Let Π be an extended logic program. The relationship between answer sets and relevance answer sets is shown in the following corollary.

Corollary 6.1: Let S be a consistent subset of Lit. Then, S is a relevance answer set of Π if and only if it is an answer set of Π.

Proof: Because S is consistent, it does not contain any pair of complementary literals. Hence, the second condition in the definition of an answer set for *not*-free extended logic programs is not applied to S, and thus the corollary is proved. □

In the relaxed definition, extended logic programs which do not have answer sets can have relevance answer sets. For instance, in Section 6.1, the simplified example formulated in our representation by using extended logic programs does not have answer sets, but it has a unique relevance answer set $R = \{abA, fly, \neg fly\}$, which is precisely the inconsistent labeling obtained from a TMS.

Once a relevance answer set R of Π is obtained, we check whether it is inconsistent. If it is, then the *conflict resolution process* is performed as follows: Let C be the set of all the complementary literals in R. Recall that in Chapter 5, T_s is used to denote the set of strictly true literals derived from Π. If C contains a pair of complementary literals both in T_s, then the contradiction cannot be resolved as in TMSs. Thus, we only consider the case in which each pair of complementary literals are not both strictly true. Recall that dependency-directed backtracking used in a TMS to resolve conflicts is essentially "trace back from the contradiction to find the supporting assumptions which cause inconsistency, and then pick a relevant assumption and put it *out* by making one of its *out*-justifiers *in*." As noted in [42, 92], there exists an analogy between $out(x)$ in TMSs and *not* x in extended logic programs. Thus, an assumption in TMSs corresponds to a defeasible (i.e., true by default) literal in a relevance answer set, and an *out*-justifier in TMSs corresponds to a literal proceeded by *not*. For convenience, we will use the terms "assumption" and "*out*-justifier" in the following discussions. Now, we can simulate dependency-directed backtracking by the following algorithm to compute the set Π_R of clauses:

Algorithm 6.1: *Computing* Π_R

Input: Π, R, T_s, and C.

Output: Π_R.

```
1  $\Pi_R = \Pi$ and $D = C - T_s$;
2  while $D \neq \emptyset$ do
3    for each clause $c : L \leftarrow A_1, \ldots, A_m, not\ B_1, \ldots, not\ B_n$ in Π such that
4        $L \in D$, each $A_i \in R$, and each $B_j \notin R$ do begin
5      for each $A_i \notin T_s$ $(i = 1, \ldots, m)$ do
6        $\overline{A_i} \leftarrow A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_m, not\ B_1, \ldots, not\ B_n, \overline{L}$
7        is added to $\Pi_R$ and $D = D \cup \{A_i\}$;
8      for each $B_j$ $(j = 1, \ldots, n)$ do
9        $B_j \leftarrow A_1, \ldots, A_m, not\ B_1, \ldots, not\ B_{j-1}, not\ B_{j+1}, \ldots, not\ B_n, \overline{L}$
10        is added to $\Pi_R$;
```

```
11      $\Pi = \Pi - \{c\}$ and $D = D - \{L\}$;
12    end_for.
```

In the above algorithm, Π_R is initialized to Π and D is initialized to the difference of C and T_s at line 1. Note that D includes some defeasible literals in R which cause inconsistencies. The related clauses which derive defeasible literals leading to contradictions are chosen at lines 3-4. The supporting defeasible literals which cause contradictions are found and added to D at lines 5-7, and the related contrapositives for assumptions are added to Π_R. The related contrapositives for *out*-justifiers are added to Π_R at lines 8-10, which simulated the step "by making one of its *out*-justifiers *in*." Π and D are reset at line 11 because clause c and literal L are already considered. It is clear that the **while** loop of line 2 simulates the process of dependency-directed backtracking. However, dependency-directed backtracking only chooses a relevant assumption and puts it *out*, whereas the **while** loop in the above algorithm chooses all relevant assumptions and adds a corresponding contrapositive for each of these assumptions.

Theorem 6.1: Algorithm 6.1 will terminate eventually.

Proof: Note that D is a subset of the relevance answer set R and only contains defeasible literals. At each iteration of the **while** loop, let L in D be selected. There must exist clauses in Π whose heads are L such that the required conditions at line 3 are satisfied. Here, only defeasible literals A_i in R are added to D, and then L is deleted from D and the selected clauses are removed from Π. Since R is finite and a chosen defeasible literal in D is deleted after each iteration, D will eventually become empty. □

As we can see from the above algorithm, Π_R includes related contrapositives of the clauses which lead to contradictions. Now, we are ready to define the generalized answer sets of Π such that contradictions are removed and conflicts are resolved.

Definition 6.1: Let R be a relevance answer set of Π. If R is consistent, then it is a *generalized answer set* of Π. Otherwise, if S is a consistent relevance answer set of Π_R, then S is a *generalized answer set* of Π. □

Corollary 6.2: Each consistent answer set of Π is a generalized answer set of Π.

Proof: Let S be a consistent answer set of Π. By Corollary 6.1, S is also a consistent relevance answer set of Π and thus a generalized answer set of Π. □

Since in Algorithm 6.1, all the assumptions which lead to contradictions are chosen and made false by adding appropriate contrapositives to Π_R, each generalized answer set of Π corresponds to a revised consistent well-founded labeling obtained from dependency-directed backtracking. As noted in [86, 119], dependency-directed backtracking can be applied for explanation. It is obvious that the generalized answer set semantics can generate all possible explanations for conflicting information, which is illustrated by the following examples.

Example 6.3: Consider the simplified example formalized by our representation as an extended logic program Π repeated in the following:

c1: $abA \leftarrow not\ abB$,

c2: $\neg fly \leftarrow not\ abA$,

c3: $fly \leftarrow not\ abB$,

c4: $\neg fly \leftarrow$.

It has no answer set, but has a unique relevance answer set $R = \{abA,\ fly,\ \neg fly\}$. Because R contains a pair of complementary literals, fly and $\neg fly$, it is inconsistent. Hence, the conflict resolution process is performed as follows: Π_R is initialized to Π. Since $C = \{fly, \neg fly\}$ and $T_s = \{\neg fly\}$, $D = C - T_s = \{fly\}$. Because the head of clause c3 is in D and abB is not in R, the following contrapositive of c3 is added into Π_R:

c5: $abB \leftarrow \neg fly$.

Then, Π is reset to $\Pi - \{r3\}$, and D is reset to $D - \{fly\} = \emptyset$. Since D becomes empty, the process terminates and Π_R becomes {c1, c2, c3, c4, c5}. Now, Π_R has a unique consistent relevance answer set $S = \{\neg fly,\ abB\}$. By definition, S is a generalized answer set of Π. Note that S is precisely the revised labeling IN obtained from a TMS. An intuitive interpretation for S can be "because Tweety cannot fly ($\neg fly$), it must be an abnormal bird wrt flying (abB)," for both $\neg fly$ and abB are strictly true in Π_R. □

Example 6.4: Consider the following extended logic program Π which is a simplification of the well-known Nixon-diamond problem:

c1: $P \leftarrow not\ abQ$,

c2: $\neg P \leftarrow not\ abR$,

where the propositions P, abQ, and abR stand for "Nixon is a pacifist," "Nixon is an abnormal Quaker," and "Nixon is an abnormal republican," respectively. Similarly, Π has no answer set, but it has a single relevance answer set $R = \{P,\ \neg P\}$. Since R is inconsistent, the conflict resolution process is performed so that the following contrapositives of c1 and c2 are added into Π_R:

c3: $abQ \leftarrow \neg P$,

c4: $abR \leftarrow P$.

Thus, Π_R becomes {c1, c2, c3, c4}, which has two relevance answer sets $S_1 = \{P, abR\}$ and $S_2 = \{\neg P,\ abQ\}$. Because both S_1 and S_2 are consistent, they are generalized answer sets of Π. Note that since $T_s = \emptyset$, P, $\neg P$, abQ, and abR are all true by default; i.e., they are all defeasible conclusions. An intuitive interpretation for S_1 can be "Nixon may be an abnormal republican and thus a pacifist." Similarly, an interpretation for S_2 can be "Nixon may be an abnormal Quaker and thus a non-pacifist." □

Note that the above two interpretations provide explanations to the conflicting information originally obtained in the answer set semantics. Note also that the two relevance answer sets are exactly the same as those obtained from credulous reasoning in ambiguous inheritance hierarchies as discussed in Chapter 3. This seems to suggest that the credulous reasoning in ambiguous inheritance hierarchies can be simulated by the generalized answer set semantics.

Example 6.5: Consider the following diagnostic example taken from [14], which is simplified and formulated by the extended logic program Π as follows:

c1: $appendicitis \leftarrow pain \wedge not\ indigestion \wedge not\ colitis$,

c2: $\neg appendix \leftarrow say\_noappendix \wedge not\ unreliable$,

c3: $appendix \leftarrow appendicitis$,

c4: $say\_noappendix$,

c5: $pain$.

As before, it has no answer set, but has a unique relevance answer set $R = \{pain, say\_noappendix, appendicitis, appendix, \neg appendix\}$. Following the conflict resolution process in our approach, the following four contrapositives are added into Π_R:

c6: $\neg appendicitis \leftarrow \neg appendix$,

c7: $indigestion \leftarrow pain \wedge \neg appendicitis \wedge not\ colitis$,

c8: $colitis \leftarrow pain \wedge not\ indigestion \wedge \neg appendicitis$,

c9: $unreliable \leftarrow say\_noappendix \wedge appendix$.

By definition, Π has three generalized answer sets:

$S_1 = \{say\_noappendix, pain, \neg appendicitis, \neg appendix, indigestion\}$,

$S_2 = \{say\_noappendix, pain, \neg appendicitis, \neg appendix, colitis\}$,

$S_3 = \{say\_noappendix, pain, appendicitis, appendix, unreliable\}$.

Note that $T_s = \{say\_noappendix, pain\}$ for Π_R. Thus, an intuitive interpretation for S_1 can be "a patient who has pain in the right side and says that he has no appendix may have indigestion, but not appendicitis and thus no appendix." Similar interpretations can also be applied for S_2 and S_3. □

The above three examples illustrate how conflicting information can be resolved by the generalized answer set semantics which generates explanations. However, not all conflicts in extended logic programs can be resolved, even if they do not contain any pair of complementary strictly true literals. The following example gives such an unresolvable conflict.

Example 6.6: Consider the following extended logic program Π [34]:

c1: $P \leftarrow not\ \neg P$,

c2: $Q \leftarrow P$,

c3: $\neg Q \leftarrow P$.

It has the only inconsistent relevance set $R = \{P,\ Q,\ \neg Q\}$. After performing the conflict resolution process, Π_R is augmented by the following two contrapositives:

c4: $\neg P \leftarrow \neg Q$,

c5: $\neg P \leftarrow Q$.

Now, the inconsistent R is also the unique relevance answer set of Π_R. Thus, the conflict in this program cannot be resolved, and thus it does not have any generalized answer sets. □

6.5. Conflict Resolution in Other Formalizations

Similar to the answer set semantics, the major formalizations of nonmonotonic reasoning, such as default logic, AEL, and circumscription, illustrated in Section 6.1 also cannot resolve conflicting information. In this section, we briefly discuss by example how the conflict resolution process used in the generalized answer set semantics can also be applied to these formalizations with minor modification.

6.5.1. Generalized Extensions

Consider the default logic representation of the simplified example by using non-normal defaults which is repeated in the following:

$$W = \{\neg fly\},$$

$$D = \left\{ \frac{: M\neg abB}{abA}, \quad \frac{: M\neg abA}{\neg fly}, \quad \frac{: M\neg abB}{fly} \right\}.$$

The default theory (W, D) has no extensions because if $\neg abB$ is assumed, then fly is derived from the third default in D which contradicts the axiom $\neg fly$ and thus abB can be derived from the contradiction by classical logic. If we use a relevance or contradiction-tolerant logic, then the above difficulty can be alleviated and a *relevance extension* $E = \{fly,\ \neg fly,\ abA\}$ is produced. Since E is inconsistent, a conflict resolution process needs to be performed to resolve this contradiction. Recall that in Chapter 5, $thm(W) = \{\neg fly\}$ is used to denote the set of literals which are strictly true as derived from W. In [34], a default of the form

$$\frac{A_1, \ldots, A_m : MB_1, \ldots, MB_n}{L}$$

can be identified by a clause of the form in extended logic programs

$$L \leftarrow A_1, \ldots, A_m, not\ \overline{B_1}, \ldots, not\ \overline{B_n}$$

if each A_i, each B_j, and L are literals, where $\overline{B_j}$ is the complement of B_j as defined in Chapter 5. Thus, the conflict resolution process is performed as follows: Since fly in E is not strictly true and is the consequent of the third default in D, we add the new axiom (roughly, a contrapositive of the default) $\neg fly \rightarrow abB$ to W. Now, the augmented default theory (W, D) has a unique relevance extension $E' = \{\neg fly,\ abB\}$, which agrees with our intuition. Since E' is consistent, it is a *generalized extension* of the original default theory.

6.5.2. Generalized Stable Expansions

Consider the simplified example formulated by the autoepistemic theory A listed below:

1. $\neg L\,abB \rightarrow abA$,
2. $\neg L\,abA \rightarrow \neg fly$,
3. $\neg L\,abB \rightarrow fly$,
4. $\neg fly$.

Similarly, A does not have stable expansions because if abB is not believed, then fly is derived from axiom 3 which contradicts axiom 4 ($\neg fly$) and thus abB is derived from this contradiction by classical logic. If a relevance logic is used, then there is a unique *relevance stable expansion* S which includes abA, fly, and $\neg fly$. Since S is inconsistent, a conflict resolution process needs to be performed. Let A' be the set of axioms obtained from A by removing each axiom including modal operator L. Obviously, $A' = \{\neg fly\}$. Recall that in Chapter 5, $thm(A') = A'$ is used to denote the set of literals which are strictly true as derived from A'. In [86], there is a rough correspondence between $\neg Lx$ in AEL and $out(x)$ in TMSs. Thus, the conflict resolution process for AEL can be performed as follows: Since fly in S is not strictly true and appears in axiom 3, we add the following new axiom (roughly, a contrapositive of axiom 3) to the original autoepistemic theory:

5. $\neg fly \rightarrow abB$.

The intuition of this new axiom is "the non-flying Tweety must be an abnormal bird," for axiom 3 says "if Tweety being an abnormal bird is not believed, then it can fly," but axiom 4 stating "Tweety does not fly" provides an evidence which contradicts our expectation. The augmented autoepistemic theory has a unique relevance stable expansion S' which includes $\neg fly$ and abB. Since S' is consistent, it is a *generalized stable expansion* of the original autoepistemic theory.

6.5.3. Generalized Supported Models

As demonstrated in Section 6.1, supported circumscription [138] is proposed to exclude the anomalous extension of the YSP. In this approach, ***oriented clauses*** are defined to mimic clauses in logic programs. Therefore, an oriented clause is no longer a first-order axiom because it cannot have contrapositives. However, this approach cannot resolve conflicts as mentioned earlier. We repeat the simplified example formulated by the first-order theory T_1 in the following with $P = \{abA, abB\}$ and $Z = \{fly\}$:

1. $\neg abB \rightarrow abA$,
2. $\neg abA \rightarrow \neg fly$,
3. $\neg abB \rightarrow fly$,
4. $\neg fly$.

Note that each axiom in T_1 is an oriented clause in supported circumscription. Here, T_1 has a unique (P, Z)-minimal model $\{abB\}$, but it is not supported because abB is not the head of any oriented clause. In [138], it is pointed out that prioritized circumscription in the same context is actually supported circumscription. But this is not true if conflicting information is present. Indeed, when conflicting information is present, supported circumscription cannot resolve such a conflict, while prioritized circumscription can. In the following, we present a modification of supported circumscription to resolve conflicts, and then sketch how prioritized circumscription can also be used to resolve conflicts in addition to excluding anomalous extensions.

Let T be a set of oriented clauses, and P and Z be as in the definition of circumscription. In order to simulate an inconsistent labeling in circumscription, we can perform renaming as follows: Each negative literal $\neg p(X_1, \ldots, X_k)$ appearing in T is renamed to become a positive literal $no\_p(X_1, \ldots, X_k)$ if $p \notin P$. The renamed theory is called T'. Then, if a supported model of T' is inconsistent (i.e., it contains a pair of complementary literals, say A and no_A), then a similar approach of adding contrapositives as before is performed. The augmented theory is called T''. Now, if a supported model of T'' is consistent, then it is renamed back to be a *generalized supported model* of T.

Consider again the theory T_1 above which has no supported model. The renaming operation only applies to clauses 2 and 4 and obtains

2′. $\neg abA \rightarrow no\_fly$,

4′. no_fly.

There is only one supported model $\{no\_fly,\ fly,\ abA\}$ for the renamed theory T_1'. Since it is inconsistent, the following contrapositive of axiom 3 is added to T_1':

5. $no\_fly \rightarrow abB$.

Let T_1'' be the augmented theory obtained from T_1'. Here, the augmented theory T_1'' has a unique supported model $M_1 = \{no\_fly,\ abB\}$. Since M_1 is consistent, it will be renamed back to $\{abB\}$ which is a generalized supported model of T because abB is supported by clause 5. Note that since each model only contains atoms, the literal no_fly (i.e., $\neg fly$) in M_1 is not included in the generalized supported model.

In the following, we briefly show how prioritized circumscription can also be used to resolve conflicts. Let T be a set of oriented clauses, and P and Z be as in the definition of circumscription. We first perform renaming as before and obtain a renamed theory T'. Since T' is a disjunction theory, if T' is stratified (see [37]), then there exists a stratification of T'. For any stratification of T', we can partition P into $P^1, \ldots, P^n$ such that $P^1 > \cdots > P^n$ (called a *priority ordering*) as in the definition of prioritized circumscription. Let P' be a priority ordering of T'. Then, a (P', Z)-minimal model of T is a model what we want to resolve conflicting information. Again, we examine the previous theory T_1. We have a unique priority ordering P' ($abB > abA$) of T_1' because of axiom 1. There is a unique (P', Z)-minimal model

$M_2 = \{abB\}$ of T_1, which is the same as the generalized supported model obtained earlier.

CHAPTER 7

"UNLESS" SEMANTICS

Some of the earlier formalizations of nonmonotonic reasoning, such as nonmonotonic logic (NML) [73] and default logic [106], and the later ones, such as autoepistemic logic (AEL) [84], are based (partly) on the notion of logical consistency. For example, the commonsense knowledge "birds fly" can be expressed in NML by

$$bird(X) \wedge M\,fly(X) \rightarrow fly(X).$$

This formula can be interpreted as "if X is a bird and it is consistent to assert that X can fly, then X flies." In default logic, this commonsense knowledge is usually represented by the normal default

$$\frac{bird(X) : M\,fly(X)}{fly(X)},$$

which may be read as "if X is a bird and can be consistently assumed to fly, then that X flies can be derived." Similarly, this general rule can be formalized in AEL by

$$bird(X) \wedge \neg L \neg fly(X) \rightarrow fly(X),$$

which is possibly paraphrased as "if X is a bird and it is not believed (known) that X cannot fly, then X flies." Note that in these consistency-based formalizations, the conventional use of logic consistency is either to consistently assume the consequent as in NML and default logic or to consistently assume that the negation of the consequent is not believed as in AEL in order to derive the consequent.

However, the conventional use of consistency to formalize commonsense knowledge in consistency-based formalizations became changed after the *simple abnormality formalism* was presented by McCarthy [71]. The formalism was originally proposed to represent commonsense knowledge in circumscription essentially by introducing ***abnormality predicates***, $ab_i(X)$, to express general rules and by adding ***inheritance cancellation axioms*** to express the preference in inheritance hierarchies. For instance, "birds fly" can be formalized in this formalism by the first-order axiom

$$bird(X) \wedge \neg ab_1(X) \rightarrow fly(X),$$

where $ab_1(X)$ means "X is abnormal in aspect 1 (i.e., flying)." The axiom can be read as "if X is a bird, then it can fly unless it is abnormal in aspect 1." Furthermore, the inheritance cancellation axiom

$$ostrich(X) \rightarrow ab_1(X)$$

is introduced if we also know "ostriches are birds and normally do not fly." This axiom says "ostriches are abnormal in aspect 1." Note that general rules can be interpreted by using "unless," and we call these interpretations "unless" semantics.

Although McCarthy [71] pointed out that this formalism is not adequate to express commonsense knowledge in circumscription, it is widely adopted in other formalizations of nonmonotonic reasoning, especially, to exclude the anomalous extension of the YSP as discussed in Chapter 6. For example, "birds fly" can be represented in default logic by the non-normal default

$$\frac{bird(X) : M\neg ab_1(X)}{fly(X)},$$

which can be interpreted as "if X is a bird, then it can fly unless it is derivable that X is not abnormal in aspect 1." Similarly, this knowledge can be formalized in AEL by

$$bird(X) \wedge \neg L\, ab_1(X) \rightarrow fly(X),$$

which may be read as "if X is a bird, then it can fly unless X being abnormal in aspect 1 is believed (known)." Note that "unless" semantics gradually becomes a new interpretation of commonsense knowledge. Moreover, Kowalski and Sadri [58] argued that the negation implicit in the word "unless" can be interpreted as the negation-as-failure *not* in extended logic programs. In this case, "birds fly" can also be expressed by the clause

$$fly(X) \leftarrow bird(X) \wedge not\ ab_1(X).$$

This is one of the basic concept we adopt to formalize general rules in commonsense reasoning as discussed in Chapter 3.

In spite of the fact that the simple abnormality formalism is commonly used in other formalizations of nonmonotonic reasoning, there are still two important problems encountered in expressing commonsense knowledge. Firstly, these formalizations do not completely capture the idea of "unless" semantics as in the original proposal (i.e., circumscription). For example, in addition to the general rule "birds fly," suppose we also know that Tweety is a bird (i.e., $bird(tweety)$) and is abnormal in aspect 1 (i.e., $ab_1(tweety)$). Then, circumscribing over fly will conclude that Tweety cannot fly (i.e., $\neg fly(tweety)$). Similarly, if the above knowledge is formalized in terms of a TMS, the same conclusion is also drawn. However, all other aforementioned reformulations cannot derive this conclusion. It is apparent that all these formalizations only capture part of the idea of "unless" semantics in circumscription. The second problem is caused by the original simple abnormality formalism itself. The formalism may lead to semantic inconsistencies in the interpretations of conclusions. Consider the previous cancellation axiom. If we happen to know that Sam is a flying ostrich, then we will conclude that Sam can fly, but is abnormal wrt flying (aspect 1), which is apparently inconsistent. In order to cope with these problems, in the following sections, we first modify the inheritance cancellation axioms such that the semantic inconsistency problem is removed. Then, we explore "unless" semantics and endow the formalizations of commonsense reasoning with the full power of this semantics.

7.1. Overview with an Example

In this section, we give an overview of our solutions to the aforementioned two problems with an example. Consider the following commonsense knowledge:

- Elephants are typically gray.
- Royal elephants are elephants but are typically not gray.
- Circus elephants are royal elephants.
- Clyde is a circus elephant.

This knowledge can be formalized by the extended logic program Π_1 as follows:

1. $gray(X) \leftarrow elephant(X) \land not\ ab_1(X)$,
2. $elephant(X) \leftarrow royal\_elephant(X)$,
3. $\neg gray(X) \leftarrow royal\_elephant(X) \land not\ ab_2(X)$,
4. $ab_1(X) \leftarrow royal\_elephant(X)$,
5. $royal\_elephant(X) \leftarrow circus\_elephant(X)$,
6. $circus\_elephant(clyde) \leftarrow$,

where $ab_1(X)$ and $ab_2(X)$ stand for "X is abnormal in aspect 1 (i.e., grayness)" and "X is abnormal in aspect 2 (i.e., non-grayness)," respectively. Note that clause 4 is an inheritance cancellation rule introduced by the simple abnormality formalism. Program Π_1 has a unique answer set

$$S_1 = \{circus\_elephant(clyde), royal\_elephant(clyde), elephant(clyde),$$
$$ab_1(clyde), \neg gray(clyde)\},$$

in which only $\neg gray(clyde)$ is defeasible, while all others are strictly true. The defeasible conclusion $\neg gray(clyde)$ means "Clyde is assumed not gray," which is just a temporary belief. However, since $ab_1(clyde)$ is strictly true, it states that Clyde must be abnormal wrt grayness. Note here that the interpretation of S_1 by using the notion of defeasible conclusion as discussed in Chapter 5 produces a little bit of incoherence due to the sightly different meaning conveyed by defeasible $\neg gray(clyde)$ and strict $ab_1(clyde)$.

If we later come to know that Clyde is really gray, then the following clause is added to the above program Π_1:

7. $gray(clyde) \leftarrow$.

Let Π_2 be the augmented logic program. Now, Π_2 has no answer sets because the conflicting information (clause 7) is learned. However, this conflict can be resolved and then a unique general answer set

$$S_2 = \{circus\_elephant(clyde), royal\_elephant(clyde), elephant(clyde), \\ gray(clyde), ab_1(clyde), ab_2(clyde)\}$$

is obtained. Note that all the literals in S_2 are strictly true. Since $ab_1(clyde)$ and $ab_2(clyde)$ are both strictly true, they state that the gray elephant Clyde is abnormal wrt both grayness and non-grayness. Obviously, this causes semantic inconsistency in the interpretation of S_2.

For coping with this difficulty, in our solution discussed in the following section, we argue that clause 4 should be formalized by

4′. $ab_1(X) \leftarrow royal\_elephant(X) \wedge not\ ab_2(X)$.

Note that the expression $not\ ab_2(X)$ is added to the body of clause 4′ because clause 3 expresses the general rule "royal elephants are typically not gray." Let Π_1' be Π_1 with clause 4 replaced by clause 4′. Program Π_1' has a single answer set S_1' which is the same as S_1 except that $ab_1(clyde)$ now becomes defeasible, rather than strictly true as in Π_1. This means that Clyde is assumed not gray and thus is assumed abnormal wrt grayness. Clearly, this interpretation seems quite reasonable because that Clyde is not gray (i.e., $\neg gray(clyde)$) is only a defeasible conclusion, and that Clyde is abnormal wrt grayness (i.e., $ab_1(clyde)$) is also a temporary belief.

After adding the conflicting fact $gray(clyde)$ to Π'_1, we call the new augmented program Π'_2. Similarly, program Π'_2 has no answer sets, but it has a single generalized answer set S'_2 (which is identical, in this example, to its *extended answer set* defined in Section 7.4) which is almost the same as S_2 except that $ab_1(clyde)$ becomes $\neg ab_1(clyde)$. This means that the gray elephant Clyde is abnormal wrt non-grayness (i.e., $ab_2(clyde)$) but is not abnormal wrt to grayness (i.e., $\neg ab_1(clyde)$). In other words, since Clyde is gray, it must not be abnormal wrt to grayness, which turns out that Clyde must be abnormal wrt non-grayness. It is apparent that this interpretation is more reasonable and the semantic inconsistency problem in the interpretation of S_2 above is removed.

Furthermore, we examine how the extended answer set is obtained as follows: Since clause 7 is a conflicting fact, by the generalized answer set semantics, this conflict is resolved by adding the new ground clause

8. $ab_2(clyde) \leftarrow royal\_elephant(clyde) \wedge gray(clyde)$.

Since $royal\_elephant(clyde)$ and $gray(clyde)$ are both strictly true, $ab_2(clyde)$ is also strictly true. Then, by "unless" semantics, the strictly true $ab_2(clyde)$ causes clauses 3 and 4′ to be transformed into the following two new clauses:

3′. $gray(clyde) \leftarrow royal\_elephant(clyde) \wedge ab_2(clyde)$,

4″. $\neg ab_1(clyde) \leftarrow royal\_elephant(clyde) \wedge ab_2(clyde)$.

Similarly, since $\neg ab_1(clyde)$ becomes strictly true, clause 1 is transformed into

1′. $gray(clyde) \leftarrow elephant(clyde) \wedge \neg ab_1(clyde)$.

As we can see, the given fact $gray(clyde)$ is also derivable from the new clauses 1′ and 3′, while it is not generated by any other clauses in Π'_2 except the given fact itself. Therefore, the transformed program (which contains clauses 1′, 2, 3′, 4″, 5, 6, 7, and 8) not only generates a reasonable and consistent interpretation of its answer set S_2' but also the transformed program itself becomes a coherent set of clauses for the given commonsense knowledge.

7.2. Semantic Inconsistencies in the Abnormality Formalism

Recall that the intuition underlying inheritance hierarchies with exceptions is that subclasses are more specific so that they can incorporate information about exceptional cases; i.e., subclasses are allowed to override superclasses. In the simple abnormality formalism, the introduced inheritance cancellation axioms are essentially used to explicitly express this intuition. However, the formalism may cause semantic inconsistencies in the interpretations of conclusions. We illustrate the semantic inconsistency problem in the original proposal with an example.

Suppose we have the commonsense knowledge:

- Birds normally fly.
- Ostriches normally do not fly.
- Ostriches are birds.

In circumscription, the knowledge can be represented by the first-order theory T_1 with $P = \{ab_1, ab_2\}$ and $Z = \{fly\}$ as follows:

1. $bird(X) \wedge \neg ab_1(X) \rightarrow fly(X)$,
2. $ostrich(X) \wedge \neg ab_2(X) \rightarrow \neg fly(X)$,
3. $ostrich(X) \rightarrow bird(X)$,
4. $ostrich(X) \rightarrow ab_1(X)$,

where $ab_1(X)$ and $ab_2(X)$ stand for "X is abnormal in aspect 1 (i.e., flying)" and "X is abnormal in aspect 2 (i.e., non-flying)," respectively. Note that the last axiom is an inheritance cancellation axiom introduced by the abnormality formalism, which means that all ostriches are abnormal wrt flying. However, if we happen to know that

Sam is an ostrich (i.e., $ostrich(sam)$) and can fly (i.e., $fly(sam)$) then circumscribing over ab_1 and ab_2 by varying fly will obtain a unique minimal model

$$M_1 = \{ostrich(sam), bird(sam), fly(sam), ab_1(sam), ab_2(sam)\}.$$

The model says that the flying ostrich Sam is a bird and is abnormal wrt both flying (aspect 1) and non-flying (aspect 2), which is obviously inconsistent. Since the conclusion $ab_2(sam)$ (Sam is abnormal wrt non-flying) agrees with the given fact $fly(sam)$, the semantic inconsistency is indeed caused by $ab_1(sam)$ (Sam is abnormal wrt flying) which is derived from the inheritance cancellation axiom.

With careful examination of the above example, one may notice that $fly(sam)$ is a conflicting information because it contradicts what we expect from the general rule that ostriches normally fly, and M_1 is obtained from conflict resolution discussed in Chapter 6. Thus, it may be argued that we can create a subclass FLYING_OSTRICH and express the above commonsense knowledge with the following axioms added to T_1:

5. $flying\_ostrich(X) \rightarrow ostrich(X)$,
6. $flying\_ostrich(X) \rightarrow fly(X)$,
7. $flying\_ostrich(X) \rightarrow ab_2(X)$.

Let T_2 be the augmented theory. With the fact $flying\_ostrich(sam)$, we still obtain from T_2 the same conclusion as that obtained from T_1, and thus the semantic inconsistency problem remains unsolved. Note that creating an exceptional subclass to express conflicting information as in the previous example provides another way for conflict resolution.

From the above reformalization, we can see that the semantic inconsistency problem is independent of the way how the knowledge is formalized. In fact, it is caused by the improper formalization of inheritance cancellation axioms introduced by the abnormality formalism. To see why, we carefully examine the above theory

T_1. Since we know that Sam is an ostrich and ostriches are abnormal wrt flying (aspect 1) by axiom 4, Sam must be abnormal wrt flying which conflicts the given fact that Sam can fly. However, since $fly(sam)$ is a given fact, it should be more credible than the conclusion $ab_1(sam)$ obtained from axiom 4. In fact, not every ostrich is abnormal wrt flying because there may exist some flying ostriches, such as Sam in the above example. Therefore, the added inheritance cancellation axiom (axiom 4) does not reflect this exception and thus causes the semantic inconsistency problem. In response to this difficulty, we use the following modified axiom to replace axiom 4:

4′. $ostrich(X) \wedge \neg ab_2(X) \rightarrow ab_1(X)$,

which means that if X is an ostrich and is not abnormal wrt non-flying (aspect 2), then it is abnormal wrt flying (aspect 1). The above new inheritance cancellation axiom provides a consistent meaning because flying ostriches must be abnormal wrt non-flying which does not allow us to derive that they are also abnormal wrt flying as in axiom 4. Let T_1' contain axioms 1, 2, 3, and 4′. Then, circumscribing over ab_1 and ab_2 by varying fly will obtain a unique minimal model M_2 which is almost the same as M_1 except that $ab_1(sam)$ is removed. Obviously, M_2 is a consistent conclusion. Similarly, for theory T_2, with axiom 4 replaced by axiom 4′, we still obtain the same consistent conclusion as that obtained from T_1'. Thus, the semantic inconsistency problem is resolved by the modified inheritance cancellation axioms.

Recall that in Chapter 3, we have defined a class R being abnormal in aspect i in Definitions 3.6 and 3.7, which is expressed by the axiom

$r(X) \rightarrow ab_i(X)$

based on the abnormality formalism. Now, according to the above discussion, we can modify the expressions for abnormalities as follows: If the link in the first condition of Definitions 3.6 and 3.7 is strict, then class R being abnormal in aspect i is expressed by the above axiom. Otherwise, if the link is default with label ab_j, then class R being abnormal in aspect i should be expressed by the axiom

$$r(X) \wedge ab_j(X) \rightarrow ab_i(X).$$

7.3. Exploring "Unless" Semantics

In [58], it was argued that the negation-as-failure *not* can be used to express the negation implicit in the word "unless" in commonsense reasoning. For example, the flying bird example can be expressed by the following clause:

$$fly(X) \leftarrow bird(X) \wedge not\ ab_1(X),$$

which may be read as "if X is a bird, then it can fly unless it is abnormal in aspect 1 (i.e., flying)." The clause implicitly implies "if X is a bird and is known to be abnormal in aspect 1, then it cannot fly." However, the answer set semantics cannot derive this conclusion $\neg fly(tweety)$ if we know $bird(tweety)$ and $ab_1(tweety)$. But in the original formalism, the conclusion $\neg fly(tweety)$ is derivable from circumscription with $P = \{fly\}$. In addition, if the above knowledge is formalized in terms of a TMS, we will obtain the same conclusion. It seems that the answer set semantics does not capture the full idea of "unless" semantics in the original abnormality formalism. In the following, we examine "unless" semantics in the convention of commonsense reasoning by using extended logic programs for illustration.

We first look at the simple commonsense knowledge as follows:

(A) *John is not going to the movie unless Mary is.*

Using *not* to represent "unless," knowledge (A) can be expressed by the clause

A1. $\neg go(john) \leftarrow not\ go(mary),$

where predicate $go(X)$ stands for "X is going to the movie." Now, consider the following cases:

(i) If we know that Mary is not going to the movie (i.e., $\neg go(mary)$), then so is not John. This statement can be expressed by the clause

A2. $\neg go(john) \leftarrow \neg go(mary)$,

and the conclusion $\neg go(john)$ is then derived from the given fact $\neg go(mary)$. Note that the above clause can be transformed from clause A1 since $go(mary)$ is strictly false.

(ii) If we know that Mary is going to the movie (i.e., $go(mary)$), then so is John. This statement can be expressed by the clause

A3. $go(john) \leftarrow go(mary)$,

and the conclusion $go(john)$ is then derived from the given fact $go(mary)$. Note that clause A3 can be transformed from clause A1 since $go(mary)$ is strictly true.

(iii) If we have no idea about whether or not Mary is going to the movie, then we may assume by default that John is not going to the movie. The default conclusion $\neg go(john)$ is precisely obtained from the answer set of clause A1 because $go(mary)$ is *unknown*.

(iv) If we know that John is going to the movie (i.e., $go(john)$), then so is Mary. This statement can be expressed by the clause

A4. $go(mary) \leftarrow go(john)$,

and the conclusion $go(mary)$ is then derived from the given fact $go(john)$. Note that clause A4 is obtained from clause A2 by contraposition since $go(john)$ is strictly true.

(v) If we know that John is not going to the movie (i.e., $\neg go(john)$), then so is not Mary. This statement can be expressed by the clause

A5. $\neg go(mary) \leftarrow \neg go(john)$,

and the conclusion $\neg go(mary)$ is then derived from the given fact $\neg go(john)$. Note that clause A5 is obtained from clause A3 by contraposition since $go(john)$ is strictly false.

In the above example, cases (i) and (ii) roughly correspond to *deductive reasoning* in AI because they deduce the conclusions from the given information. Case (iii) corresponds to *commonsense reasoning* because of the presence of incomplete information. And cases (iv) and (v) roughly correspond to *abductive reasoning* because they provide plausible explanations to the known conclusions.

We examine a more complex example as follows:

(B) *John is not going to the movie unless Mary and Jim are.*

This knowledge can be formalized by the clauses

B1. $\neg go(john) \leftarrow not\ both(mary, jim)$,

B2. $both(mary, jim) \leftarrow go(mary) \wedge go(jim)$,

B3. $\neg both(mary, jim) \leftarrow \neg go(mary)$,

B4. $\neg both(mary, jim) \leftarrow \neg go(jim)$.

Note that clauses B3 and B4 can be combined into a single clause as follows if disjunctions are allowed in the bodies of clauses:

B5. $\neg both(mary, jim) \leftarrow \neg go(mary) \vee \neg go(jim)$.

Now, consider the following cases:

(i) If we know that Mary or Jim is not going to the movie (i.e., $\neg both(mary, jim)$), then so is not John. This can be formulated by the clause

B6. $\neg go(john) \leftarrow \neg both(mary, jim)$, (transformed from B1)

and the conclusion $\neg go(john)$ is then derived from the given fact.

(ii) If we know that both Mary and Jim are going to the movie (i.e., $both(mary, jim)$), then so is John. This can be expressed by the clause

B7. $go(john) \leftarrow both(mary, jim)$, (transformed from B1)

and the conclusion $go(john)$ is then derived from the given fact.

(iii) If we have no idea about whether Mary or Jim is going to the movie, then we may assume by default that John is not going to the movie. The defeasible conclusion

$\neg go(john)$ is derived from clause B1. Note that the above three cases are similar to those of knowledge (A).

(iv) If we know that John is going to the movie (i.e., $go(john)$), then so are Mary and Jim. This can be expressed by the clause

B8. $go(mary) \land go(jim) \leftarrow go(john)$,

which is derived by transitivity from the following two clauses:

B9. $both(mary, jim) \leftarrow go(john)$,

B10. $go(mary) \land go(jim) \leftarrow both(mary, jim)$.

Note that clauses B9 and B10 are, respectively, obtained from clauses B6 and B5 by contraposition since $go(john)$ is strictly true and so is $both(mary, jim)$. Note also that clauses B8 and B10 are obtained by allowing conjunctions in the heads of clauses.

(v) If we know that John is not going to the movie (i.e., $\neg go(john)$), then so is not Mary or Jim. This can be expressed by the clause

B11. $\neg go(mary) \lor \neg go(jim) \leftarrow \neg go(john)$,

which is derived by transitivity from the following two clauses:

B12. $\neg both(mary, jim) \leftarrow \neg go(john)$,

B13. $\neg go(mary) \lor \neg go(jim) \leftarrow \neg both(mary, jim)$.

Note that clauses B12 and B13 are, respectively, obtained from clauses B7 and B2 by contraposition since $go(john)$ is strictly false and so is $both(mary, jim)$. Note also that clauses B11 and B13 are obtained by allowing disjunctions in the heads of clauses.

Now, we examine the last example as follows:

(C) *John is not going to the movie unless Mary or Jim is.*

This can be represented by the clause

C1. $\neg go(john) \leftarrow not\ go(mary) \land not\ go(jim)$.

Clause C1 can also be represented by the following clauses such that the general rule clause C2 is similar to clause A1 of knowledge (A):

C2. $\neg go(john) \leftarrow not\ or(mary, jim)$,

C3. $or(mary, jim) \leftarrow go(mary) \vee go(jim)$.

C4. $\neg or(mary, jim) \leftarrow \neg go(mary) \wedge \neg go(jim)$.

Consider the following cases:

(i) If we know that both Mary and Jim are not going to the movie (i.e., $\neg or(mary, jim)$), then so is not John. This can be expressed by the clause

C5. $\neg go(john) \leftarrow \neg or(mary, jim)$. (transformed from C2)

(ii) If we know that Mary or Jim is going to the movie (i.e., $or(mary, jim)$), then so is John. This can be expressed by the clause

C6. $go(john) \leftarrow or(mary, jim)$. (transformed from C2)

(iii) If we have no idea about whether or not Mary or Jim is going to the movie, then we may assume by default that John is not going to the movie.

(iv) If we know that John is going to the movie (i.e., $go(john)$), then so is Mary or Jim. This can be expressed by the clause

C7. $go(mary) \vee go(jim) \leftarrow go(john)$,

which is derived by transitivity from the following two clauses:

C8. $or(mary, jim) \leftarrow go(john)$, (obtained from C5)

C9. $go(mary) \vee go(jim) \leftarrow or(mary, jim)$. (obtained from C4)

(v) If we know that John is not going to the movie (i.e., $\neg go(john)$), then so are Mary and Jim. This can be expressed by the clause

C10. $\neg go(mary) \wedge \neg go(jim) \leftarrow \neg go(john)$,

which is derived by transitivity from the following two clauses:

C11. $\neg or(mary, jim) \leftarrow \neg go(john)$, (obtained from C6)

C12. $\neg go(mary) \wedge \neg go(jim) \leftarrow \neg or(mary, jim)$. (obtained from C3)

Note that the above five cases are similar to those of knowledge (B).

7.4. Formalizing Unless Semantics in Extended Logic Programs

As we can see from the discussions of the previous section, clauses A1, B1, and C2 play an important role of the reasoning. Each of them can be transformed into one of other four different forms depending on the given information. A clause, such as A1, of the form

$L \leftarrow A \wedge not\ B$

is called a *canonical* clause where L, A, and B are literals. It is noted from knowledge (C) in the previous section that any ground clause that follows containing *not* can be represented by a canonical clause by the following transformation:

1. $L \leftarrow A_1 \wedge \cdots \wedge A_m \wedge not\ B_1 \wedge \cdots \wedge not\ B_n \qquad (m \geq 0,\ n \geq 1)$

where each A_i, each B_j, and L are literals. Let us define three new clauses

2. $A \leftarrow A_1 \wedge \cdots \wedge A_m$,
3. $B \leftarrow B_1 \vee \cdots \vee B_n$,
4. $\overline{B} \leftarrow \overline{B_1} \wedge \cdots \wedge \overline{B_n}$,

where A and B are new literals, where $\overline{B_j}$ is the complement of B_j as defined in Chapter 5. Then, clause 1 can be expressed by

5. $L \leftarrow A \wedge not\ B$.

Note that clause 4 is required and essential in this transformation because it provides important information for "unless" semantics.

Once a canonical clause is obtained, the "unless" semantics discussed in the previous section can be formalized as follows: Suppose we have a canonical clause 5, which is repeated in the following with a possible interpretation parenthesized after it,

5. $L \leftarrow A \wedge not\ B$ (if A holds, then conclude L unless B is derivable),

and we know that A is strictly true. Then, the above canonical clause can be transformed into

6. $L \leftarrow A \wedge \overline{B}$ if B is strictly false,

7. $\overline{L} \leftarrow A \wedge B$ if B is strictly true,

8. $B \leftarrow A \wedge \overline{L}$ if L is strictly false,

9. $\overline{B} \leftarrow A \wedge L$ if L is strictly true.

Note that clauses 6 and 7 deduce strict conclusions given strict evidence of the conditions, whereas clauses 8 and 9 provide plausible explanations to the known conclusions. Note also that clauses 8 and 9 are essentially contrapositives of clauses 6 and 7, respectively. In the above transformation, clause 7 may also be transformed into

7′. $\overline{L} \leftarrow B$ if B is strictly true,

where the truth value of A is immaterial in this situation. For example, consider the commonsense knowledge "If Henry has money, then he will go to the movies unless he is busy." If we know that he is busy, then we will conclude that he is not going to the movies, no matter whether or not he has money. In our discussion, we will use the transformation of clause 7 instead of clause 7′ for uniformity.

Now, we can incorporate "unless" semantics into the generalized answer set semantics so that the resulting conclusions are more reasonable and the transformed logic programs are more coherent. Let Π be an extended logic program. Let Π' be Π if Π does not contain conflict information; otherwise, let Π' be Π_R which is an augmented program produced from the conflict resolution process as discussed in Chapter 6. Then, the process of incorporating "unless" semantics consists of three phases. Phase 1 transforms each clause containing *not* in Π' into a canonical clause as discussed above. Let Σ be the transformed program obtained from Π'. Phase 2 performs the transformation of the canonical clauses in Σ such that they can deduce strict conclusions. This phase is described in Algorithm 7.1 and the transformed program is called Σ'. Phase 3 performs the transformation of the canonical clauses in

Σ' such that they can generate plausible explanations, which is described in Algorithm 7.2, and the transformed program is called Σ''.

Algorithm 7.1: *Transformation for deducing strict conclusions*

Input: Σ.

Output: Σ'.

```
1  $T = \alpha(\Sigma^+)$ and $\Sigma' = \Sigma$;
2  while there exists a canonical clause $c : L \leftarrow A \wedge not\ B$ in $\Sigma'$ such that
3        $A \in T$ and ($B \in T$ or $\overline{B} \in T$) do begin
4     if $\overline{B} \in T$ then $L \leftarrow A \wedge \overline{B}$ is added to $\Sigma'$ and $T = T \cup \{L\}$;
5     if $B \in T$ then $\overline{L} \leftarrow A \wedge B$ is added to $\Sigma'$ and $T = T \cup \{\overline{L}\}$;
6     $\Sigma' = \Sigma' - \{c\}$;
7  end_while.
```

Algorithm 7.2: *Transformation for generating plausible explanations*

Input: Σ'.

Output: Σ''.

```
1  $T = \alpha(\Sigma'^+)$ and $\Sigma'' = \Sigma'$;
2  while there exists a canonical clause $c : L \leftarrow A \wedge not\ B$ in $\Sigma''$ such that
3        $A \in T$ and $B \notin T$ and ($L \in T$ or $\overline{L} \in T$) do begin
4     if $\overline{L} \in T$ then $B \leftarrow A \wedge \overline{L}$ is added to $\Sigma''$ and $T = T \cup \{B\}$;
5     if $L \in T$ then $\overline{B} \leftarrow A \wedge L$ is added to $\Sigma''$ and $T = T \cup \{\overline{B}\}$;
6     $\Sigma'' = \Sigma'' - \{c\}$;
7  end_while.
```

In Algorithm 7.1, T is initialized to the set of literals which are strictly true in Σ, and Σ' is initialized to Σ at line 1. The **while** loop of line 2 chooses a canonical clause in Σ' such that it can be transformed to deduce a strict conclusion. The transformation of clauses 6 and 7 is performed at lines 4 and 5, respectively. The selected clause is then deleted from Σ' at line 6 because each clause is considered only once. The explanation for Algorithm 7.2 is similar to that for Algorithm 7.1 except that it is a transformation for generating plausible explanations. Note that the additional constraint $B \notin T$ of line 3 in Algorithm 7.2 is to eliminate the repetition of transformation. The reason for separating into two algorithms and placing Algorithm 7.2 after Algorithm 7.1 is that generating plausible explanations is less important.

Once a transformed logic program Σ'' is obtained from a given extended logic program Π, it is ready to define extended answer sets of Π.

Definition 7.1: Let Π be an extended logic program, and Σ'' be an output of Algorithm 7.2. Let E be a consistent subset of Lit. Then, E is an *extended answer set* of Π if E is an answer set of Σ''. □

In the following examples, we show how the extended answer set semantics can provide more reasonable conclusions for given commonsense knowledge.

Example 7.1: Consider the commonsense knowledge "birds normally fly and Tweety is a bird," which can be formalized by the extended logic program Π_1 as follows:

1. $fly(X) \leftarrow bird(X) \wedge not\ ab_1(X)$,
2. $bird(tweety) \leftarrow$,

where $ab_1(X)$ stands for "X is abnormal wrt flying." Π_1 has a single answer set $S_1 = \{bird(tweety), fly(tweety)\}$, in which $fly(tweety)$ is defeasible. S_1 concludes that the bird Tweety is assumed to fly, which corresponds with our intuition in commonsense reasoning. Now, consider the following four cases:

Case 1: If we later come to know that Tweety is really abnormal wrt flying (i.e., $ab_1(tweety)$), then we add the clause

3. $ab_1(tweety) \leftarrow$

into Π_1. Let Π_2 be Π_1 augmented by clause 3. Program Π_2 has a single answer set S_2 $= \{bird(tweety), ab_1(tweety)\}$, which just states that the bird Tweety is abnormal wrt flying. However, since $ab_1(tweety)$ is strictly true, by "unless" semantics, a ground instance of clause 1 is transformed into

1′. $\neg fly(tweety) \leftarrow bird(tweety) \wedge ab_1(tweety)$.

Now, the transformed program (which contains clauses 1′, 2, and 3) has a single extended answer set S_2', which is S_2 augmented by a strictly true $\neg fly(tweety)$. The extended answer set S_2' concludes that the abnormal bird Tweety cannot fly, which makes perfect sense because we already know Tweety is really abnormal wrt flying.

Case 2: If we happen to know that Tweety is not abnormal (i.e., normal) wrt flying (i.e., $\neg ab_1(tweety)$), then we add the clause

4. $\neg ab_1(tweety) \leftarrow$

into Π_1. Let Π_3 be Π_1 augmented by clause 4. Here, program Π_3 has a unique answer set $S_3 = \{bird(tweety), \neg ab_1(tweety), fly(tweety)\}$, in which $fly(tweety)$ is defeasible, whereas the other two are strictly true. This means that the normal bird Tweety can be assumed to fly. However, since $ab_1(tweety)$ is strictly false in Π_3, by "unless" semantics, a ground instance of clause 1 is transformed into

1″. $fly(tweety) \leftarrow bird(tweety) \wedge \neg ab_1(tweety)$.

The transformed program (containing clauses 1″, 2, and 4) has a single extended answer set S_3' which is exactly the same as S_3 except that $fly(tweety)$ now becomes strictly true. This means that the normal bird Tweety does fly. Note that the above conclusion is stronger than S_3, which is reasonable because we already know that Tweety is a normal bird in Π_3.

Case 3: If we are told that Tweety can fly, then the clause

5. $fly(tweety) \leftarrow$

is added into Π_1 and a new program Π_4 is obtained. Program Π_4 has an answer set S_4

$= \{bird(tweety), fly(tweety)\}$, which concludes that the bird Tweety flies. However, since $fly(tweety)$ is strictly true, by "unless" semantics, a ground instance of clause 1 is transformed into

1‴. $\neg ab_1(tweety) \leftarrow bird(tweety) \wedge fly(tweety)$.

Now, the transformed program has an extended answer set S_4', which is S_4 augmented by a strictly true $\neg ab_1(tweety)$. S_4' concludes that the flying bird Tweety is not abnormal wrt flying. Note here that the added literal $\neg ab_1(tweety)$ provides a plausible explanation for the fact $fly(tweety)$, which is not in S_4.

Case 4: If we observed that Tweety cannot fly, then the clause

6. $\neg fly(tweety) \leftarrow$

is added into Π_1. Let Π_5 be the augmented program. Program Π_5 has no answer set because the conflicting fact (clause 6) is obtained. However, this conflict can be resolved by the conflict resolution process by adding the ground clause

7. $ab_1(tweety) \leftarrow bird(tweety) \wedge \neg fly(tweety)$

into Π_4 and a generalized answer set $S_5 = \{bird(tweety), \neg fly(tweety), ab_1(tweety)\}$ is generated, in which all literals are strictly true. S_5 concludes that the non-flying bird Tweety is abnormal wrt flying. Note that in this case, the extended answer set of Π_5 is identical to S_5 because clause 7 is already added to resolve conflict. □

Example 7.2: Consider the the extended logic program Π in the following:

1. $\neg fly(X) \leftarrow not\ ab_1(X)$,
2. $ab_1(X) \leftarrow bird(X) \wedge not\ ab_2(X)$,
3. $fly(X) \leftarrow bird(X) \wedge not\ ab_2(X)$,
4. $bird(X) \leftarrow canary(X) \wedge not\ ab_3(X)$,
5. $canary(henry) \leftarrow$,
6. $\neg fly(henry) \leftarrow$.

where predicates $ab_1(X)$, $ab_2(X)$, and $ab_3(X)$ stand for "X is abnormal wrt non-flying," "X is abnormal wrt flying," and "X is abnormal wrt a bird," respectively. Since clause 6 is a conflicting fact, program Π has no answer set. However, this conflict can be resolved by the conflict resolution process by adding the following three new ground clauses into Π:

7. $ab_2(henry) \leftarrow bird(henry) \wedge \neg fly(henry)$,
8. $\neg bird(henry) \leftarrow \neg fly(henry) \wedge not\ ab_2(henry)$,
9. $ab_3(henry) \leftarrow canary(henry) \wedge \neg bird(henry)$.

Let Π_R be the augmented program. Now, program Π_R has two generalized answer sets:

$S_1 = \{canary(henry), \neg fly(henry), bird(henry), ab_2(henry)\}$,

$S_2 = \{canary(henry), \neg fly(henry), \neg bird(henry), ab_3(henry)\}$.

Note that only $canary(henry)$ and $\neg fly(henry)$ are strictly true in S_1 and S_2. Then, S_1 can be interpreted as "the non-flying canary Henry may be an abnormal bird wrt flying." Similarly, S_2 can be interpreted as "the non-flying canary Henry may not be a bird and thus is abnormal wrt a bird."

Since $\neg fly(tweety)$ is strictly false, by "unless" semantics, a ground instance of clause 1 is transformed into

1'. $\neg ab_1(henry) \leftarrow \neg fly(henry)$.

Now, the transformed program of Π also has two extended answer sets, which are almost the same as the generalized answer sets except that the strictly true conclusion $\neg ab_1(henry)$ is also included. Note that, the added literal $\neg ab_1(henry)$ (Henry is not abnormal wrt non-flying) provides a plausible explanation for the non-flying canary Henry, which is not in the generalized answer sets. □

7.5. "Unless" Semantics in Default Logic and Autoepistemic Logic

In this section, we briefly discuss by example that "unless" semantics can easily be applied to default logic and autoepistemic logic if they use the simple abnormality formalism to represent commonsense knowledge. It is also shown that the conventional use of logic consistency in formalizing commonsense knowledge works well with "unless" semantics.

Consider the commonsense knowledge "professors are normally doctors." This can be represented in default logic and AEL, respectively, by

$$\frac{prof(X) : M\neg ab_1(X)}{phd(X)}$$

and

$$prof(X) \wedge \neg L\, ab_1(X) \rightarrow phd(X)$$

if the abnormal formalism is used, in which predicates $prof(X)$, $phd(X)$, and $ab_1(X)$ stand for "X is a professor," "X is a doctor," and "X is an exceptional professor to be a doctor," respectively. Suppose we know that John is a professor (i.e., $prof(john)$). Then, by "unless" semantics, both the ground non-normal default

$$\frac{prof(john) : M\neg ab_1(john)}{phd(john)}$$

and the ground autoepistemic axiom

$$prof(john) \wedge \neg L\, ab_1(john) \rightarrow phd(john)$$

can be transformed into

1. $prof(john) \wedge \neg ab_1(john) \rightarrow phd(john)$ if $ab_1(john)$ is strictly false,
2. $prof(john) \wedge ab_1(john) \rightarrow \neg phd(john)$ if $ab_1(john)$ is strictly true,
3. $prof(john) \wedge \neg phd(john) \rightarrow ab_1(john)$ if $phd(john)$ is strictly false,
4. $prof(john) \wedge phd(john) \rightarrow \neg ab_1(john)$ if $phd(john)$ is strictly true.

Note that as in the extended answer set semantics, axioms 1 and 2 deduce strict conclusions, whereas axioms 3 and 4 generate plausible explanations. Note also that axioms 3 and 4 are contrapositives of axioms 1 and 2, respectively.

If the above commonsense knowledge is formalized by the conventional use of logic consistency, then it can be expressed in default logic and AEL, respectively, by

$$\frac{prof(X) : M phd(X)}{phd(X)}$$

and

$$prof(X) \wedge \neg L \neg phd(X) \rightarrow phd(X).$$

Given the fact that John is a professor, by "unless" semantics, both the ground normal default

$$\frac{prof(john) : M phd(john)}{phd(john)}$$

and the ground autoepistemic axiom

$$prof(john) \wedge \neg L \neg phd(john) \rightarrow phd(john)$$

can be transformed into

5. $prof(john) \wedge phd(john) \rightarrow phd(john)$ if $phd(john)$ is strictly true,

6. $prof(john) \wedge \neg phd(john) \rightarrow \neg phd(john)$ if $phd(john)$ is strictly false.

Note that the transformation of deducing strict conclusions is identical to that of generating plausible explanations in this case. From the above illustration, the notion of logic consistency seems to work quite well with the "unless" semantics.

CHAPTER 8

COMPUTING ANSWER SETS

As noted in previous chapters, extended logic programs with an appropriate answer set semantics are a powerful knowledge representation tool. They are successfully employed in the following aspects of commonsense reasoning:

- Formalizing inheritance hierarchies with exceptions.
- Excluding anomalous extensions by the unidirectional nature of clauses.
- Identifying defeasible conclusions by their simple syntax.
- Resolving conflicts by the augmented conflict resolution mechanism in the generalized answer set semantics.
- Producing more reasonable conclusions and more coherent extended logic programs to represent given commonsense knowledge by incorporating "unless" semantics in the extended answer set semantics.

However, the definition of answer sets is based on a non-constructive fixed point definition. One possible way to find all answer sets of an extended logic program Π is to check each subset of *Lit* of Π to see whether it is an answer set by the fixed point definition. Nevertheless, this naive approach is not feasible because the number of subsets is exponential.

In [34], Gelfond and Lifschitz presented a renaming procedure to transform an extended logic program Π into a general logic program Π^* and then showed that for a consistent set $S \subsetneq Lit$, S is an answer set of Π if and only if the renamed S (i.e., S^*) is an answer set of Π^*. This is because extended logic programs are an extension of general logic programs formed by incorporating classical negation. In fact, for a general logic program P, M is an answer set of P if and only if M is a stable model [33] of P because the definition of answer sets is also an extension of that of stable models. The above renaming procedure seems to suggest a method of computing answer sets of an extended logic program Π by computing the stable models of the renamed general logic program Π^*. Indeed, this is the basic idea of the algorithm of computing answer sets presented in this chapter.

In the literature, there are two algorithms of computing stable models of general logic programs. Pimentel and Cuadrado [92] proposed a new TMS based on stable models which are computed directly by employing a label propagating algorithm and the data structure called *compressible semantic tree* [111]. Eshghi [25] recently proposed a method of computing stable models by using the assumption-based TMSs [20]. In this chapter, we present a new approach to computing stable models based on the notion of stratification [3, 129]. The basic idea of our method is to find a minimal set C of rules in a general logic program P such that $P - C$ becomes stratified. There are five steps in our algorithm: First, construct the dependency graph G of a general logic program P. Second, simplify G such that the simplified dependency graph G' contains only strongly connective components; i.e., the value of each node in G' completely depends on the values of the nodes incident to it. Third, find all the loops with even and odd number of negative arcs (which are called *even* and *odd* negative loops, respectively) in G'. Then, find the collection of minimal sets (each of these sets is called a *cut*) of rules in P' containing *not*, which appear in the even negative loops. Finally, for each cut, form a stratified logic program and check its iterated fixed point model [3] to see whether the model is a stable model of P. With the above renaming procedure, an algorithm of computing answer sets of extended logic programs based on computing stable models is then presented. Since a default can be represented by a clause, it seems to suggest that the algorithm of computing answer sets can also be used to compute extensions of default theories. Similarly, since answer sets of extended logic programs are closely connected to stable expansions of autoepistemic theories, our algorithm also can be used to compute stable expansions of autoepistemic theories.

8.1. Reduction to General Programs

Before presenting the renaming procedure of transforming extended logic programs into general logic programs proposed in [34], we briefly review the definition of stable models of general logic programs. A general logic program P is a set of clauses of the form

$$H \leftarrow A_1, \ldots, A_m, not\ B_1, \ldots, not\ B_n$$[1],

where H, each A_i, and each B_j are atoms. As assumed in [33], each clause containing variables is replaced by all its ground instances so that all atoms in P are ground. Let I be an interpretation of P. Let P_I be the logic program obtained from P by deleting

(i) each clause containing the expression $not\ B$ in the body with $B \in I$, and

(ii) all the expressions $not\ B$ in the bodies of the remaining clauses.

Since P_I is *not*-free (i.e., a definite logic program), it has a unique minimal Herbrand model [67]. If this model is identical to I, then I is said to be a *stable model* of P. The following theorem shows that for general logic programs, the definition of answer sets reduces to that of stable models.

[1]In general logic programs, the negation-as-failure is usually represented by the notation "$\neg$." For the consistent use of the notations in this dissertation, *not* is used to represent the negation-as-failure in general logic programs, rather than the conventional $\neg$. (The notation "$\neg$" is used to denote the classical negation in this dissertation.)

Theorem 8.1: Let M be an interpretation of a general logic program P. Then, M is an answer set of P if and only if M is a stable model of P.

Proof: Note that, by definition, the procedure of computing P_I for a general logic program P is essentially the same as that of computing Π^S for an extended logic program Π, where I is an interpretation of P and S is a subset of Lit of Π. Since P is a general logic program, $P_M = P^M$, which is *not*-free. Because P contains atoms only (the expression *not* A is not considered as a negative literal, where A is an atom), the answer set of P^M is the same as the minimal Herbrand model of P_M, and thus the theorem is proved. □

In [34], Gelfond and Lifschitz have shown that an extended logic program can be transformed into a general logic program by the following renaming procedure: Let Π be an extended program. For any predicate symbol p occurring in Π, let no_p be a new predicate symbol of the same arity. The atom $no\_p(\ldots)$ is called the *positive form* of the negative literal $\neg p(\ldots)$. By definition, each positive literal is its own positive form. The positive form of a literal L is denoted by L^*. Let Π^* stand for the general program obtained from Π by replacing each clause

$$L_0 \leftarrow A_1, \ldots, A_m, not\ B_1, \ldots, not\ B_n$$

with

$$L_0^* \leftarrow A_1^*, \ldots, A_m^*, not\ B_1^*, \ldots, not\ B_n^*.$$

For any set $S \subset Lit$, S^* stands for the set of the positive forms of the elements of S. Then, Gelfond and Lifschitz [34] provided the relationship between an extended logic program Π and its renamed program Π^* by the following theorem:

Theorem 8.2 (from [34]): A consistent set $S \subset Lit$ is an answer set of Π if and only if S^* is an answer set of Π^*. □

Recall that in Chapter 6, for resolving conflicting information in extended logic programs, the definition of relevance answer sets is provided to allow contradictions.

The following corollary relaxes the constraint of the consistency condition in Theorem 8.2.

Corollary 8.1: A set $S \subset Lit$ is a relevance answer set of Π if and only if S^* is an answer set (stable model) of Π^*.

Proof: If S is consistent, then the theorem reduces to Theorem 8.2. Now, consider the case when S is inconsistent. Since relevance answer sets allow inconsistencies, by definition, the method of computing relevance answer sets is the same as that of computing minimal Herbrand models. Thus, the theorem is proved. □

8.2. Computing Stable Models

As mentioned in previous section, a general logic program can be assumed to be a set of ground clauses for defining stable models. Since each ground atom can be viewed as a proposition, in subsequent discussions, a general logic program P can be regarded as a finite[2] set of propositional clauses of the form

$$p \leftarrow p_1, \ldots, p_m, not\ q_1, \ldots, not\ q_n, \tag{8.1}$$

where p, each p_i, and each q_j are propositions. Each propositional clause is called a *rule* in this chapter. For convenience, each rule in P is associated with a unique natural number in front of it, and each of these rule numbers is used to stand for the corresponding rule.

The algorithm for computing stable models presented in the following is based on the notion of stratification [3, 129]. The algorithm is divided into five steps: Step 1 constructs a dependency graph G for a general logic program P. Step 2 simplifies the given logic program P and its dependency graph G into P' and G', respectively,

[2]As in the methods of computing stables models [92, 25], the set P of propositional clauses is also assumed to be finite.

such that the value of each node in G' completely depends on the values of the nodes incident to it. Step 3 finds the sets of even and odd negative loops in G'. Step 4 finds the set CUT of cuts, each of which contains a set of rules in the simplified program P' such that by removing them from G', there is no even negative loop. In Step 5, for each cut in CUT, remove the required rules in the odd negative loops from P' such that it becomes stratified. Then, check the iterated fixed point model of the stratified program to see whether it is a stable model of the original program P. In the following, we elaborate each step in detail.

8.2.1. Step 1: Constructing Dependency Graphs

Dependency graphs are a useful tool for query processing in database systems [127], for finding minimal deduction graphs [48], and so on. In our algorithm for computing stable models, a dependency graph is modified such that each arc of it is labeled with a rule number and with either a positive ("+") or a negative ("−") sign as in the following definition.

Definition 8.1: Let P be a general logic program. The *dependency graph* $G = (V, E)$ of P is a directed graph consisting of two nonempty and finite sets V and E. V is the set of nodes corresponding to the propositions in P. E is the set of arcs of the form $\langle p, q, s, r\rangle$ (indicating an arc incident from p to q with label r and sign s, where both p and q are in V and rule r is in P) which is generated as follows: Given a rule r in P of the form (8.1), each arc $\langle p_i, p, +, r\rangle$ (for $i = 1, \ldots, m$) in E is called a *positive arc* and each arc $\langle q_j, p, -, r\rangle$ (for $j = 1, \ldots, n$) in E is called a *negative arc.* □

The construction of the dependency graph G of P is shown in Algorithm 8.1, which is easily followed by definition. The arrays INDEG and OUTDEG store the *indegree* (the number of in-coming arcs) and *outdegree* (the number of out-going arcs) of each node in V, respectively. Note that the computation of the indegrees and outdegrees of nodes in V at lines 6-8 is needed for the simplification process discussed in Step 2.

Algorithm 8.1: *Constructing Dependency Graphs*

Input: A logic program P.

Output: The dependency graph $G = (V, E)$ of P associated with the arrays INDEG and OUTDEG.

Method:

```
1  let $V$ be the set of propositions occurred in $P$; $E = \emptyset$;
2  **for** each rule $r : p \leftarrow p_1, \ldots, p_m, not\ q_1, \ldots, not\ q_n$ in $P$ **do begin**
3     **for** $i = 1$ **to** $m$ **do** $E = E \cup \{\langle p_i, p, r, \rangle\}$;
4     **for** $j = 1$ **to** $n$ **do** $E = E \cup \{\langle q_j, p, -r, \rangle\}$;
5  **end_for**;
6  **for** each node $v$ in $V$ **do** INDEG[$v$] = 0 **and** OUTDEG[$v$] = 0;
7  **for** each arc $\langle u, v, s, r\rangle$ in $E$ **do**
8     INDEG[$v$] = INDEG[$v$] + 1 **and** OUTDEG[$u$] = OUTDEG[$u$] + 1.
```

8.2.2. Step 2: Simplifying Logic Programs and Dependency Graphs

There are six cases in simplifying the dependency graph G and thus the logic program P. Let T be the set $\{p | p \leftarrow \text{ is in } P\}$ of true propositions of P. Now, consider the following cases:

(1) Suppose there is a true proposition p in T. Then, any rule in P with head p will be redundant in determining the truth value of p because p is already true. This implies that such a kind of rules can be deleted. In addition, any rule with *not* p in the body will be deleted by the stable model definition.

(2) Suppose there exists a rule of the form $u \leftarrow \ldots, v, \ldots$ with INDEG$[v] = 0$ and $v \in T$. Then, since v is true, the truth value of u in the rule will depend on the rest of the body. This implies that v can be removed from the body of the rule.

(3) Suppose there exists a rule of the form $u \leftarrow \ldots, v, \ldots$ with INDEG$[v] = 0$ and $v \notin T$. Then, since v is *unknown*, u is impossible derived from the rule. So, the rule can be deleted.

(4) Suppose there exists a rule of the form $u \leftarrow \ldots, \mathit{not}\ v, \ldots$ with INDEG$[v] = 0$ and $v \in T$. Then, since v is true, the rule will be deleted by the stable model definition.

(5) Suppose there exists a rule of the form $u \leftarrow \ldots, \mathit{not}\ v, \ldots$ with INDEG$[v] = 0$ and $v \notin T$. Then, since v is *unknown*, by definition, *not* v can be removed from the body of the rule.

(6) Suppose there exists a rule with head v, non-empty body, and OUTDEG$[v] = 0$. Then, the rule is of no use in finding loops. So, it will be temporarily removed and put in the set REM for later use in Algorithm 8.5.

In the above six cases, case (1) deletes redundant rules, cases (2)-(5) remove the nodes with indegrees being zero from G, and case (6) removes the nodes with outdegrees being zero from G.

The simplification process is shown in Algorithm 8.2. Case (1) is performed at lines 2-3, especially, by the procedure **Remove**. Since both cases (3) and (4) involve the deletion of rules, they are performed at lines 7-8, especially, by the procedure **Update**. Similarly, since both cases (2) and (5) involve the elimination of redundant propositions in the bodies of rules, they are executed at lines 9-17. Finally, case (6) of removing nodes with outdegree being zero is performed at lines 19-26. Note that the set REM contains the simplified rules in P' which are temporarily removed in case (6).

Algorithm 8.2: *Simplifying Logic Programs and Dependency Graphs*

Input: A logic program P and its dependency graph $G = (V, E)$ with the arrays INDEG and OUTDEG.

Output: The simplified program P', its simplified dependency graph $G' = (V', E')$, the set REM of removed rules, and the set T of true propositions.

Method:

```
1  $P' = P$; $V' = V$; $E' = E$; REM = $\emptyset$;
2  $T = \{p | p \leftarrow$ is in $P'\}$;
3  for each $p$ in $T$ do call Remove($p$); /* case (1) */
4  while there exists a node $v$ in $V'$ such that INDEG[$v$] = 0 do begin
5    $V' = V' - \{v\}$;
6    while there exists an arc $\langle v, u, s, r \rangle$ in $E'$ do
7      if ($s$ = "+" and $v \notin T$) or ($s$ = "−" and $v \in T$) then
8        call Update($r$) /* cases (3) and (4) */
9      else begin /* cases (2) and (5) */
10       $E' = E' - \{\langle v, u, s, r \rangle\}$; INDEG[$u$] = INDEG[$u$] − 1;
11       if $s$ = "+" then delete $v$ from the body of rule $r$ in $P'$
12       else begin
13         delete *not* $v$ in the body of rule $r$ in $P'$;
14         if rule $r$ is of the form $u \leftarrow$ in $P'$ then
15           $T = T \cup \{u\}$ and call Remove($u$)
16       end_if
17     end_if
18 end_while;
19 while there exists a node $v$ in $V'$ such that OUTDEG[$v$] = 0
20 do begin /* case (6) */
```

```
21      $V' = V' - \{v\}$;
22      for each arc $\langle u, v, s, r\rangle$ in $E'$ do begin
23          $E' = E' - \{\langle u, v, s, r\rangle\}$; OUTDEG[$u$] = OUTDEG[$u$] $- 1$;
24          delete rule $r$ from $P'$; REM = REM $\cup$ $\{r\}$
25      end_for
26  end_while.
```

```
procedure Remove($p$);
begin
1  while there exists a rule $r$ in $P'$ with head $p$ do
2      call Update($r$);
3  while there exists a rule $r$ in $P'$ containing $not\ p$ in the body do
4      call Update($r$)
end_procedure.
```

```
procedure Update($r$);
begin
1  let rule $r$ be of the form $p \leftarrow p_1, \ldots, p_m, not\ q_1, \ldots, not\ q_n$;
2  $E' = E' - \{\langle p_1, p, +, r\rangle, \ldots, \langle p_m, p, +, r\rangle, \langle q_1, p, -, r\rangle, \ldots, \langle q_n, p, -, r\rangle\}$;
3  INDEG[$p$] = INDEG[$p$] $- m - n$;
4  delete rule $r$ from $P'$
end_procedure.
```

It is easy to see that the simplified dependency graph $G' = (V', E')$ has the property that the indegree and outdegree of each node in V' is non-zero. In addition, it is easy to show that G' is a set of strongly connected components. As we can see, the truth value of each node in V' completely depends on the truth values of the nodes incident to it. Note that the set REM of rules is a stratified logic program because the proposition in the head of each rule in REM has outdegree being zero.

8.2.3. Step 3: Finding Negative Loops

Definition 8.2: A *negative loop* in the simplified dependency graph G' is defined to be a loop including at least one negative arc. An *even* or *odd* negative loop is a negative loop with even or odd number of negative arcs, respectively. □

The method of finding negative loops is based on performing depth-first search on each node of G', which is shown in Algorithm 8.3. In the algorithm, the required data structures are described in the following:

(a) ELOOP and OLOOP are global variables which are sets containing all the even and odd negative loops in G', respectively.

(b) NEG is a global variable which is a set containing all the labels of the negative arcs occurring in even negative loops of ELOOP.

(c) RULE is also a global variable which is a set containing all the labels of the arcs occurring in even negative loops of ELOOP.

(d) MARK is a global array used to mark each visited node.

(e) ARC (or Arc in **Find_Loops**) is a local array used to store the arcs in a loop.

(f) Each element (i.e., a loop) in ELOOP or OLOOP is represented by a three-tuple of the form $(neg, pos, loop)$ where *pos* and *neg* are sets of labels of the positive and negative arcs occurring in a loop, respectively, and *loop* is the set of arcs in the loop.

The depth-first search, the central part of the algorithm, is implemented by the procedure **Find_Loops**. Note that $|V|$ is used to denote the cardinality of set V. Since the procedure **Find_Loops** will be performed on each node in V, some subgraphs of G' will be repeatedly searched. Owing to this reason, a table can be used to store all the possible paths from a node to another found so far to avoid redundant search. Note that in procedure **Find_Loops**, a "_" is used to denote a field in an arc whose value is not interested.

Algorithm 8.3: *Finding Negative Loops*

Input: A simplified dependency graph $G' = (V', E')$.

Output: A set ELOOP of even negative loops in G', a set OLOOP of odd negative loops in G', a set NEG of labels of the negative arcs occurring in even negative loops, and a set RULE of labels of the arcs occurring in even negative loops.

Method:

```
1  ELOOP = $\emptyset$; OLOOP = $\emptyset$; NEG = $\emptyset$; RULE = $\emptyset$;
2  for each node $v$ in $V'$ do MARK[$v$] = 0; /* MARK is a global array */
3  for $i$ = 1 to $|V'|$ do ARC[$i$] = 0; /* ARC is a local array */
4  for each node $v$ in $V'$ do
5     if MARK[$v$] = 0 do MARK[$v$] = 1 and call Find_Loops($v$, ARC, 1);
6  for each loop $(neg, pos, loop)$ in ELOOP do
7     NEG = NEG $\cup$ $neg$ and RULE = RULE $\cup$ $neg \cup pos$.
```

```
procedure Find_Loops($v$, Arc, $i$);
begin
1  for each arc $\langle v, u, s, r\rangle$ in $E'$ do begin
2     MARK[$u$] = 1; Arc[$i$] = $\langle v, u, s, r\rangle$;
3     if there exists an $l$ such that $1 \leq l \leq i$ and Arc[$l$] = $\langle u, \_, \_, \_\rangle$ then begin
4        $pos = \emptyset$; $neg = \emptyset$; $loop = \emptyset$;
5        for $k = l$ to $i$ do begin
6           let Arc[$k$] be of the form $\langle p, q, s', r'\rangle$; $loop = loop \cup \{\langle p, q, s', r'\rangle\}$;
```

```
            **if** $s' = $ "+" **then** $pos = pos \cup \{r'\}$ **else** $neg = neg \cup \{r'\}$
        **end_for**;
        **if** $|neg| > 0$ **then** /* a negative loop */
            **if** $|neg|$ is even **then** ELOOP = ELOOP $\cup$ $\{(neg, pos, loop)\}$
            **else** OLOOP = OLOOP $\cup$ $\{(neg, pos, loop)\}$
       **end_then**
       **else** Arc[$i$] = $\langle v, u, s, r\rangle$ **and call Find_Loops**($u$, Arc, $i+1$)
    **end_for**;
**end_procedure**.
```

8.2.4. Step 4: Finding Cuts

Definition 8.3: In the simplified dependency graph G', a *cut* is defined to be a subset C of labels of NEG such that by deleting each arc with label in C from G', there does not exist any even negative loop. □

Note that some odd negative loops in OLOOP may be removed because of the cuts. The idea of finding cuts is based on the strategy of backtracking, which is shown in Algorithm 8.4, by the main procedure **Find_Cuts**. CUT is a global variable which is a set containing all the cuts in G'. Each element of CUT is a three-tuple of the form (*cut*, OR, OL) indicating that *cut* is a cut, OR is the set of labels of arcs in the removed odd negative loops by *cut*, and OL is the set of remaining odd negative loops after *cut* is found. Note that even though there is no even negative loop, procedure **Find_Cuts** still generates the empty cut, ($\emptyset$, $\emptyset$,OL).

Algorithm 8.4: *Finding Cuts*

Input: A set ELOOP of even negative loops, a set OLOOP of odd negative loops, and a set NEG of labels of the negative arcs occurring in even negative loops.

Output: A set CUT of cuts.

Method:

```
1  CUT = $\emptyset$;
2  call Find_Cuts($\emptyset$, $\emptyset$, ELOOP, OLOOP, NEG).


procedure Find_Cuts($cut$, OR, EL, OL, Neg);
begin
1  if EL = $\emptyset$ then CUT = CUT $\cup$ {($cut$, OR, OL)}
2  else for each rule number $r$ in Neg do begin
3    if there exists a loop $(neg, pos, loop)$ in EL such that $r \in pos \cup neg$ do begin
4      $cut = cut \cup \{r\}$; $R = \emptyset$;
5      for each loop $(n1, p1, l1)$ in OL do /* remove odd negative loops */
6        if $r \in n1 \cup p1$ then OL = OL $-$ $\{(n1, p1, l1)\}$ and OR = OR $\cup\, n1 \cup p1$;
7      for each loop $(n2, p2, l2)$ in EL do /* remove even negative loops */
8        if $r \in n2 \cup p2$ then EL = EL $-$ $\{(n2, p2, l2)\}$ and $R = R \cup n2$;
9      for each loop $(n3, p3, l3)$ in EL do /* update the set $n3$ */
10         $n3 = n3 - R$ and update the value of $n3$ in the loop
11    end_if;
12    call Find_Cuts($cut$, OR, EL, OL, Neg $-$ $\{r\}$)
13  end_for;
end_procedure.
```

8.2.5. Step 5: Generating Stable Models

Recall that RULE is the set of rule numbers (or labels) occurred in even negative loops of OLOOP. For each cut (cut, OR, OL) in CUT, it is easy to see that (RULE $\cup$ OR $-$ cut) forms a stratified logic program. Its iterated fixed point model N can easily be found. Now, consider the following cases to further remove odd negative loops in OL based on model N:

(1) Suppose there exists an arc $\langle p, q, s, r\rangle$ in an odd negative loop of OL such that q is in N (i.e., there exists a rule r of the form $q \leftarrow \ldots, (not)\ p, \ldots$). Since q is in N, the rule is of no use in determining the truth value of q.

(2) Suppose there exists an arc $\langle p, q, s, r\rangle$ in an odd negative loop of OL such that p is in N and $s =$ "$-$" (i.e., there exists a rule r of the form $q \leftarrow \ldots, not\ p, \ldots$). Since p is in N, the rule will be deleted by definition.

Let *oddcut* be the set of rule numbers to be deleted in the above two cases. Let Q be the set of rule numbers occurring in the deleted odd negative loops of OL. Recall that REM is the set of simplified rules removed in Algorithm 8.2. Now, we can see that the set $P''' = (\text{RULE} \cup \text{REM} \cup \text{OR} \cup Q) - (cut \cup oddcut)$ is a stratified logic program. Let M be the iterated fixed point model of P'''. Then, check $(M \cup T)$ to see whether it is a stable model of the original program P, where T is the set of true propositions obtained from Algorithm 8.2. The above procedure of computing stable models is shown in Algorithm 8.5.

Algorithm 8.5: *Generating Stable Models*

Input: A general logic program P, the set CUT of cuts for P', the set REM of removed rules, the set T of true propositions, and the set RULE of labels occurred in even negative loops.

Output: Either the set S of stable models or "no stable model."

Method:

```
1  $S = \emptyset$;
2  for each cut ($cut$, OR, OL) in CUT do begin
3      OL1 = $\emptyset$; $oddcut = \emptyset$; $Q = \emptyset$;
4      while OL $\neq$ OL1 do begin
5          OL1 = OL; $P'' = (\text{RULE} \cup \text{OR} \cup Q) - (cut \cup oddcut)$;
```

```
6      let $N$ be the iterated fixed point model of $P''$;
7      for each loop $(neg, pos, loop)$ in OL do begin
8        for each arc $\langle p, q, s, r\rangle$ in $loop$ do
9          if $(q \in N)$ or $(p \in N$ and $s =$ "$-$") then $oddcut = oddcut \cup \{r\}$;
10        if $|oddcut| > 0$ then OL $=$ OL $- \{(neg, pos, loop)\}$ and $Q = Q \cup neg \cup pos$
11     end_for
12    end_while;
13    $P''' =$ (RULE $\cup$ REM $\cup$ OR $\cup$ $Q) - (cut \cup oddcut)$;
14    let $M$ be the iterated fixed point model of $P'''$;
15    if $(M \cup T)$ is a stable model of $P$ then $S = S \cup \{M \cup T\}$
16  end_for;
17  if $S = \emptyset$ then print("no stable model").
```

As we can see in Algorithm 8.5 of generating stable models of a general logic program P, each computed set of atoms is checked to see whether it is a stable model of P. Therefore, each model M in S of Algorithm 8.5 is a stable model of P. This soundness property of the algorithm is provided in the following theorem without proof.

Theorem 8.3 (Soundness): Let P be a general logic program. Let S be the set of models computed in Algorithm 8.5. Then, each computed model in S is a stable model of P. □

8.3. Examples of Computing Stable Models

Example 8.1: Consider the following general logic program P:

1. $nfly \leftarrow not\ ab1$,
2. $ab1 \leftarrow bird \wedge not\ ab2$,

$$\begin{tikzcd}
\mathsf{NFLY} \arrow[r, "8"] \arrow[dr, "7"] & \mathsf{NBIRD} \arrow[ddd, bend left=40, "9"] \\
\mathsf{AB1} \arrow[u, "1", "|" marking] & \mathsf{AB2} \arrow[l, "2", "|" marking] \arrow[u, "8", "|" marking] \arrow[d, "3", "|" marking] \\
\mathsf{BIRD} \arrow[u, "2"] \arrow[ur, "7"] \arrow[r, "3"] & \mathsf{FLY} \\
\mathsf{CANARY} \arrow[u, "4"] \arrow[r, "9"] & \mathsf{AB3} \arrow[ul, "4", "/" marking]
\end{tikzcd}$$

Figure 8.1: The dependency graph of P

3. $fly \leftarrow bird \wedge not\ ab2$,

4. $bird \leftarrow canary \wedge not\ ab3$,

5. $canary \leftarrow$,

6. $nfly \leftarrow$,

7. $ab2 \leftarrow bird \wedge nfly$,

8. $nbird \leftarrow nfly \wedge not\ ab2$,

9. $ab3 \leftarrow canary \wedge nbird$.

In step 1, the corresponding dependency graph G of P is constructed, which is depicted in Figure 8.1, in which a positive arc is represented by $\rightarrow$, while a negative arc is denoted by $\not\rightarrow$. In step 2, $T = \{canary,\ nfly\}$, and rules 5 and 6 are deleted because their heads are true. Now, since both *canary* and *nfly* have indegrees being zero, rules 4, 7, 8, and 9 are simplified, respectively, into

10. $bird \leftarrow not\ ab3$,

11. $ab2 \leftarrow bird$,

12. $nbird \leftarrow not\ ab2$,

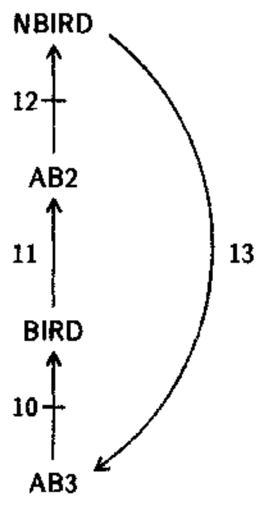

Figure 8.2: The simplified dependency graph of P

13. $ab3 \leftarrow nbird.$

Similarly, since both $ab1$ and fly have outdegrees being zero, rules 2 and 3 are temporarily removed and are put in the set REM. The simplified program P' contains rules 10, 11, 12, and 13, and its corresponding simplified dependency graph G' is shown in Figure 8.2. In step 3, only an even negative loop *loop* is found in G', which contains two positive arcs (defined by rules 11 and 13) and two negative arcs (defined by rules 10 and 12 containing *not*) as follows:

$$loop = \{\langle ab3, bird, -, 10\rangle, \langle bird, ab2, +, 11\rangle, \langle ab2, nbird, -, 12\rangle,$$
$$\langle nbird, ab3, +, 13\rangle\}.$$

Note that the set RULE = $\{10, 11, 12, 13\}$ contains the rules occurring in the only even negative loop *loop*. In step 4, there are two different cuts: $cut_1 = \{10\}$ and $cut_2 = \{12\}$ both with OR = $\emptyset$ and OL = $\emptyset$. In step 5, stable models is generated as follows: For the cut cut_1, the iterated fixed point model of RULE $-$ $cut_1 = \{11, 12, 13\}$ is $N_1 = \{nbird,\ ab3\}$. By considering the set REM = $\{2, 3\}$ of removed rules, P has a stable model $M_1 = \{canary,\ nfly,\ nbird,\ ab3\}$. Similarly, for the cut cut_2, P has

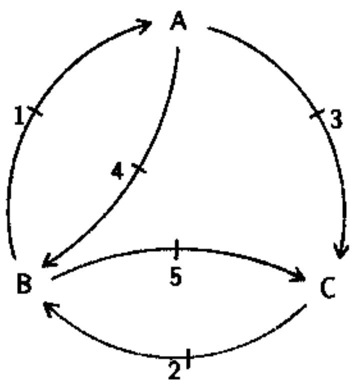

Figure 8.3: The dependency graph of P

another stable model $M_2 = \{canary, nfly, bird, ab2\}$. □

Example 8.2: Consider the following general logic program P:

1. $a \leftarrow not\ b$,

2. $b \leftarrow not\ c$,

3. $c \leftarrow not\ a$,

4. $b \leftarrow not\ a$,

5. $c \leftarrow not\ b$.

In step 1, the corresponding dependency graph G of P is constructed, which is shown in Figure 8.3. In step 2, since $T = \emptyset$ and each node in V has non-zero indegree and outdegree, P is already in the simplified form. In step 3, there are two even negative loops in ELOOP:

$$loop_1 = \{\langle a, b, -, 4\rangle, \langle b, a, -, 1\rangle\},$$

$$loop_2 = \{\langle b, c, -, 5\rangle, \langle c, b, -, 2\rangle\},$$

and one odd negative loop in OLOOP:

$$loop_3 = \{\langle a, c, -, 3\rangle, \langle c, b, -, 2\rangle, \langle b, a, -, 1\rangle\}.$$

Here, RULE = $\{1, 2, 4, 5\}$ is the set of rules occurring in ELOOP. In step 4, there are four possible cuts:

$cut_1 = \{1,2\}$ with OL $= \emptyset$ and OR $= \{3\}$,

$cut_2 = \{1,5\}$ with OL $= \emptyset$ and OR $= \{2,3\}$,

$cut_3 = \{2,4\}$ with OL $= \emptyset$ and OR $= \{1,3\}$,

$cut_4 = \{4,5\}$ with OR $= \{loop_3\}$ and OR $= \emptyset$.

In step 5, it is easy to see that both cut_1 and cut_2 generate the same stable model $M_1 = \{b,c\}$ of P, and cut_3 generates a stable model $M_2 = \{a,c\}$ of P. Now, consider the case of cut cut_4. The iterated fixed point model of the set RULE $-$ $cut_4 = \{1,2\}$ is $N = \{b\}$. Then, since b is in N, the arcs $\langle c,b,-,2\rangle$ and $\langle b,a,-,1\rangle$ in the negative loop $loop_3$ are deleted, and $oddcut$ becomes $\{1,2\}$ and Q becomes $\{1,2,3\}$. Now, the set RULE $\cup\ Q - (cut \cup oddcut) = \{3\}$ has the iterated fixed point model $M_3 = \{c\}$, which is not a stable model of P. Therefore, P has two stable models M_1 and M_2. □

8.4. Computing Answer sets

As assumed in [34], an extended logic program Π can be regarded as a finite set of ground clauses. In this section, an algorithm of computing answer sets of an extended logic program Π is sketched below.

(1) Using the renaming procedure to transform Π into a general logic program Π^* (in Section 8.1).

(2) Computing the stable models of Π^* (in Section 8.2).

(3) For each stable model M^* of Π^*, if M^* is consistent (i.e., it does not contain any pair of propositions of the forms p and no_p), then M is an answer set of Π.

(4) If M^* is inconsistent and $M^* = Lit^*$, then M is an answer set of Π.

CHAPTER 9

CONCLUSION AND FUTURE RESEARCH

The major contribution of this dissertation is to investigate how extended logic programs recently proposed by Gelfond and Lifschitz [34] can be used as a powerful knowledge representation tool to formalize commonsense reasoning. In this investigation, we first use extended logic programs to formalize an important knowledge structure in semantic networks called inheritance hierarchies with exceptions by adopting McCarthy's simple abnormality formalism [71] to express uncertain knowledge. In addition, we also proposed a new network representation based on Etherington and Reiter's network notation [28, 26, 27]. In our formalization, not only the conventional credulous reasoning [28, 124] can be performed but also the ambiguity blocking inheritance [50, 51] and the ambiguity propagating inheritance [126, 120] in the newly proposed skeptical reasoning [50] are simulated.

Stimulated by the anomalous extension problem of the famous YSP, we investigate the intuition behind commonsense reasoning and discover that commonsense reasoning performed by human being is a kind of forward reasoning. Its common inference pattern is reasoning from given information to derive some intuitive conclusions or properties. Thus, commonsense reasoning can be viewed as a reasoning of intuitive expectation. One important property of this reasoning is its unidirectional nature of inference as observed by Morris [86]. We then examine several proposals of reformulations of the YSP to exclude the anomalous extension and find out that these proposals are all based on this property.

Since in commonsense reasoning, some conclusions are assumed by default, but almost every formalization of nonmonotonic reasoning does not distinguish this difference. We first identify the defeasible conclusions of our formalization just based on the simple syntax of extended logic programs and then apply similar idea to other formalizations to achieve this purpose.

As mentioned earlier, commonsense reasoning can be viewed as a reasoning of intuitive expectation. However, sometimes it may happen that a new fact contradicts what we expect. In this case, our formalization and the reformulations mentioned above cannot deal with this situation. Along the line of Giordano and Martelli [42], we propose the generalized answer set semantics to resolve conflicts by capturing the idea of dependency-directed backtracking in TMSs [21]. Also, similar approaches are applied to the above reformalizations to resolve conflicting information.

Although we adopt McCarthy's abnormality formalism to express uncertain knowledge, we discover that this formalism may have the semantic inconsistency problem in interpreting conclusions and propose a simple way to fix it. We then explore an interesting property in commonsense reasoning called "unless" semantics and enhance the generalized answer set semantics to incorporate this important property. It turns out that a clause in an extended logic program can be transformed into one of other four forms depending on the given information, and the transformed programs become more coherent and their conclusions become more reasonable.

Because the definition of answer sets of an extended logic program is a generalization of that of stable models of a general logic program, both of them are not constructive. We first present a new approach to compute stable models of a general logic program. Then, we propose a method to compute answer sets based on the computation of stable models. In this way, the formalization of commonsense knowledge based on extended logic programs becomes more practical.

It is noted that our formalization of commonsense knowledge is based on the new knowledge representation tool of extended logic programs. Note that each clause in an extended logic program is an inference rule. In our approach, it takes advantage of the unidirectional nature of clauses in extended logic programs to exclude anomalous extensions. However, it cannot have the full power of first-order logic. In order to increase the inference ability of extended logic programs, we can make the following extension: A clause containing negation-as-failure *not* in its body is really an inference rule because it will generate a defeasible conclusion by assumption, whereas a clause not containing *not* can be regarded as a first-order axiom because it represents a piece of certain information. In this case, each clause containing *not* can be transformed into a default in default logic [34], and each clause not containing *not* can be viewed as an axiom in first-order logic. Thus, an extended logic program can be transformed into a default theory to have full power of first-order logic reasoning ability. For example, suppose a given commonsense knowledge contains a piece of information "dogs are mammals" which can be expressed by the clause

$$mammal(X) \leftarrow dog(X).$$

If we later come to know that Tweety is not a mammal, then we cannot derive that Tweety is not a dog by regarding the above clause as an inference rule, but we can derive this conclusion through contraposition by viewing the above clause as a first-order axiom.

In the following, we would like to highlight the issues of possible future research.

- As discussed above, extended logic programs can be transformed into a class of default theories. It is interesting to investigate whether any default theory can also be transformed into an extended logic program.
- In previous developments, we see that there exists a rough correspondence between the nonmonotonic operators *not* (in extended logic programs), *out* (in TMSs), $\neg L$ (in AEL), and $M\neg$ (in default logic). This correspondence reveals

that these formalizations have some closed connections. It is interesting to find the relationships between these formalizations.

- It is known that Yang's deduction graph (DG) [48, 136, 137] is a powerful inference tool and is successfully applied to database and logic query processing [134, 135], better relational database schemes design [136], integrity constraint checking [49], probabilistic reasoning [61, 62], abductive reasoning [61], and default reasoning [88]. In future research, we want to investigate how DGs can also be used to perform commonsense reasoning based on our formalization.
- As we can see, the head of a clause in extended logic programs only allow to contain a literal. Similar to disjunctive logic programs [6, 7, 69, 79, 100, 101, 121] which are a generalization of general logic programs by allowing a disjunction of atoms in the head of a clause, in future research, we also want to extend extended logic programs to allow a disjunction of literals in the head of a clause.

BIBLIOGRAPHY

1. American Association for Artificial Intelligence, *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA (1986).

2. American Association for Artificial Intelligence, *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA (1987).

3. Apt, K.R., Blair, H.A., and Walker, A., Towards a theory of declarative knowledge, in: J. Minker (Ed.), *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann Publishers, Los Altos, CA, 1988) 89–148.

4. Apt, K.R. and van Emden, M.H., Contributions to the theory of logic programming, *Journal of the ACM* **29** (3) (1982) 841–862.

5. Baker, A.B. and Ginsberg, M.L., A theorem prover for prioritized circumscription, in: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI (1989) 463–467.

6. Baral, C., Lobo, J., and Minker, J., Generalized disjunctive well-founded semantics for logic programs: Procedural semantics, in: *Proceedings of the Fifth International Symposium on Methodologies for Intelligent Systems* (North-Holland, New York, 1990) 456–464.

7. Baral, C., Lobo, J., and Minker, J., Generalized disjunctive well-founded semantics for logic programs: Declarative semantics, in: *Proceedings of the Fifth International Symposium on Methodologies for Intelligent Systems* (North-Holland, New York, 1990) 465–473.

8. Bidoit, N. and Froidevaux, C., General logical databases and programs: Default logic semantics and stratification, *Information and Computation* **91** (1) (1991) 15–54.

9. Bobrow, D. and Winograd, T., An overview of KRL, a knowledge representation language, *Cognitive Science* **1** (1) (1977) 3-46.

10. Brachman, R.J. and Schmolze, J., An overview of the KL-one knowledge representation system, *Cognitive Science* **9** (2) (1985) 171–216.

11. Brachman, R.J., I lied about the trees, or defaults and definitions in knowledge representation, *The AI Magazine* **6** (3) (1985) 80–93.

12. Cercone, N. and Schubert, L., Toward a state based conceptual representation, in: *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, USSR (1975) 83–90.

13. Chandra, A.K. and Harel, D., Horn clause queries and generalizations, *Journal*

of Logic Programming **2** (1) (1985) 1–15.

14. Charniak, E., Riesbeck, C.K., and McDermott, D.V., *Artificial Intelligence Programming* (Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ, 1980).

15. Charniak, E., A common representation for problem-solving and language-comprehension information, *Artificial Intelligence* **16** (1981) 225–255.

16. Clark, K.L., Negation as failure, in: H. Gallaire and J. Minker (Eds.), *Logic and Data Bases* (Plenum Press, New York, 1978) 293–322.

17. Clark, K.L., Predicate logic as a computational formalism, Research Report DOC 79/59, Department of Computing, Imperial College (1979).

18. Colmerauer, A., Kanoui, H., Roussel, P., and Pasero, R., *Un Système de Communication Homme-Machine en Français*, Groupe de Recherche en Intelligence Artificielle, Universitè d'Aix-Marseille, Marseille, France (1973).

19. Decker, H., Integrity enforcement on deductive databases, in: *Proceedings of the First International Conference on Expert Database Systems*, Charleston, SC (1986) 271–285.

20. de Kleer, J., An assumption-based TMS, *Artificial Intelligence* **28** (1986) 127–162.

21. Doyle, J., A truth maintenance system, *Artificial Intelligence* **12** (1979) 231–272.

22. Dung, P.M., On the strong completion of logic programs, in: *Proceedings of the Second International Conference on Algebraic and Logic Programming*, LNCS 463 (Springer-Verlag, Berlin, 1990) 158–172.

23. Dung, P.M. and Kanchanasut, K., On the generalized predicate completion of non-Horn program, in: *Logic Programming: Proceedings of the North American Conference 1989* (MIT Press, Cambridge, MA, 1989) 587–603.

24. Enderton, H.B., *A Mathematical Introduction to Logic* (Academic Press, New York, 1972).

25. Eshghi, K., Computing stable models by using the ATMS, in: *Proceedings of the Eighth National Conference on Artificial Intelligence* (MIT Press, Cambridge, MA, 1990) 272–277.

26. Etherington, D.W., Formalizing nonmonotonic reasoning systems, *Artificial Intelligence* **31** (1987) 41–85.

27. Etherington, D.W., *Reasoning with Incomplete Information* (Pitman Publishing, London, 1988).

28. Etherington, D.W. and Reiter, R., On inheritance hierarchies with exceptions, in: *Proceedings of the National Conference on Artificial Intelligence*, Washington, DC (1983) 104–108.

29. Fahlman, S., *NETL: A System for Representing and Using Real-World Knowledge* (MIT Press, Cambridge, MA, 1979).

30. Fahlman, S.E., Touretzky, D.S., and van Roggen, W., Cancellation in a parallel semantic network, in: *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, BC, Canada (1981) 257–263.

31. Fujiwara, Y. and Honiden, S., Relating the TMS to autoepistemic logic, in: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI (1989) 1199–1205.

32. Gelfond, M., Autoepistemic logic and formalization of commonsense reasoning: Preliminary report, in: *Proceedings of the Second International Workshop on Non-Monotonic Reasoning*, LNAI 346 (Springer-Verlag, Berlin, 1989) 176–186.

33. Gelfond, M. and Lifschitz, V., The stable model semantics for logic programming, in: *Logic Programming: Proceedings of the Fifth International Conference* (MIT Press, Cambridge, MA, 1988) 1070–1080.

34. Gelfond, M. and Lifschitz, V., Classical negation in logic programs and disjunctive databases, *New Generation Computing* **9** (3,4) (1991) 365–385.

35. Gelfond, M. and Przymusinska, H., Negation as failure: Careful closure procedure, *Artificial Intelligence* **30** (1986) 273–287.

36. Gelfond, M., Przymusinska, H., and Przymusinski, T., The extended closed world assumption and its relationship to parallel circumscription, in: *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Cambridge, MA, (1986) 133–139.

37. Gelfond, M., Przymusinska, H., and Przymusinski, T., On the relationship between circumscription and negation as failure, *Artificial Intelligence* **38** (1989) 75–94.

38. Genesereth, M.R. and Nilsson, N.J., *Logic Foundations of Artificial Intelligence* (Morgan Kaufmann Publishers, Los Altos, CA, 1987).

39. Ginsberg, M.L., Multivalued logics: A uniform approach to reasoning in artificial intelligence, *Computational Intelligence* **4** (1988) 265–316.

40. Ginsberg, M.L., A circumscriptive theorem prover: Preliminary report, in: *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN (1988) 470–474.

41. Ginsberg, M.L., A circumscription theorem prover, *Artificial Intelligence* **39** (1989) 209–230.

42. Giordano, L. and Martelli, A., Generalized stable models, truth maintenance and conflict resolution, in: *Logic Programming: Proceedings of the Seventh International Conference* (MIT Press, Cambridge, MA, 1990) 427–441.

43. Goldstein, I. and Roberts, R.B., Nudge: A knowledge-based scheduling program, in: *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA (1977).

44. Hanks, S. and McDermott, D., Default reasoning, nonmonotonic logics, and the frame problem, in: *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA (1986) 328–333.

45. Hanks, S. and McDermott, D., Nonmonotonic logic and temporal projection, *Artificial Intelligence* **33** (1987) 379–412.

46. Hayes, P.J., The logic of frames, in: D. Metzing (Ed.), *Frame Conception and Text Understanding* (Walter de Gruyter, Berlin, 1980) 46–61.

47. Hill, R., LUSH-resolution and its completeness, DCL Memo 78, Department of Artificial Intelligence, University of Edinburg (1974).

48. Horng, W.B. and Yang, C.C., A shortest path algorithm to find minimal deduction graphs, *Data and Knowledge Engineering* **6** (1) (1991) 27–46.

49. Horng, W.B. and Yang, C.C., An application of deduction graphs to integrity constraint checking in deductive databases, Technical Report #N-91-001, Department of Computer Science, University of North Texas, Denton, TX (1991).

50. Horty, J.F., Thomason, R.H., and Touretzky, D.S., A skeptical theory of inheritance in nonmonotonic semantic networks, in: *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA (1987) 358–363.

51. Horty, J.F., Thomason, R.H., and Touretzky, D.S., A skeptical theory of inheritance in nonmonotonic semantic networks, *Artificial Intelligence* **42** (1990) 311–348.

52. Hou, B.H., Togashi, A., and Noguchi, S., Generalized predicate completion and its relation to circumscription, *Journal of Japanese Society for Artificial Intelligence* **5** (3) (1990) 343–353.

53. Kleene, S.C., *Introduction to Metamathematics* (Van Nostrand, New York, 1952).

54. Konolige, K., On the relation between default and autoepistemic logic, *Artificial Intelligence* **35** (1988) 343–382.

55. Kowalski, R.A., Predicate logic as a programming language, in: *Information Processing: Proceedings of the IFIP-74*, Stockholm, North Holland (1974) 569–574.

56. Kowalski, R.A., Algorithm = logic + control, *Communications of the ACM* **22** (7) (1979) 424–436.

57. Kowalski, R.A., *Logic for Problem Solving* (North-Holland, Amsterdam, 1979).

58. Kowalski, R.A. and Sadri, F., Logic programs with exceptions, *New Generation*

Computing **9** (3,4) (1991) 387–400.

59. Levesque, H.J., All I know: An abridged report, in: *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA (1987) 426–431.

60. Levesque, H.J., All I know: A study in autoepistemic logic, *Artificial Intelligence* **42** (1990) 263–309.

61. Li, H.L. and Yang, C.C., Abductive reasoning by constructing probabilistic deduction graphs for solving the diagnosis problem, *Decision Support Systems* **7** (1991) 121–131.

62. Li, H.L., Yang, C.C., and Horng, W.B., Probabilistic deductive and abductive reasoning by using probabilistic deduction graphs and integer programming, in: *Proceedings of the Fourth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, Kauai, Hawaii (1991) 618–627.

63. Lifschitz, V., Computing circumscription, in: *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Las Angeles, CA (1985) 121–127.

64. Lifschitz, V., Pointwise circumscription: Preliminary report, in: *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA (1986) 406–410.

65. Lifschitz, V., Between circumscription and autoepistemic logic, in: *Proceedings of the First International Conference on Principles of Knowledge Representation*, Toronto, Ontario, Canada (1989) 235–244.

66. Lin, F., Reasoning in the presence of inconsistency, in: *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA (1987) 139–143.

67. Lloyd, J.W., *Foundations of Logic Programming* (Springer-Verlag, Berlin, 1987).

68. Lobo, J., Minker, J., and Rajasekar, A., Weak completion theory for non-Horn programs, in: *Logic Programming: Proceedings of the Fifth International Conference* (MIT Press, Cambridge, MA, 1988) 828–842.

69. Lobo, J., Minker, J., and Rajasekar, A., Extending the semantics of logic programs to disjunctive logic programs, in: *Logic Programming: Proceedings of the Sixth International Conference* (MIT Press, Cambridge, MA, 1989) 255–267.

70. McCarthy, J., Circumscription—A form of non-monotonic reasoning, *Artificial Intelligence* **13** (1980) 27–39.

71. McCarthy, J., Applications of circumscription to formalizing common-sense knowledge, *Artificial Intelligence* **28** (1986) 89–116.

72. McDermott, D., Nonmonotonic logic II: Nonmonotonic modal theories, *Journal of the ACM* **29** (1) (1982) 33–57.

73. McDermott, D. and Doyle, J., Non-monotonic logic I, *Artificial Intelligence* **13** (1980) 41–72.

74. Michalski, R.S., A theory and methodology of inductive learning, in: R.S. Michalski et al. (Eds.), *Machine Learning* (Tioga Publishing, Palo Alto, CA, 1983) 83–129.

75. Minker, J., On indefinite databases and the closed world assumption, in: *Proceedings of the Sixth Conference on Automated Deduction*, LNCS 138 (Springer-Verlag, Berlin, 1982) 292–308.

76. Minker, J., Deductive databases: An overview of some alternative theories, in: Z.W. Ras and M. Zemankova (Eds.), *Proceedings of the Second International Symposium on Methodologies for Intelligent Systems*, Charlotte, NC (North-Holland, Amsterdam, 1987) 148–158.

77. Minker, J. (Ed.), *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann Publishers, Los Altos, CA, 1988).

78. Minker, J., Perspectives in deductive databases, *Journal of Logic Programming* **5** (1) (1988) 33–60.

79. Minker, J., Toward a foundation of disjunctive logic programming, in: *Logic Programming: Proceedings of the North American Conference 1989* (MIT Press, Cambridge, MA, 1989) 1215–1235.

80. Minker, J. and Perlis, D., Applications of protected circumscription, in: *Proceedings of the Seventh International Conference on Automated Deduction*, LNCS 170 (Springer-Verlag, Berlin, 1984) 414–425.

81. Minker, J. and Perlis, D., Protected circumscription, in: *Proceedings of AAAI Workshop on Non-Monotonic Reasoning*, New Paltz, NY (1984) 337–343.

82. Minker, J. and Perlis, D., Computing protected circumscription, *Journal of Logic Programming* **2** (4) (1985) 235–249.

83. Minsky, M., A framework for representing knowledge, in: P.H. Winston (Ed.), *The Psychology of Computer Vision* (McCraw-Hill Book Company, New York, 1975) 211–277.

84. Moore, R.C., Semantical considerations on nonmonotonic logic, *Artificial Intelligence* **25** (1985) 75–94.

85. Morris, P.H., Curing anomalous extensions, in: *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA (1987) 437–442.

86. Morris, P.H., The anomalous extension problem in default reasoning, *Artificial Intelligence* **35** (1988) 383–399.

87. Morris, P.H., Autoepistemic stable closures and contradiction resolution, in: *Proceedings of the Second International Workshop on Non-Monotonic Reason-*

ing, LNAI 346 (Springer-Verlag, Berlin, 1989) 60–73.

88. Munoz, R.A. and Yang, C.C., Using normal deduction graphs in default reasoning, in: *Proceedings of the Sixth International Symposium on Methodologies for Intelligent Systems* (North-Holland, New York, 1991).

89. Nilsson, N.J., Probabilistic logic, *Artificial Intelligence* **28** (1986) 71–87.

90. Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Morgan Kaufmann Publishers, San Mateo, CA, 1988).

91. Perlis, D., Autocircumscription, *Artificial Intelligence* **36** (1988) 223–236.

92. Pimentel, S.G. and Cuadrado, J.L., A truth maintenance system based on stable models, in: *Logic Programming: Proceedings of the North American Conference 1989* (MIT Press, Cambridge, MA, 1989) 274–290.

93. Plümer, L., *Termination Proofs for Logic Programs* (Springer-Verlag, Berlin, 1990).

94. Poole, D., A logical framework for default reasoning, *Artificial Intelligence* **36** (1988) 27–47.

95. Przymusinska, H. and Przymusinski, T.C., Weakly perfect model semantics for logic programs, in: *Logic Programming: Proceedings of the Fifth International Conference* (MIT Press, Cambridge, MA, 1988) 1106–1120.

96. Przymusinski, T.C., An algorithm to compute circumscription. *Artificial Intelligence* **38** (1989) 49–73.

97. Przymusinski, T.C., Non-monotonic formalisms and logic programming, in: *Logic Programming: Proceedings of the Sixth International Conference* (MIT Press, Cambridge, MA, 1989) 655–674.

98. Quilian, R., Semantic memory, in: M. Minsky (Ed.), *Semantic Information Processing* (MIT Press, Cambridge, MA, 1968) 216–270.

99. Rajasekar, A., Lobo, J., and Minker, J., Weak generalized closed world assumption, Technical Report CS-TR-1992, Department of Computer Science, University of Maryland, College Park (1987).

100. Rajasekar, A., Lobo, J., and Minker, J., Skeptical reasoning and disjunctive programs, in: *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning,* Toronto, Ontario, Canada (1989) 349–356.

101. Rajasekar, A. and Minker, J., A stratification semantics for general disjunctive programs, in: *Logic Programming: Proceedings of the North American Conference 1989* (MIT Press, Cambridge, MA, 1989) 573–586.

102. Raphael, B., SIR: A computer program for semantic information retrieval, in:

M. Minsky (Ed.), *Semantic Information Processing* (MIT Press, Cambridge, MA, 1968) 33–134.

103. Reinfrank, M., Dressler, O., and Brewka, G., On the relation between truth maintenance and autoepistemic logic, in: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI (1989) 1206–1212.

104. Reinfrank, M. and Freitag, H., Rules and justifications: A uniform approach to reason maintenance and non-monotonic inference, in: *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, Tokyo, Japan (1988) 439–446.

105. Reiter, R., On closed world data bases, in: H. Gallaire and J. Minker (Eds.), *Logic and Data Bases* (Plenum Press, New York, 1978) 55–76.

106. Reiter, R., A logic for default reasoning, *Artificial Intelligence* **13** (1980) 81–132.

107. Reiter, R., Nonmonotonic reasoning, in: *Annual Review of Computer Science*, volume 2 (Annual Reviews Inc., Palo Alto, CA, 1987) 147–186.

108. Reiter, R., and Criscuolo, G., Some representational issues in default reasoning, *Journal of Comput. Math. Appl.* **9** (1983) 1–13, (Special Issue on Computational Linguistics).

109. Roberts, R.B. and Goldstein, I.P., The FRL manual, Technical Report AI Memo 409, MIT, Cambridge, MA (1977).

110. Robinson, J.A., A machine-oriented logic based on the resolution principle. *Journal of the ACM* **12** (1) (1965) 23–41.

111. Rodi, W., A new algorithm for truth maintenance, in: *AI Systems in Government* (1989).

112. Roussel, P., *PROLOG: Manuel de Reference et d'Utilization*, Groupe d'Intelligence Artificielle, Universitè d'Aix-Marseille, Marseille, France (1975).

113. Schubert, L.K., Extending the expressive power of semantic networks, *Artificial Intelligence* **7** (1976) 163–198.

114. Shepherdson, J.C., Negation as failure: A comparison of Clark's completed data base and Reiter's closed world assumption, *Journal of Logic Programming* **1** (1) (1984) 51–79.

115. Shepherdson, J.C., Negation as failure II, *Journal of Logic Programming* **2** (3) (1985) 185–202.

116. Shepherdson, J.C., Negation in logic programming, in: J. Minker (Ed.), *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann Publishers, Los Altos, CA, 1988) 19–88.

117. Shoham, Y., Chronological ignorance: Time, nonmonotonicity, necessity and

causal theories, in: *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA (1986) 389–393.

118. Shvarts, G., Autoepistemic modal logics, in: *Proceedings of the Third Conference on Theoretical Aspects of Reasoning about Knowledge*, Pacific Grove, CA (1990) 97–109.

119. Stallman, R. and Sussman, G.J., Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, *Artificial Intelligence* **9** (1977) 135–196.

120. Stein, L.A., Skeptical inheritance: Computing the intersection of credulous extensions, in: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI (1989) 1153–1158.

121. Subrahmanian, V.S. and Minker, J., Completion semantics for general and disjunctive logic programs, in: *Proceedings of the Fifth International Symposium on Methodologies for Intelligent Systems* (North-Holland, New York, 1990) 545–552.

122. Togashi, A., Hou, B.H., and Noguchi, S., Generalized predicate completion, in: *Proceedings of the International Conference on Knowledge Based Computer Systems*, LNAI 444 (Springer-Verlag, Berlin, 1989) 286–295.

123. Togashi, A. and Noguchi, S., Prioritized predicate completion, in: *Pacific Rim International Conference on Artificial Intelligence '90*, Nagoya, Japan (1990).

124. Touretzky, D.S., Implicit ordering of defaults in inheritance systems, in: *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX (1984) 322–325.

125. Touretzky, D.S., *The Mathematics of Inheritance Systems* (Pitman Publishing, London, 1986).

126. Touretzky, D.S., Horty, J.F., and Thomason, R.H., A clash of intuitions: The current state of nonmonotonic multiple inheritance systems, in: *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milano, Italy (1987) 476–482.

127. Ullman, J.D., *Principles of Database and Knowledge-based Systems*, volume I (Computer Science Press, Rockville, MD, 1988).

128. van Emden, M.H. and Kowalski, R.A., The semantics of predicate logic as a programming language, *Journal of the ACM* **23** (4) (1976) 733–742.

129. Van Gelder, A., Negation as failure using tight derivations for general logic programs, in: J. Minker (Ed.), *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann Publishers, Los Altos, CA, 1988) 149–176.

130. Van Gelder, A., Ross, K., and Schlipf, J.S., Unfounded sets and well-founded semantics for general logic programs: Extended abstract, in: *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database*

Systems, Philadelphia, PA (1989) 221–230.

131. Woods, W.A., What's in a link: Foundations for semantic networks, in: D.G. Bobrow and A. Collins (Eds.), *Representation and Understanding: Studies in Cognitive Science* (Academic Press, New York, 1975) 35–82.

132. Yahya, A. and Henschen, L.J., Deduction in non-Horn databases, *Journal of Automated Reasoning* **1** (1985) 141–160.

133. Yang, C.C., *Relational Databases* (Prentice-Hall, Englewood Cliffs, NJ, 1986).

134. Yang, C.C., An algorithm for logically deducing Horn clauses and processing logic queries, *International Journal of Pattern Recognition and Artificial Intelligence* **1** (1) (1987) 157–168.

135. Yang, C.C., Extending deduction graphs for inferring and redundancy-checking function-free rules, in: *The Second International Symposium on Methodologies for Intelligent Systems*, Charlotte, NC (1987) presented at the Colloquial Program.

136. Yang, C.C., Deduction graphs: An algorithm and applications, *IEEE Transactions on Software Engineering* **15** (1) (1989) 60–67.

137. Yang, C.C., Chen, J.J.Y., and Chau, H.L., Algorithms for constructing minimal deduction graphs, *IEEE Transactions on Software Engineering* **15** (6) (1989) 760–770.

138. You, J.H. and Li, L., Supported circumscription and its relation to logic programming with negation, in: *Logic Programming: Proceedings of the North American Conference 1989* (MIT Press, Cambridge, MA, 1989) 291–309.

139. Zadeh, L.A., PRUF—A meaning representational language for natural languages, in: E.H. Mamdani and B.R. Gaines (Eds.), *Fuzzy Reasoning and Its Applications* (Academic Press, New York, 1981) 1–66.

140. Zadeh, L.A., The role of fuzzy logic in the management of uncertainty in expert systems, *Fuzzy Sets and Systems* **11** (1983) 199–227, also in: M.M. Gupta et al. (Eds.), *Approximate Reasoning in Expert Systems* (North-Holland, Amsterdam, 1985) 3–31.

141. Zadeh, L.A., Commonsense and fuzzy logic, in: N. Cercone and G. McCalla (Eds.), *The Knowledge Frontier: Essays in the Representation of Knowledge* (Springer-Verlag, Berlin, 1987) 103–136.